# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## FACULTAD DE INGENIERÍA

# IMPLEMENTACIÓN DE UN ALGORITMO PARA LA ASIGNACIÓN DE TRÁFICO VEHICULAR

**T E S I S**

QUE PARA OBTENER EL TÍTULO DE

**INGENIERO EN COMPUTACIÓN**

P R E S E N T A :

**GONZÁLEZ CABRERA HÉCTOR**

**DIRECTOR: DRA. ANGÉLICA DEL ROCÍO LOZANO CUEVAS**

**CIUDAD UNIVERSITARIA** **MÉXICO, D. F., 2012**

**A mis padres por todo su amor, trabajo y sacrificio.**

**Mi más sincero agradecimiento al**
**Laboratorio de Transporte y Sistemas Territoriales.**
**En especial a Angélica del Rocío Lozano Cuevas,**
**Luis Alejandro Guzmán Castro y**
**Gloria Elena Londoño Mejía.**

**Gracias por todo el apoyo y tiempo invertido en el presente trabajo.**

# Índice

# Introducción

En la presente introducción se expone en forma breve la descripción general del problema que ha dado origen a la creación de esta tesis. De igual forma, se proporcionan los antecedentes, justificación, objetivos y consideraciones de la investigación. Finalmente, se concluye con una reseña en la que se explica en forma general la estructura que posee la tesis y los aspectos más relevantes que serán abordados en los capítulos posteriores.

## Definición del problema

El mundo se urbaniza rápidamente y la densidad de población aumenta en forma considerable. Un informe de las Naciones Unidas calcula que aproximadamente el 70% de la población mundial vivirá en ciudades en el año 2050 (Handwerk, 2008). Este crecimiento conlleva una expansión de la demanda a la que están sometidas todas las infraestructuras urbanas, entre ellas, se encuentra el transporte (Houghton, Reiners y Lim, 2009).

Hoy en día, los viajes diarios en algunas de las ciudades económicamente más importantes del mundo son demasiados largos y extenuantes, lo cual refleja claramente las dificultades que existen en la infraestructura de transporte para mantener el ritmo de la actividad económica. El tiempo requerido para recorrer distancias relativamente cortas se ha elevado drásticamente. Por ejemplo, tan sólo en EE.UU. se pierden 3700 millones de horas cada año en el tráfico y se queman innecesariamente 2300 millones de galones de combustible, suficiente para llenar 58 súper cisternas, lo que tiene un costo para la economía de 78000 millones de dólares por año (IBM, 2010).

Las soluciones tradicionales, como la construcción de más carreteras o autopistas, no serán suficientes para superar el gran crecimiento del tráfico en las ciudades en rápido desarrollo

económico y poblacional. Por lo tanto, es preciso poner en marcha otro tipo de alternativas que permitan gestionar el flujo vehicular y evitar el colapso un las infraestructuras de transporte.

## Antecedentes

El mejoramiento de la red vial mediante la adición de nuevas vialidades o el incremento de la capacidad de una vialidad existente, se ha contemplado como la solución al problema del tráfico vehicular. Aparentemente, mejorar la red vial implica el mejoramiento del flujo en la misma; sin embargo, esto puede no ocurrir; inclusive, puede producir un aumento en el total de emisiones generadas. Esto se resume de forma clara en el siguiente enunciado de la paradoja de Braess, la cual afirma: *"El hecho de agregar una nueva vialidad a una red de transporte puede no mejorar la operación del sistema, en el sentido de la reducción del tiempo total de viaje en el sistema"* (Lozano y Antún, 2003, p.41).

Es por ello que, en los últimos años, se han desarrollado modelos para entender ampliamente el complejo desempeño del tráfico vehicular y, así, incrementar la capacidad vehicular de las vías y mejorar el manejo de los sistemas carreteros existentes. Se ha intentado representar el tráfico vehicular de acuerdo con la teoría hidrodinámica, mediante modelos de aproximaciones de fluidos de primer y segundo orden, pero dichos modelos solo han permitido representar situaciones muy simples del comportamiento del flujo vehicular. Se han desarrollado algunos modelos de simulación para representar el movimiento del flujo vehicular a nivel microscópico, pero estos modelos requieren gran cantidad de información para representar cada detalle de la infraestructura vial y de la demanda de transporte, información que frecuentemente no está disponible. También se han utilizado modelos de simulación para representar carreteras, pero el comportamiento vehicular que se describe en ellas es más simple que en las áreas urbanas (Lozano et al., 2003).

Otra tecnología recientemente incorporada al análisis de las redes de flujo es la de los sistemas de información geográfica para transporte. La representación y el análisis de redes de transporte han mejorado con el desarrollo de estos sistemas, pues además de permitir una representación topológica de la red, con ellos se puede hacer una representación geográfica de la misma. Y lo más importante, es que los análisis de redes pueden ser combinados con los análisis tradicionales de los sistemas de información geográfica, los cuales pueden considerar diversas características del territorio, tales como uso de suelos, crecimiento de la mancha urbana y datos de atributos demográficos o socioeconómicos. Con ello, surge la posibilidad de crear escenarios de tipo hipotético para analizar cambios en la infraestructura vial o en la demanda (Lozano et al., 2003).

## Justificación

Actualmente existen algunos paquetes de software que permiten analizar el flujo vehicular a nivel macroscópico mediante la utilización de algoritmos de asignación de tráfico. Sin embargo, la mayoría de ellos solo permite utilizar una función de costo de entre un conjunto de funciones predeterminadas y propias de países desarrollados. Dichas funciones pueden no ser las mejores para representar el flujo vehicular de ciudades de países como México, caso de estudio en el que es necesario emplear funciones distintas para los diferentes tipos de vialidad, ya que su operación y características no son similares.

Es por ello que la presente tesis busca obtener una estimación más realista del flujo vehicular mediante la implementación de un algoritmo de asignación de equilibrio que considere las diversas características de una red vial específica, haciendo distinción entre los diferentes tipos de arcos que conforman dicha red.

Para cumplir con esta función, se ha implementado el algoritmo Frank Wolfe (conocido como el método de optimización de combinaciones convexas), el cual utiliza una serie de funciones de tiempo de viaje (costo), entre las que se encuentra: la función BPR propuesta por U.S. Bureau of Public Roads (1964), la función de tiempo de viaje con demora de Webster y la función de tiempo de viaje con demora por congestión de Akcelik, además de otras funciones propuestas en el Laboratorio de Transporte y Sistemas Territoriales (LTST) de la UNAM (Londoño, 2003).

El resultado que se obtiene posee gran importancia, ya que el flujo estimado puede ser utilizado tanto para describir el tráfico como para predecir o recomendar un patrón de flujo vehicular en una red vial donde existe cierta demanda de viajes y los efectos de la congestión hacen que los tiempos de viaje en los arcos dependan del flujo. A diferencia de los paquetes de software existentes, se obtienen estimaciones de flujo más cercanas a la realidad, ya que en el algoritmo Frank Wolfe es posible incluir para cada arco la función de tiempo de viaje que mejor trabaja con sus características.

Para presentar los datos que conforman el resultado, se hace uso del Sistema de Información Geográfica TRANSCAD, con ello, es posible realizar la representación geográfica de los mismos. Desde la interfaz de la implementación del algoritmo Frank Wolfe, los datos son enviados a dicha plataforma utilizando .NET, para posteriormente ser desplegados en un mapa georeferenciado. Esto facilita enormemente su interpretación, ya que al presentarse gráficamente dentro del SIG TRANSCAD, su significado se vuelve más claro para los diversos usuarios que utilizan la aplicación.

Para que el desarrollo del software descrito cuente con la calidad requerida es necesario hacer uso de las técnicas y métodos que provee la Ingeniería de Software, es por ello que, la implementación

ha aplicado con éxito el Proceso Unificado, esto con el objeto de controlar y guiar la construcción. Gracias a ello se obtuvo una ventaja fundamental en el proyecto: Progreso visible creado sobre una arquitectura solida que ha sido capaz de adaptarse a los requisitos que se han presentado.

## Objetivo general

El objetivo general de esta tesis es llevar a cabo la implementación de un algoritmo para resolver el problema de equilibrio del usuario (PEU) en una red vial congestionada.

## Objetivos específicos

Los objetivos específicos se listan a continuación:

Selección del software adecuado para llevar a cabo la implementación del algoritmo.

Aplicación de una metodología de desarrollo de software que proporcione estabilidad, control y organización durante el proceso de implementación del algoritmo.

Empleo del paradigma de programación orientada a objetos con el propósito de facilitar la reutilización y extensión del código en futuros proyectos que se desarrollen en el Laboratorio de Transporte y Sistemas Territoriales (LTST) del Instituto de Ingeniería de la UNAM.

Obtención de una versión estable de la implementación del algoritmo con el objeto de efectuar diversas pruebas en una red vial conformada por diferentes tipos de arcos.

Conexión de la implementación del algoritmo con el SIG (Sistema de Información Geográfica) TRANSCAD para visualizar geográficamente el resultado obtenido.

## Consideraciones

Para llevar a cabo la implementación del algoritmo se consideró lo siguiente:

El PEU puede ser resuelto a través del algoritmo Frank Wolfe, el cual emplea el algoritmo Dijkstra, el algoritmo de Bisección y el modelo de asignación de flujo vehicular Todo o Nada. Además incluye funciones de costo (en el caso de esta tesis se incluyen tanto aquellas tradicionales como algunas propuestas por el LTST-IINGEN).

La red vial de prueba es una pequeña subred de la red vial de la Zona Metropolitana del Valle de México que se compone por dos tipos de arcos: semaforizados y de acceso controlado.

La metodología de desarrollo de software que se ha elegido para realizar la implementación del algoritmo es el Proceso Unificado, propuesto por Rumbaugh, Booch y Jacobson (2000). Los artefactos generados por dicho proceso se representan mediante UML.

El algoritmo se ha construido empleando el lenguaje de programación C#. Es importante mencionar que hace uso de los servicios de acceso a datos que forman parte de la biblioteca de clases base que se encuentran incluidas en el Microsoft .NET Framework.

## Estructura de la tesis

El capítulo 1 "*Ingeniería de tránsito*" establece la base teórica que permite analizar el volumen vehicular y el fenómeno de la congestión. Además, presenta el concepto de nivel de servicio como medida cualitativa de las condiciones de operación del volumen vehicular. Es importante mencionar que únicamente dentro de este capítulo se utiliza el término "*tránsito*" en lugar de "*tráfico*", la razón por la cual se realiza dicho cambio reside en el uso común de sus traducciones.

El capítulo 2 "*Modelo de equilibrio del usuario*" presenta el modelo de equilibrio del usuario con su respectiva formulación matemática y el algoritmo Frank Wolfe; cuya solución permite obtener el patrón de flujo que se ajusta al criterio de equilibrio del usuario. Además, se provee la teoría referente a las ecuaciones de tiempo de viaje y algoritmos auxiliares utilizados por Frank Wolfe. Al final, son descritas las características de la red de prueba y del caso de estudio implementado.

El capítulo 3 "*Sistemas de Información Geográfica*" expone un panorama general sobre los SIG; haciendo énfasis en su componente de software y en el *¿por qué?* del uso del SIG TransCAD. Este capítulo muestra la importancia de dicha tecnología y en este caso en particular, cómo se ha empleado para representar geográficamente el patrón de flujo vehicular generado a partir de la implementación del algoritmo Frank Wolfe para resolver el problema de equilibrio del usuario.

El capítulo 4 "*Proceso Unificado*" justifica el uso de dicho modelo de desarrollo de software para la correcta implementación del algoritmo guiado a través de un conjunto de normas establecidas. De igual forma, se explican sus principales características: "*dirigido por casos de uso, centrado en la arquitectura, siendo iterativo e incremental*"; junto a sus fases y los artefactos que genera, los flujos de trabajo utilizados para la implementación, y la importancia que tienen para cada fase.

El capítulo 5 "*Artefactos del sistema*" posee los artefactos de ingeniería generados por los flujos de trabajo del Proceso Unificado en la implementación del algoritmo que resuelve el problema de equilibrio del usuario; los cuales son descritos mediante texto o diagramas que son definidos por el Lenguaje Unificado de Modelado (UML). Por el tamaño de la imagen, algunos de ellos se encuentran almacenados en el disco compacto que se provee con el ejemplar físico de esta tesis.

El capítulo 6 "Análisis de Resultados" recopila los resultados más relevantes del presente trabajo. Asimismo analiza y verifica el comportamiento de las funciones de tiempo de viaje desarrolladas por el LTST mediante la interfaz de usuario. También se proporciona una serie de recomendaciones que deben ser tomadas en cuenta si se desea continuar con la implementación de un método de asignación dinámica de tráfico.

Finalmente se incluyen las conclusiones y referencias bibliográficas.

# Capítulo 1

# Ingeniería de tránsito

En los últimos años, con el aumento cada vez mayor del número de vehículos, la circulación en las calles y carreteras se ha tornado cada vez más compleja, motivo por el cual, adquiere gran importancia la realización de análisis operacionales más detallados de los sistemas viales. La Ingeniería de tránsito juega un papel importante dentro de este contexto, ya que se encarga de la planificación, diseño y operación del tránsito, y su relación con otros medios de transporte.

El presente capítulo tiene como objetivo primordial establecer la base teórica que permite analizar el volumen vehicular y el fenómeno de congestión. Asimismo presenta el concepto de nivel de servicio como medida cualitativa de las condiciones de operación del volumen vehicular. Es importante mencionar que únicamente en este capítulo se utiliza el término "*tránsito*" en lugar de "*tráfico*"; la razón fundamental de este cambio reside en el uso común de sus traducciones.

## 1.1 Definiciones

La lista de las siguientes definiciones sirve de base para entender el concepto tanto técnico como científico de la Ingeniería de Tránsito y de Transporte (Real Academia Española, 1970):

- **Tráfico o Tránsito**

  Circulación de vehículos por calles, caminos, etc.

- **Transporte**

  Traslado de personas o bienes de un lugar a otro.

La Ingeniería de Tránsito constituye una rama de la Ingeniería de Transporte, el ITE (Institute of Transportation Engineers) las define de la siguiente forma (ITE, 1999):

- **Ingeniería de Transporte**

  Es la aplicación de los principios tecnológicos y científicos a la planeación, al proyecto funcional, a la operación y a la administración de las diversas partes de cualquier modo de transporte, con el fin de proveer la movilización de personas y mercancías de una manera segura, rápida, confortable, conveniente, económica y compatible con el medio ambiente.

- **Ingeniería de Tráfico o Tránsito**

  Aquella fase de la Ingeniería de Transporte que tiene que ver con la planeación, el proyecto geométrico y la operación del tráfico por calles y carreteras, sus redes, terminales, tierras adyacentes y su relación con otros modos de transporte.

## 1.2 Sistema vial

Tal como lo menciona Cal Mayor y Cárdenas (2007, p.104): "*Uno de los patrimonios más valiosos con los que cuenta cualquier país es su infraestructura y en particular la del sistema vial, por lo que su magnitud y calidad constituyen un indicador del grado de desarrollo del mismo*". Considerando esta situación, es común encontrar un excelente sistema vial en un país de primer mundo y un sistema vial deficiente en un país subdesarrollado.

La red vial es toda aquella superficie terrestre, pública o privada, por donde circulan peatones y vehículos, se encuentra señalizada y bajo jurisdicción de las autoridades nacionales y/o locales. Son diversas las clasificaciones que existen de una red vial, usualmente cada país cuenta con una en particular. A continuación se proporcionan algunas de las clasificaciones más comunes.

### 1.2.1 Clasificación funcional

Considerando un criterio amplio de planeación, la red vial, tanto rural como urbana, debe ser clasificada de tal manera que sea posible fijar funciones especificas a las diferentes carreteras y calles que la conforman, para cumplir con las necesidades de movilidad de personas y de mercancías de forma rápida, económica, confortable y segura; de igual forma, con las necesidades de accesibilidad a las distintas propiedades o usos del área colindante (Cal Mayor et al., 2007).

Para facilitar la movilidad es necesario disponer de carreteras y calles rápidas; para tener acceso es indispensable contar con carreteras y calles lentas. De forma natural, entre estos dos extremos aparece todo el sistema de carreteras (rurales) y calles (urbanas). En términos generales, las carreteras y calles se pueden clasificar funcionalmente en tres grandes grupos (Cal Mayor et al., 2007):

1) Principales (arterias)
2) Secundarias (colectoras)
3) Locales

Las carreteras y calles principales son de accesos controlados destinados a proveer alta movilidad a grandes volúmenes de tránsito de paso y de poco o nulo acceso a la propiedad lateral; mientras que las carreteras y calles locales son de accesos no controlados que proveen fácil acceso a la propiedad lateral, de volúmenes de tránsito menores y poco utilizadas por el tránsito de paso. La figura 1.1 muestra gráficamente los grados de movilidad y accesibilidad.

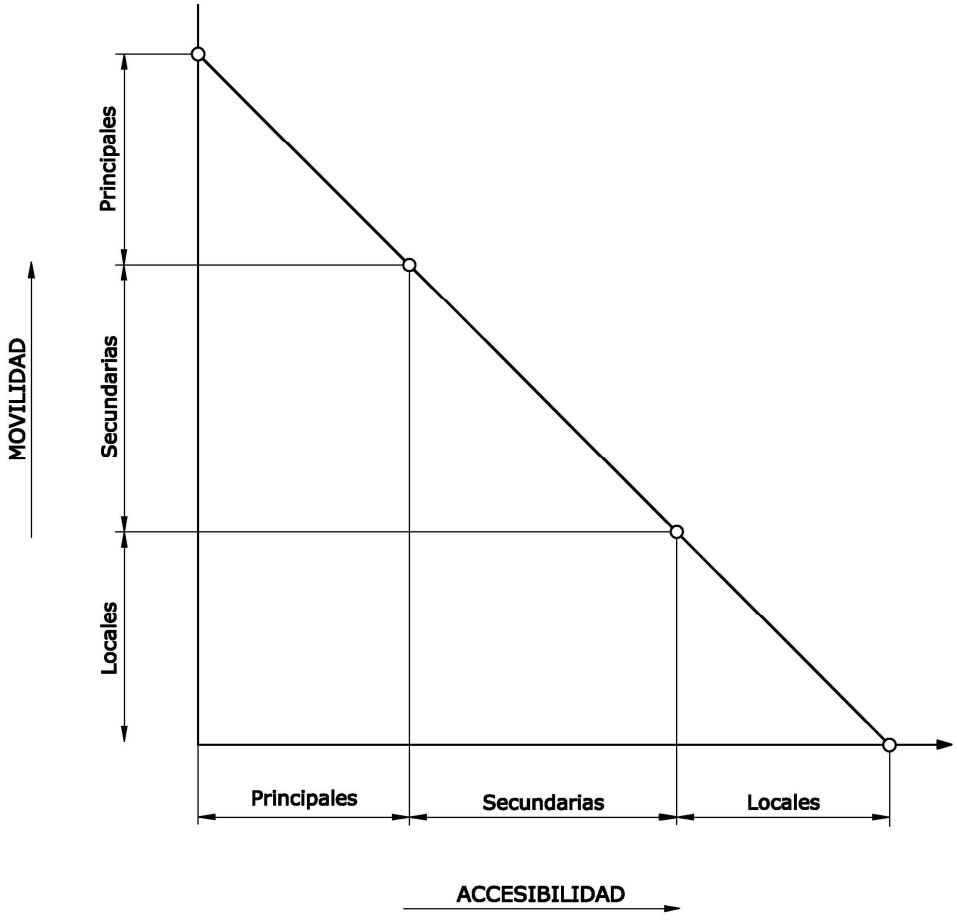



**Figura 1.1** *Clasificación funcional de un sistema vial*
(Cal Mayor et al., 2007, p.107)

### 1.2.2 Sistema vial urbano

La clasificación de un sistema vial urbano en términos de movilidad y accesibilidad se muestra en la figura 1.2, dicha clasificación es congruente con el esquema de jerarquía mostrado en la figura 1.3; con el propósito de unificar y simplificar la nomenclatura, se sugiere la siguiente clasificación (Cal Mayor et al., 2007):

1. **Autopistas y vías rápidas**

   Las autopistas facilitan el movimiento con rapidez de grandes volúmenes de tránsito entre áreas, a través o alrededor de la ciudad o área urbana. Son divididas, con control total de sus accesos y carecen de comunicación directa con las propiedades colindantes. Una autopista tiene separación total de los flujos conflictivos; mientras que una vía rápida puede o no tener algunas intersecciones a desnivel, pero puede ser la etapa anterior a una autopista. Estos dos tipos de arterias forman parte de la red vial primaria de un área urbana.

2. **Calles principales**

   Son las que permiten el movimiento del tráfico entre áreas o partes de la ciudad. Dan servicio directo a los generadores principales del tránsito y se conectan con el sistema de autopistas y vías rápidas. Con frecuencia son divididas y pueden tener control parcial de sus accesos. Las calles principales se combinan entre sí para formar un sistema que mueve el tránsito en toda la ciudad, en todas las direcciones.

3. **Calles colectoras**

   Son las que ligan las calles principales con las calles locales, proporcionando a su vez acceso a las propiedades colindantes.

4. **Calles locales**

   Son las que proporcionan acceso directo a las propiedades, ya sean residenciales, comerciales, industriales o de algún otro uso. Se conectan directamente con las calles colectoras y/o con las calles principales.

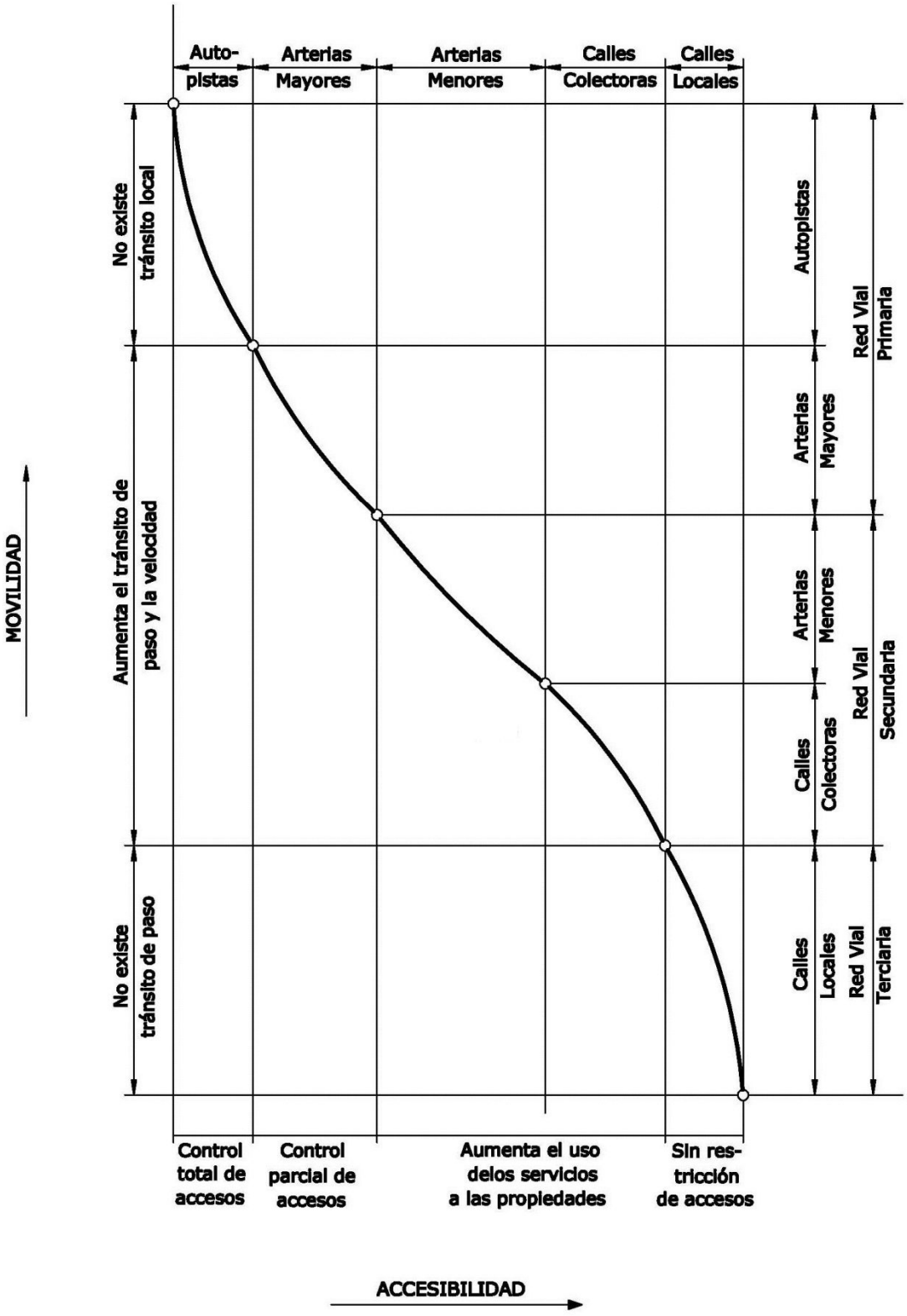



**Figura 1.2** *Movilidad y accesibilidad de un sistema vial urbano*
(Cal Mayor et al., 2007, p.109)

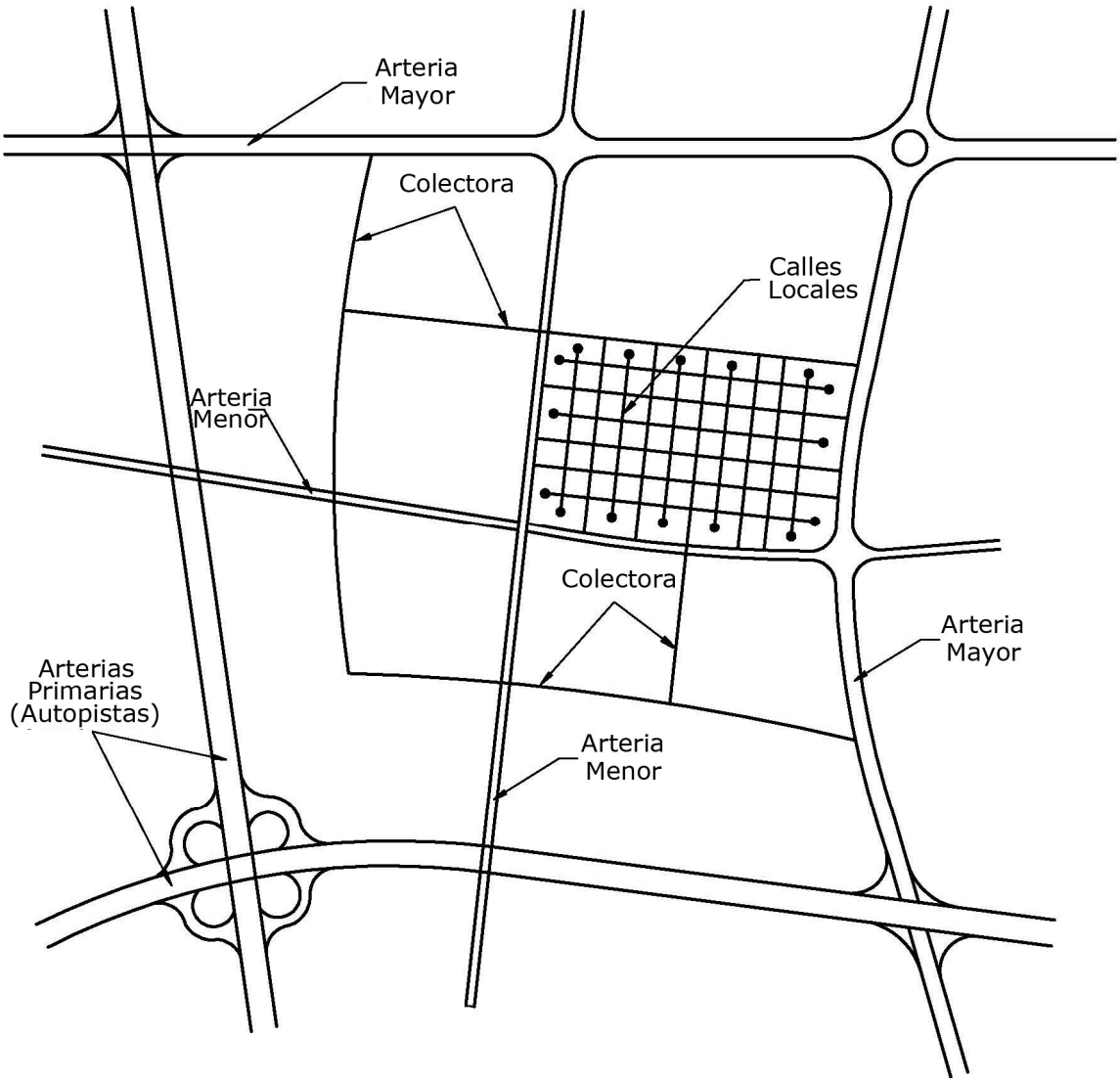



**Figura 1.3** *Jerarquía de un sistema vial urbano*
(Cal Mayor et al., 2007, p.109)

## 1.3 Análisis del flujo vehicular

Mediante el análisis de los elementos del flujo vehicular es posible entender las características y el comportamiento del tránsito, lo que constituye la base para el planeamiento, proyecto y operación de carreteras, calles y sus obras complementarias dentro del sistema de transporte. Con la aplicación de las leyes de la física y las matemáticas, el análisis del flujo vehicular permite describir la forma como circulan los vehículos en cualquier tipo de vialidad, lo que hace posible determinar el nivel de eficiencia de la operación (Cal Mayor et al., 2007).

Entre los resultados más útiles del análisis del flujo vehicular se encuentra el desarrollo de modelos microscópicos y macroscópicos que relacionan sus diferentes variables, tales como el volumen, la velocidad, la densidad, el intervalo y el espaciamiento. Dichos modelos han sido la base del desarrollo del concepto de capacidad y niveles de servicio aplicado a diferentes tipos de elementos viales (Cal Mayor et al., 2007).

### 1.3.1 Conceptos fundamentales

Ya que al final de este capítulo se abordan los modelos microscópicos y macroscópicos, en esta sección se presenta una breve descripción de algunas de las características fundamentales del flujo vehicular tomando como base la información que presenta Cal Mayor y Cárdenas (2007):

1. **Tasa de flujo o flujo** $(\boldsymbol{q})$

   Es la frecuencia a la cual pasan los vehículos por un punto o sección transversal de un carril o calzada, durante un intervalo de tiempo determinado, inferior a una hora.

   $q = \frac{N}{T}$ Usualmente $(veh/min)$ o $(veh/s)$ **(1.1)**

   Donde:

   $N$ = número de vehículos
   $T$ = intervalo de tiempo

2. **Velocidad** $(\boldsymbol{v})$

   Es la relación entre el espacio y el tiempo empleado en recorrerlo.

   $v = \frac{d}{t}$ Usualmente $(km/h)$ **(1.2)**

   Donde:

   $t$ = tiempo de recorrido
   $d$ = distancia recorrida

3. **Densidad** $(\boldsymbol{k})$

   Es el número de vehículos que ocupan una longitud específica de una vialidad en un momento determinado (ver figura 1.4).

   $k = \frac{N}{d}$ Usualmente $(veh/km)$ **(1.3)**

Donde:

$N$ = número de vehículos
$d$ = longitud

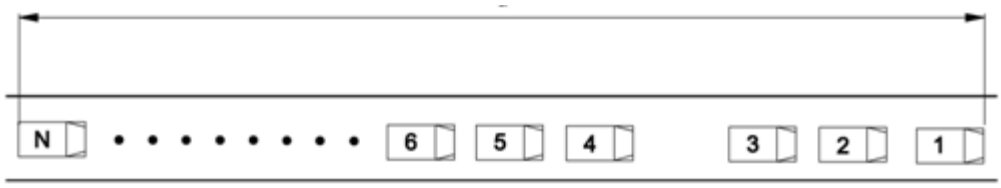



**Figura 1.4** *Densidad o concentración*
(Cal Mayor et al., 2007, p.283)

**4. Intervalo simple $(\mathbf{h_i})$**

Es la diferencia de tiempo entre el paso de dos vehículos consecutivos medido entre puntos homólogos y generalmente expresado en segundos.

**5. Intervalo promedio $\bar{h}$**

Es el promedio de todos los intervalos simples $\mathrm{h_i}$, existentes entre los diversos vehículos que circulan por una vialidad. Se expresa en segundos por vehículo $(\mathrm{s/veh})$ y se calcula mediante la siguiente expresión (ver figura 1.5):

$$\bar{h} = \sum_{i=1}^{N-1} h_i \, \frac{1}{N-1} \qquad \textbf{(1.4)}$$

Donde:

$\bar{h}$ = intervalo promedio (s/veh)
$N$ = número de vehículos
$N-1$ = número de intervalos (veh)
$h_i$ = intervalo simple entre el vehículo *i* y el vehículo *i + 1*

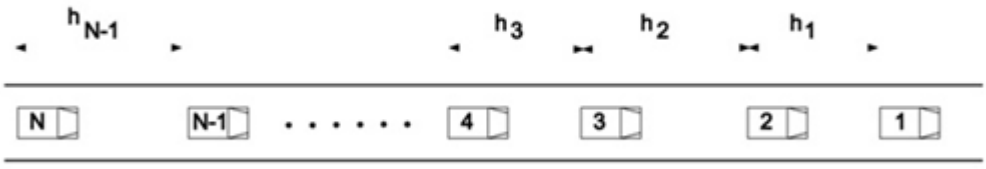



**Figura 1.5** *Intervalo entre vehículos*
(Cal Mayor et al., 2007, p.278)

Ya que las unidades del *intervalo promedio* $\bar{h}$ (s/veh) son las unidades inversas de la *tasa de flujo q* (veh/s), también puede plantearse la siguiente relación:

$$\bar{h} = \frac{1}{q} \qquad \textbf{(1.5)}$$

### 6. Espaciamiento $(s_i)$

Es la distancia entre el paso de dos vehículos consecutivos medido entre puntos homólogos y generalmente expresado en metros.

### 7. Espaciamiento promedio $(\bar{s})$

Es el promedio de todos los espaciamientos simples $s_i$, existentes entre los diversos vehículos que circulan en una vialidad. Se expresa en metros por vehículo $(\mathrm{m/veh})$ y se calcula mediante la siguiente expresión (ver figura 1.6):

$$\bar{s} = \sum_{i=1}^{N-1} s_i \, \frac{1}{N-1} \qquad \textbf{(1.6)}$$

Donde:

$\bar{s}$ = intervalo promedio (m/veh)
$N$ = número de vehículos
$N-1$ = número de espaciamientos (veh)
$h_i$ = espaciamiento simple entre el vehículo *i* y el vehículo *i + 1*

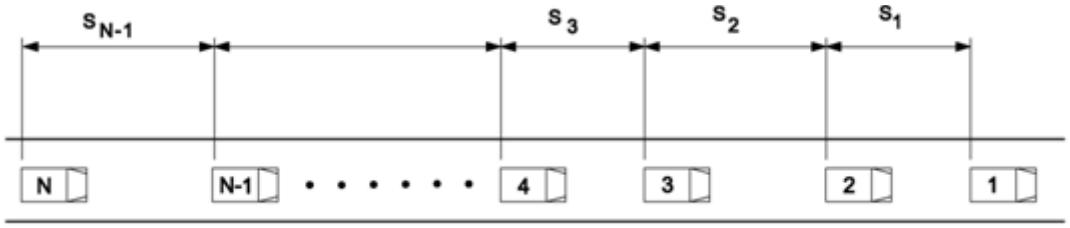



**Figura 1.6** *Espaciamiento entre vehículos*
(Cal Mayor et al., 2007, p.284)

Ya que las unidades del *espaciamiento promedio* $\bar{s}$ (m/veh) son las unidades inversas de la *densidad k* (veh/m), también es posible llegar a establecer la siguiente relación:

$$\bar{s} = \frac{1}{k} \qquad \textbf{(1.7)}$$

### 1.3.2 Ecuación fundamental del flujo vehicular

El esquema de la figura 1.7 muestra un par de vehículos consecutivos a los que se les han asociado atributos tanto en el tiempo como en el espacio.

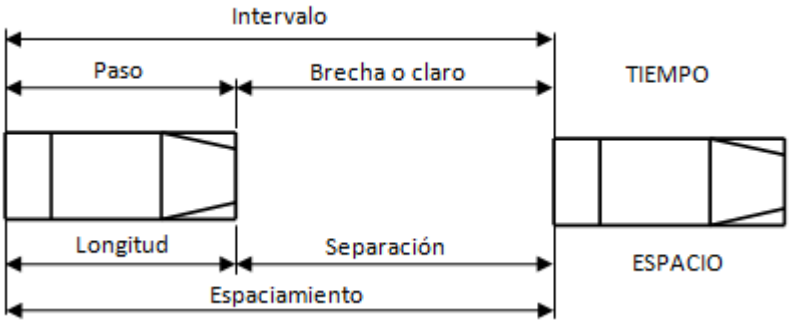



**Figura 1.7** *Relaciones de tiempo y espacio entre vehículos*
(Cal Mayor et al., 2007, p.290)

Con base en la figura 1.7, si se considera un grupo vehicular que se mueve a una velocidad $(\bar{v}_e)$ aproximadamente constante, su intervalo promedio $(\bar{h})$ y su espaciamiento promedio $(\bar{s})$ se pueden relacionar de la siguiente forma (Cal Mayor et al., 2007):

$$Espacio = (velocidad)(tiempo) \quad \textbf{(1.8)}$$

$$\bar{s} = \bar{v}_e \bar{\mathrm{h}} \quad \textbf{(1.9)}$$

Por las ecuaciones 1.5 y 1.7, se sabe que:

$$\bar{h} = \frac{1}{q}$$

$$\bar{s} = \frac{1}{k}$$

Al remplazar los dos valores anteriores en la ecuación 1.8 se obtiene:

$$\frac{1}{k} = \bar{v}_e \left(\frac{1}{q}\right) \quad \textbf{(1.10)}$$

De donde:

$$q = \bar{v}_e \, k \quad \textbf{(1.11)}$$

A la anterior relación se le conoce como ecuación fundamental del flujo vehicular, que en forma general se expresa como:

$$q = vk \quad \textbf{(1.12)}$$

## 1.4 Congestión vehicular

La congestión vehicular es la condición que se crea cuando el volumen de demanda de tránsito en uno o más puntos de una vía excede el volumen máximo que puede pasar por ellos, produciendo incrementos en los tiempos de viaje y embotellamientos (Bull y Thomson, 2001). Este fenómeno se presenta comúnmente en las horas pico, resultando frustrante para todos los automovilistas, ya que además de la pérdida de tiempo, desencadena diversos efectos negativos.

### 1.4.1 Significado analítico de la congestión

En general, la capacidad de un sistema es el número máximo de entidades que pueden ser procesadas por unidad de tiempo. Dada esta condición, la congestión se produce cuando en el sistema se procesa una demanda que es superior a su capacidad.

Con base en el análisis llevado a cabo por Cal Mayor y Cárdenas (2007, p.329), se considera un sistema que posee una capacidad de $\mu$ entidades por unidad de tiempo. Ya que la capacidad constituye la tasa máxima y su inverso es el intervalo máximo, cada entidad consume un tiempo promedio $t_p$ en ser procesado de:

$$t_p = \frac{1}{\mu} \quad \textbf{(1.13)}$$

Si las entidades llegan a una tasa $\lambda$ por unidad de tiempo, entonces el tiempo total de procesamiento $t_T$ por entidad será:

$$t_T = \begin{cases} t_p, & para\ \lambda \leq \mu \\ \infty, & para\ \lambda > \mu \end{cases} \quad \textbf{(1.14)}$$

Esta situación se ilustra en la figura 1.8.

Si $\lambda > \mu$ puede ocurrir que:

a) El sistema colapse, debido a una completa congestión tal que no se procesen unidades $(t_P = \infty)$.

b) Se forme una cola de espera que crece cada vez más $(t_p \to \infty)$.

c) Bajo condiciones de estado no estacionarias, solamente cuando $\lambda > \mu$ por un intervalo limitado de tiempo, la cola que se forma eventualmente se disipa.

Por otra parte, si $\lambda$ y/o $\mu$ son variables aleatorias, incluso cuando $\lambda < \mu$, las colas se pueden formar. Por lo anterior, en cualquier condición de estado (estacionario o no), el tiempo total de procesamiento $t_T$, por unidad, es igual al tiempo promedio de procesamiento $t_P$ más el tiempo de demora $t_D$, esto es:

$$t_T = t_p + t_D \qquad \textbf{(1.15)}$$

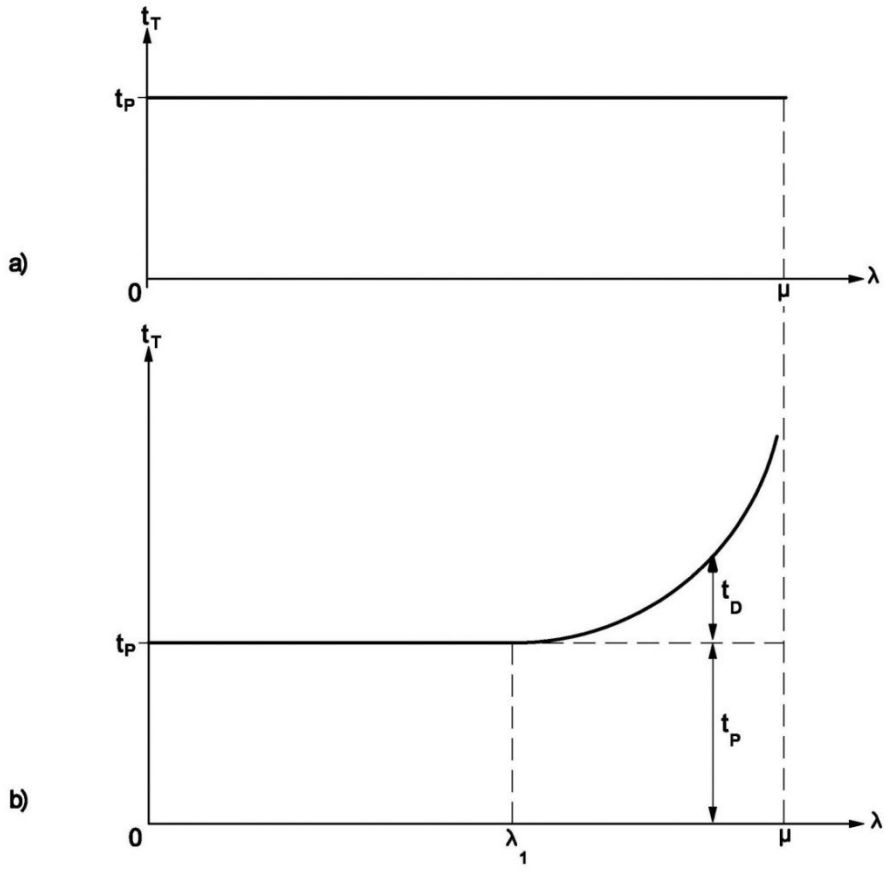



**Figura 1.8** *Significado de la congestión*
(Cal Mayor et al., 2007, p.290)

El significado práctico de la congestión se ilustra en la parte b) de la figura 1.8, su explicación analítica es la siguiente:

1. Para el rango de llegadas $0 < \lambda < \lambda_1$ no hay congestión $t_T = t_p$ ya que $t_D = 0$.

2. Para $\lambda > \lambda_1$ existe congestión puesto que $t_D > 0$, o lo que es lo mismo $t_T > t_P$. Si $\lambda$ se incrementa hasta que se aproxime a $\mu$, las demoras $t_D$ se incrementaran aún más.

3. Para cualquier nivel de demanda $\lambda > \mu$, la cola crecerá infinitamente si el nivel de demanda permanece constante. Si $\lambda$ varía, entonces la cola empezará a disiparse, siempre y cuando $\lambda$ caiga por debajo de $\mu$.

### 1.4.2 Causas de la congestión

Existen factores que contribuyen a la congestión vehicular (Bull et al., 2001):

**1. Uso intensivo de automóviles**

El creciente número de automóviles en circulación es la principal causa del aumento de la congestión vehicular; esto es consecuencia de la combinación de diversos factores, entre los que se destacan: la utilización del vehículo privado como principal medio de transporte, la disminución del precio de los automóviles, la mala calidad del transporte público, etcétera.

**2. Mal diseño y mantenimiento de la red vial**

La falta de demarcación de los carriles de circulación, los inesperados cambios en el número de carriles, los paraderos de transporte público ubicados justamente donde se reduce el ancho de la vialidad y otras deficiencias, entorpecen la fluidez del flujo vehicular. Además, el mal estado del pavimento y la presencia de baches, ocasionan restricciones de capacidad.

**3. Imprevistos en el tráfico**

El flujo vehicular se encuentra expuesto a los accidentes, reparaciones y obras viales, que pueden bloquear totalmente las vialidades o cortar uno o más carriles, creando cuellos de botella. Además, los fenómenos meteorológicos como la lluvia y las heladas provocan que la gente disminuya la velocidad por motivos de seguridad, generando retenciones vehiculares.

**4. Falta de cultura vial**

La falta de cultura vial es un problema que aumenta la congestión vehicular. Microbuses que se detienen justo antes de una intersección vial, conductores que tratan de imponerse en cruces para ahorrarse algunos segundos de viaje, autos estacionados en doble fila, son claros ejemplos de conductas de los automovilistas que afectan de forma negativa el flujo.

### 1.4.3 Consecuencias de la congestión

Los efectos perjudiciales de la congestión vehicular recaen sobre todos los habitantes de las ciudades, de una forma u otra, nadie queda inmune a sus consecuencias (Bull et al., 2001):

**1. Alto impacto económico**

La congestión vehicular aumenta los costos de operación y ocasiona pérdidas de tiempo productivo. Un estudio de la Comisión Económica para América Latina y el Caribe (CEPAL), afirma que la congestión vehicular posee un fuerte impacto sobre el producto interno bruto (PIB). De acuerdo a dicho estudio, la operación de los vehículos que circulan en las ciudades de más de 100 000 habitantes consume alrededor de 3.5% del PIB regional. Los autores del estudio, Ian Thompson, jefe de la Unidad de Transportes de CEPAL, y Alberto Bull, consultor de dicha agencia, calcularon que el valor social del tiempo consumido en exceso a causa de la congestión equivale a otro 3% del PIB latinoamericano (Bull et al., 2001).

**2. Contaminación atmosférica**

El tránsito vehicular es la principal fuente de emisiones contaminantes en las áreas urbanas. De acuerdo al inventario de Emisiones de la Zona Metropolitana del Valle de México (GDF, 2010) un importante porcentaje del total de las emisiones en la Zona Metropolitana es generado por fuentes móviles, es decir, vehículos automotores (Ver tabla 1.1).

**Tabla 1.1** *Porcentajes de emisiones por fuentes móviles*
(Inventario de Emisiones de la Zona Metropolitana del Valle de México, 2010)

| Emisiones | $PM_{10}$ | $PM_{2.5}$ | $S0_2$ | CO | NOx | COT |
|---|---|---|---|---|---|---|
| Fuentes moviles | 16% | 45% | 49% | 99% | 82% | 11% |

**3. Contaminación acústica**

Las fuentes de contaminación acústica en los núcleos urbanos son múltiples y diversas, pero actualmente el tránsito vehicular se ha constituido como uno de los principales generadores del exceso de ruido. Dicha situación puede provocar efectos negativos que van desde la disminución de la capacidad auditiva hasta diversos trastornos psicológicos y fisiológicos.

**4. Violencia vial**

Es una serie de acciones cometidas por automovilistas y que son motivadas por disputas con otros conductores o peatones, derivadas de problemas de tránsito principalmente por la congestión vehicular. Se manifiesta en diversas formas, tales como: aceleraciones bruscas, persecuciones, amenazas verbales, gestos obscenos con la mano, peleas físicas, entre otras.

## 1.5 Nivel de servicio

El nivel de servicio es una medida cualitativa empleada por los ingenieros de tránsito para describir las condiciones de operación de un flujo vehicular. El manual de capacidad vial HCM 2000 del TRB establece seis niveles de servicio denominados: A, B, C, D, E y F; los que se definen de acuerdo a las condiciones de operación de la red vial (circulación continua o discontinua) (Cal Mayor et al., 2007).

### 1.5.1 Segmentos básicos de autopistas

Los segmentos básicos de autopista son secciones de dos o más carriles por sentido, cuya operación no se ve afectada por maniobras de entrecruzamientos, ni por movimientos de convergencia o divergencia en rampas de enlace cercanas (Cal Mayor et al., 2007).

En la tabla 1.2, se describen sus correspondientes niveles de servicio, definidos por diversos factores, tales como: velocidad, densidad de flujo, retrasos, libertad para maniobrar, entre otros.

**Tabla 1.2** *Niveles de servicio para segmentos de autopistas*
(Highway Capacity Manual, 2000)

# NIVELES DE SERVICIO

## para segmentos básicos de autopistas

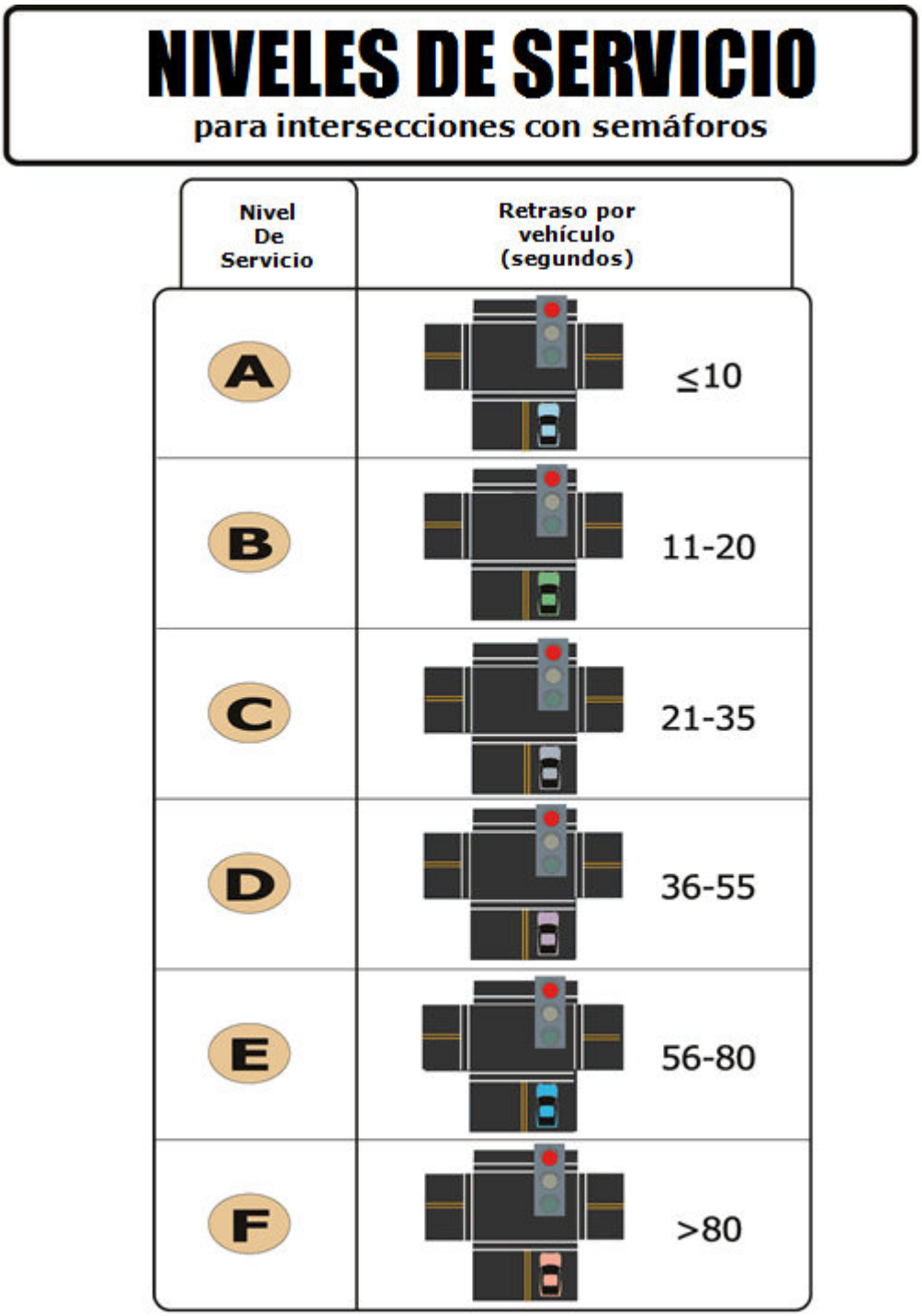
| Nivel de Servicio | Condiciones de operación | Velocidad (km/h) | Descripción técnica |
|---|---|---|---|
| A |  | 90+ | Circulación a flujo libre con mínimas restricciones para seleccionar velocidades o maniobrar. **No hay retrasos** |
| B |  | 80 | Flujo estable. Disminuye ligeramente la velocidad y la libertad de maniobra. **No hay retrasos** |
| C |  | 70 | Flujo estable, pero existe menor libertad para elegir velocidad, cambiar de carril o rebasar. **Retrasos mínimos** |
| D |  | 60 | El flujo se torna inestable. La velocidad está sujeta a cambios repentinos. Rebasar es complicado. **Retrasos mínimos** |
| E |  | 40 | El flujo es inestable. La velocidad cambia rápidamente. La libertad para maniobrar es baja. **Retrasos significativos** |
| F |  | | Alta congestión vehicular. La circulación se da bajo un comportamiento de arranque y parada. **Retrasos significativos** |

### 1.5.2 Intersecciones con semáforos

En la red vial de las zonas urbanas, las intersecciones semaforizadas poseen un papel muy importante en la regulación del tránsito, ya que permiten alternar los flujos vehiculares que se cruzan en un mismo nivel, de una forma ordenada y segura (Cal Mayor et al., 2007).

En la tabla 1.3, se describen sus correspondientes niveles de servicio, definidos a través de la demora que experimenta cada vehículo debido a las detenciones generadas por los semáforos.

**Tabla 1.3** *Niveles de servicio para intersecciones con semáforos*
(Highway Capacity Manual, 2000)



## 1.6 Modelos del flujo vehicular

En general los modelos del flujo vehicular se clasifican en dos grandes grupos: microscópicos y macroscópicos. Los modelos microscópicos consideran los espaciamientos y las velocidades individuales de los vehículos, con base en la teoría del seguimiento vehicular. A su vez, los modelos macroscópicos describen la operación en términos de sus variables de flujo, tomadas generalmente como promedios (Cal Mayor et al., 2007).

### 1.6.1 Modelos microscópicos

Los modelos microscópicos se enfocan en la descripción del comportamiento del tránsito a través del estudio de entidades discretas y su interacción existente (en este caso cada vehículo o conductor individual). Dichos modelos generan visualizaciones muy realistas de la operación, ya que permiten observar diversos detalles, tales como: utilización de carriles, generación de colas, interacción de vehículos con flujos peatonales, etc. (Larraga, 2008).

Si describe el comportamiento de cada vehículo existente, es posible representar de forma casi perfecta el tráfico vehicular, ya que todo lo que ocurra dentro de la red vial sería predecible y modelable. No obstante, es importante mencionar que el hecho de que este tipo de modelos intenten representar comportamientos humanos, aumenta en gran medida su complejidad y costo; además, requieren información que usualmente no es adquirible, o medida fácilmente.

Principales modelos microscópicos:

- Modelos del vehículo siguiente
- Modelos con autómatas celulares

Plataformas de análisis microscópico:

- Aimsum
- Corsim
- Flexsyt-II
- Getram
- Mitsim
- Netsim
- S-Paramics
- Transmodeler

### 1.6.2 Modelos macroscópicos

Los modelos macroscópicos se enfocan a captar las relaciones globales del tráfico, tales como velocidad promedio de los vehículos, flujo vehicular promedio y densidad global del tráfico. Por su naturaleza, son modelos continuos que hacen uso extensivo de ecuaciones diferenciales, y asumen que el comportamiento de los conductores depende de las condiciones del tráfico (Lozano et al., 2003).

Permiten realizar análisis sobre volúmenes de tráfico, dar respuestas aproximadas sobre los congestionamientos, las demoras y los tiempos de viaje.

Principales modelos macroscópicos:

- Modelos de gases cinéticos
- Modelos hidrodinámicos
- Modelos de asignación de equilibrio
- Modelo lineal B.D. Greenshields
- Modelo logarítmico H. Greenberg
- Modelo exponencial R.T. Underwood
- Familia de modelos L.A Pipes y P.K. Munjal

Plataformas de análisis macroscópico:

- Freflo
- Kronos
- Metacor
- Transcad
- Transyt-7f
- Visum

Los modelos de asignación de equilibrio permiten realizar una buena representación del tráfico. Dentro de ellos, el modelo de equilibrio del usuario se caracteriza por describir el comportamiento vehicular que más se apega a la realidad. Ya que el objetivo de esta tesis es: implementar un algoritmo para resolver el problema de equilibrio del usuario; el siguiente capítulo introduce formalmente "*El problema de equilibrio del usuario*".

# Capítulo 2

# Problema de equilibrio del usuario

Actualmente un enfoque que ha permitido una buena representación del tráfico es el de los modelos de asignación de equilibrio. Estos modelos pueden ser empleados para representar el comportamiento macroscópico del tráfico de grandes zonas urbanas o ciudades enteras. El flujo estimado puede utilizarse tanto para describir el tráfico como para predecir o recomendar un patrón de flujo vehicular en una red. Entre los modelos existentes, el modelo del equilibrio del usuario permite describir el comportamiento vehicular que más se apega a la realidad.

En este capítulo se presenta el problema de equilibrio del usuario con su respectiva formulación matemática; así como el algoritmo Frank Wolfe, cuya solución permite obtener un patrón de flujo que se ajusta al concepto de equilibrio del usuario. De igual forma, se provee la teoría referente a las ecuaciones de tiempo de viaje y algoritmos auxiliares utilizados por Frank Wolfe. Finalmente se describen las características de la red vial y del caso de estudio implementado.

## 2.1 Asignación de tráfico

La asignación es un proceso que predice el flujo modal por las posibles rutas que se dan entre cada par origen destino dentro de una red de interés (Londoño et al., 2003).

Durante un proceso clásico de asignación de tráfico se usan un conjunto de reglas o principios para cargar una red con una matriz de viajes establecida (de flujos reales o estimados), a fin de producir un conjunto de flujos en los arcos. Adicionalmente se pueden alcanzar otros objetivos:

- **Objetivos primarios**

1. Obtener buenas medidas agregadas en la red, por ejemplo flujos totales en los arcos, etc.
2. Estimar costos (tiempos) de viaje de zona a zona para un nivel de demanda dada de viajes.
3. Obtener flujos razonables en los arcos e identificar los que se encuentran congestionados.

- **Objetivos secundarios**

1. Estimar las rutas utilizadas para cada par origen destino.
2. Analizar cuáles pares O-D utilizan un arco o un camino particular.

- **Información necesaria**

1. Matriz O-D que expresa la demanda de viajes estimada para un intervalo de tiempo dado.
2. Una red, específicamente conformada por los arcos y sus propiedades

### 2.1.1 Selección de la ruta

Los procesos de asignación asumen que la elección de la ruta se realiza por un viajero racional, esto es, un viajero que escoge la ruta que le ofrece los costos menores percibidos para el par O-D. Algunos de los factores que influyen en su decisión son: el tiempo de viaje, la distancia, el costo en dinero, la congestión y colas, los tipos de maniobras requeridas, el tipo de camino, el paisaje, la presencia de señales, las obras en las vías, la constancia en el tiempo de viaje y los hábitos (Londoño et al., 2003).

La construcción de una expresión generalizada de costos que incorpore todos estos elementos constituye una tarea difícil. Además, no es práctico tratar de modelar todos los factores en un modelo de asignación de tráfico, por lo que, es inevitable llevar a cabo diversas aproximaciones.

La aproximación más común es considerar solo dos factores en la elección de rutas: el costo en tiempo y el costo monetario; además, el costo se considera proporcional a la distancia del viaje. La mayoría de los programas de asignación de tráfico permiten al usuario ponderar el tiempo y la distancia de viaje, para representar las percepciones de los conductores con estos factores.

Por otra parte, hay evidencias que sugieren que, por lo menos en el tráfico urbano de vehículos particulares, el tiempo es el factor dominante en la elección de rutas. Sin embargo, es un hecho que los conductores seleccionan rutas diferentes cuando viajan entre los mismos puntos O-D; situación que posiblemente sea ocasionada por las razones que a continuación son descritas:

- Diferencias en la percepción del viajero respecto a la ruta que proporciona el mejor tiempo.
- El efecto de la congestión impacta a la ruta más corta, aumenta su costo generalizado y hace menos atractiva la ruta. Se vuelve comparable con las rutas inicialmente menos atractivas.

Los diferentes tipos de modelos de asignación son adecuados para considerar estas influencias.

### 2.1.2 Modelos de asignación de tráfico

Los modelos de asignación se utilizan para estimar el flujo de tráfico en una red. Su propósito es describir, predecir o recomendar un patrón de flujo de tráfico en una red donde: existe cierta demanda de viajes y los efectos de la congestión hacen que los tiempos (costos) de viaje en los arcos sean dependientes del flujo (Lozano et al., 2003).

Existen diversos métodos de asignación de tráfico, entre los cuales se encuentran los siguientes:

1. Todo o Nada
2. Asignación de equilibrio
3. Asignación estocástica
4. Asignación por etapas con restricción de capacidad (funciones-intensidad-velocidad)
    a) Incremental
    b) Volumen medio
    c) De equilibrio
5. Asignación a rutas alternativas mediante curvas de distribución
6. Equilibrio estocástico del usuario
7. Asignación dinámica

A continuación se describen los métodos de asignación de tráfico utilizados en esta tesis: Todo o Nada y Asignación de equilibrio.

### 2.1.3 Asignación del Todo o Nada

Para un determinado par O-D todo el flujo es asignado indistintamente a la ruta más corta. Este modelo no se apega mucho a la realidad (si se trata de una red vial) ya que considera solo una ruta y excluye cualquier otra ruta a pesar de que sea muy parecida o tenga diferencias mínimas con respecto a la primera, en cuanto a distancia, tiempo o costo. Además, el tráfico es asignado sin tomar en consideración si existe o no capacidad suficiente; el tiempo de viaje es un dato fijo y no varía con la congestión en los arcos (Londoño et al., 2003).

El método de asignación Todo o Nada utiliza diversos algoritmos de ruta mínima para resolver el problema del camino más corto. En esta tesis, se utiliza el algoritmo “Dijkstra” para dar solución.

### 2.1.4 Asignación de Equilibrio

La técnica de asignación de equilibrio asume que el costo o tiempo de viaje en un arco, depende generalmente del volumen que por éste circula. Así, por ejemplo, en un arco congestionado de una arteria vial, el mayor tiempo o costo de viaje, es provocado por el volumen de vehículos que circula, y los viajeros tenderán a evitar usarlo, mientras existan otras alternativas para el viaje (Londoño et al., 2003).

En la asignación de equilibrio se dan dos tipos de enfoque:

1. **Equilibrio del usuario (EU)**

   En una red en condición de equilibrio del usuario, ningún viajero puede mejorar el tiempo de viaje por un cambio unilateral de su ruta.

   Utiliza un proceso iterativo para lograr una solución convergente en la cual ningún usuario puede mejorar su tiempo de viaje. En cada iteración, los flujos en los arcos son calculados; además, incorpora efectos de restricción de capacidad y el tiempo de viaje depende del flujo.

2. **Equilibrio de sistema optimo (SO)**

   Optimiza el sistema asignando rutas a los usuarios de tal forma que el tiempo o costo de viaje promedio sea mínimo en todo el sistema.

   Bajo este enfoque, ningún usuario podrá cambiar de ruta sin incrementar el tiempo de viaje total, aunque es posible que pueda reducir su propio tiempo de viaje. No constituye un modelo muy realista, pero puede llegar a ser útil para el análisis de diversos escenarios.

## 2.2 El modelo de equilibrio del usuario (EU)

El modelo de equilibrio del usuario asigna flujos a los arcos, de tal manera que cada usuario que se ha asignado a la red no pueda cambiar su ruta sin incrementar su tiempo (o costo) de viaje.

Este modelo se basa en el primer principio de Wardrop, el cual afirma: "Los tiempos de viaje en todas las rutas realmente usadas son menores o iguales que aquellos que requeriría un usuario en una ruta no utilizada". Esto significa que las rutas utilizadas por los usuarios son las más cortas en tiempo (o costo) bajo las condiciones prevalecientes de tráfico en la red (Londoño et al., 2003).

### 2.2.1 Formulación matemática del modelo EU

La formulación básica del modelo de equilibrio siguiendo el principio de equilibrio del usuario de Wardrop está dada por la función objetivo (2.1) y las restricciones (2.2), (2.3) y (2.4):

$$Min\ z(x) = \sum \int_0^{x_a} t_a(w)dw \quad \textbf{(2.1)}$$

**Sujeto a:**

$$\sum_k f_k^{rs} = q_{rs} \quad \forall r, s \quad \textbf{(2.2)}$$

$$f_k^{rs} \geq 0 \quad \forall r, s \quad \textbf{(2.3)}$$

$$y:\ x_a = \sum_{\boldsymbol{r}} \sum_{\boldsymbol{s}} \sum_{\boldsymbol{k}} f_k^{rs} \delta_{a,k}^{rs} \quad \forall a \quad \textbf{(2.4)}$$

**Donde:**

$a$: Nodos de la red.

$r, s$: Nodos origen destino.

$k$: Rutas definidas en la red entre los nodos $r - s$.

**Parámetros:**

$t_a(x_a)$: Tiempo de viaje en el arco $a$, como función del flujo sobre el arco.

$q_{rs}$: Demanda de viajes del origen $r$ al destino $s$.

$\delta_{a,k}^{rs}$: 1 si el arco $a$ es parte de la ruta que conecta $r$ con $s$, 0 de otro modo.

**Variable de decisión:**

$x_a$: Flujo en el arco $a$. Cantidad de viajes a través de $a$.

$f_k^{rs}$: Flujo en la ruta $k$ que conecta el origen $r$ con el destino $s$.
Cantidad de viajes que utilizan $k$ para ir de $r$ a $s$.

### 2.2.2 Algoritmo de Frank Wolfe

La aplicación del algoritmo Frank Wolfe, también conocido como el método de optimización de combinaciones convexas, resuelve el problema de minimización cuya formulación fue presentada en el punto anterior (2.2.1). El desarrollo de la solución matemática a dicho problema se encuentra fuera de los objetivos de esta tesis, pero puede consultarse dentro de la tesis para obtener el grado de maestría de Transporte en el Posgrado de Ingeniería de la UNAM, que lleva por nombre "Métodos de Asignación Dinámica de Trafico", que presenta Gloria Elena Londoño Mejía (Londoño et al., 2003).

El algoritmo Frank Wolfe, aplicado a la solución del problema de minimización de EU, puede ser resumido de la siguiente forma (Londoño, 2003, p.21):

**Algoritmo Frank Wolfe**

- **Restricciones**
  - Asignar la demanda al interior del algoritmo Frank Wolfe.
- **Paso Inicial**
  - Cargar la red mediante la asignación "Todo o nada" considerando la red vacía: $t_a^0 = t_a(0),\ \forall a$. Esto produce: $\{x_a^1\}$. Hacer el conteo $n = 1$.
- **Paso 1**
  - Actualizar $t_a^n = t_a(x_a^n), \forall a$
  - Encontrar:

$$z(x) = \sum_a \left[ t_a^0 x_a^n + \frac{t_a^0 \infty (x_a^n)^{(\beta+1)}}{(\beta+1) C_a^\beta} \right]$$

Ésta es la función objetivo que se va a minimizar en el sistema.

Su formulación depende de la función de tiempo de viaje (costo) seleccionada (BPR, Akcelik o Webster). La ecuación anterior considera la función BPR (U.S. Bureau of Public Roads).

- **Paso 2**
  - Determinar la dirección del vector descendente, asignando el flujo mediante la asignación "Todo o Nada", aplicando los $\{t_a^n\}$. Así se produce el conjunto de flujos auxiliares $\{y_a^n\}$.

- **Paso 3**
  - Encontrar el $\lambda_n$ que resuelve:

$$min_{0\leq 1} \sum_a \int_0^{x_a^n+\lambda_n(y_a^n+x_a^n)} t_a(\omega)d\omega$$

    Para realizar el proceso iterativo de búsqueda del mínimo se emplea el algoritmo de bisección aplicado de acuerdo a la función de tiempo de viaje seleccionada (BPR, Akcelik o Webster).

- **Paso 4**
  - Reasignar: $x_a^{n+1} = x_a^n + \lambda_n(y_a^n - x_a^n), \forall a$ .

- **Paso 5**
  - Efectuar la prueba de convergencia. Aplicar el criterio de convergencia y evaluarlo; si se cumple terminar, de lo contrario hacer $n = n + 1$ e ir al paso 1.

$$\frac{\sqrt{\sum_a \left(x_a^{n+1} - x_a^n\right)^2}}{\sum_a x_a^n} \leq k \qquad k = 0.001$$

El algoritmo Frank de Wolfe asigna el flujo a las rutas que ofrecen menos tiempo de viaje, y en cada iteración, reasigna el flujo a las rutas menos congestionadas. Este procedimiento permite distribuir el flujo en todas las rutas, tal que el tiempo de viaje sea igual en cualquiera de ellas, lo que garantiza el equilibrio en el sistema.

## 2.3 Funciones de tiempo de viaje

Las funciones de tiempo de viaje ($t_a(x_a^n), \forall a$) que son implementadas dentro del algoritmo Frank Wolfe, permiten trabajar con dos tipos de carreteras: con acceso controlado y controladas por semáforos. Dichas funciones son: BPR, Akcelik y Webster.

Cabe mencionar que en una red urbana de vías principales compuesta por corredores arteriales y vías de acceso controlado, las primeras poseen programas coordinados que generalmente poseen dos fases, en las que se eliminan los giros a la izquierda. El caso de los corredores arteriales se analiza macroscópicamente utilizando funciones que incluyan, al menos, la demora fija y por sobreflujo (Londoño et al., 2003).

### 2.3.1 Función de tiempo de viaje BPR

La función BPR fue sugerida por U.S. Bureau of Public Roads (1964), misma que ha sido muy utilizada en diversos trabajos de asignación de tráfico en redes urbanas. Es continua, creciente, diferenciable y tiene forma parabólica. Se compone de dos sumandos: el tiempo de viaje a flujo libre y otro término que es un factor del primero, que crece con el incremento de la relación flujo a capacidad $(x/Q)$. No incluye demoras por controles semaforizados. La función de tiempo viaje BPR es dada por (Londoño et al., 2003):

$$t(x_a^n) = t_a^0 \left(1 + \alpha \left(\frac{x_a^n}{Q}\right)^{\beta}\right) \qquad \textbf{(2.5)}$$

**Donde:**

$x_a^n$ : Es el flujo en el arco

$t_a^0 = \frac{L_a}{v_f}$ : Es el costo de viaje a flujo libre $(f_a(0))$

$L_a$ : Longitud del arco $a$

$v_f$ : Velocidad a flujo libre o velocidad máxima en el arco

Q : Capacidad del arco

$\propto, \beta$ : Parámetros del arco

La función BPR es simple y se comporta muy bien en procesos de carga de redes. Por ello, se utiliza ampliamente en modelos de planeación de transporte.

### 2.3.2 Función de tiempo de viaje con demora de Webster

En teoría de flujo de tráfico, los modelos analíticos incorporan tres tipos de demora (Londoño et al., 2003):

1. **Demora uniforme**

   La demora uniforme en una intersección semaforizada es la demora que asume llegada uniforme de vehículos, flujo estable y ninguna característica cíclica individual.

2. **Demora aleatoria**

   Es la demora adicional a la uniforme, en la que se asume que el flujo está distribuido aleatoriamente, y generalmente se presenta en intersecciones aisladas.

3. **Demora por sobreflujo**

   Es la demora adicional que ocurre cuando la capacidad de una fase individual o una serie de fases es menor que la demanda o la tasa de arribo de flujo.

La función de demora de Webster posee demoras por controles semaforizados. Se compone del tiempo de viaje a flujo libre adicionada con el componente de demora uniforme de Webster, en $s/veh$, la cual se vuelve constante cuando $x/Q$ es uno. Una vez que la demora uniforme se ha convertido en una constante, al tiempo de viaje se le agrega la demora por sobreflujo, esto es (Londoño et al., 2003):

$$t_a(x) = t_a^0 + d_a^1 + d_a^2 \qquad \textbf{(2.6)}$$

**Donde:**

$t_a^0 = \frac{L_a}{v_f}$ : Es el costo de viaje a flujo libre $(f_a(0))$

$d_a^1 = \left(\frac{C}{2}\right) * \left\{\frac{(1-(g/C))^2}{1-[\min(1,X)*(g/C)]}\right\}; [s/veh]$

$d_a^2 = \frac{T}{2}[1 - X]; [s/veh]$

**En las que:**

$d_a^1$: Demora uniforme del arco $a$

$C$: Es el ciclo del semáforo en $s$

$Q$: Capacidad en $veh/s$

$X = (x/Q)$: Grado de saturación

$x$: Tasa de flujo de llegada $veh/h$

$d_a^2$: Demora promedio por sobreflujo

$T =$ Tiempo considerado en la evaluación $s$

### 2.3.3 Función de tiempo de viaje con demora por congestión de Akcelik

La función de tiempo de viaje con demora por congestión de Akcelik se encuentra descrita por (Londoño et al., 2003):

$$t = t_a^0 + d_a^1 + \frac{1}{4}T_f\left\{(X-1) + \sqrt{(X-1)^2 + \frac{8A}{CT_f}X}\right\} \quad \textbf{(2.7)}$$

Como en las funciones de tiempo anteriores $t_a^0 = \frac{L_a}{v_f}$ es el costo de viaje a flujo libre $(f_a(0))$ y $d_a^1$ es la demora uniforme del arco $a$ de Webster.

La diferencia radica en que la demora por sobreflujo corresponde a la función dependiente del tiempo de la función de congestión de Akcelik.

$$d_a^2 = \frac{1}{4}T_f\left\{(X-1) + \sqrt{(X-1)^2 + \frac{8A}{CT_f}X}\right\} \quad (2.8)$$

**Donde:**

$X = (x/Q)$: Grado de saturación

$x$: Tasa de flujo de llegada, $veh/h$

$T_f$: Tiempo considerado en la evaluación [h]. Es definido por el usuario al cargar la red.

$Q$: Capacidad en $veh/s$

$A$: Parámetro del arco (atributo de la red)

La ecuación dependiente del tiempo está diseñada para incluir períodos de sobresaturación, definido mediante un grado de saturación $\mathrm{X} = (\mathrm{x}/\mathrm{Q})$, más grandes que uno.

## 2.4 Algoritmo de Dijkstra

El algoritmo de la ruta más corta de Dijkstra es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en sus arcos.

### 2.4.1 Conceptos generales

Los grafos constituyen una herramienta básica que permite modelar fenómenos discretos. Son fundamentales para la comprensión de las estructuras de datos y el análisis de algoritmos (Rosen, 1998).

1. Grafo

   Un grafo $G$, es un par $G = (V, E)$ donde $V$ es un conjunto de nodos y $E$ es un conjunto de pares no ordenados de nodos que reciben el nombre de arcos.

2. Dígrafo

   Un dígrafo $D$, es un par $D = (V, E)$ en el cual el conjunto de los arcos $E$ posee pares ordenados del conjunto de nodos $V$. Lo que asigna un orden en los extremos de cada arco.

3. Un grafo ponderado es

   Un grafo $G = (V, E)$, en el que a cada arco se le asigna un valor real no negativo o peso. El conjunto de arcos posee una función peso $W: E \to R$ que asigna valores reales a los arcos.

4. Un camino e

   Es la sucesión finita de nodos y arcos.

El peso de un camino $p = \langle v_0, v_1, \dots, v_k \rangle$ es la suma de los pesos de sus arcos.
Se define el peso del camino más corto de $u$ a $v$ como cualquier camino tal que:

$$\delta(u,v) = \begin{cases} min\{w(p): si\ existe\ un\ camino\ de\ u\ a\ v\} \\ \infty\ en\ caso\ caso\ de\ que\ no\ exista \end{cases}$$

El camino más corto del nodo $u$ al nodo $v$ es definido como cualquier camino $p$ con peso $w(p) = \delta(u,v)$.

El problema del camino más corto: dado un grafo $G = (V,E)$, se desea encontrar un camino más corto desde un nodo origen dado $s \in V$ a un nodo $v \in V$.

### 2.4.2 Relajación

Para cada vértice $v \in V$ se tiene un atributo $d[v]$ el cual es un límite superior en el peso del camino más corto desde el origen $s$ a $v$. El valor $d[v]$ constituye el peso del camino más corto.

El proceso de relajación de un arco $(u,v)$ consiste en examinar si se puede mejorar el peso del camino más corto a $v$ pasando por $u$ y, si es el caso, actualizar $d[v]$ y $\pi[v]$ (Hayet, 2008).

La relajación puede decrementar el valor del peso estimado del camino más corto $d[v]$ y actualizar el campo $\pi[v]$ del predecesor de $v$.

El siguiente código realiza una relajación del arco $(u,v)$:

**RELAX(u,v,w)**

1. **if** $d[v] > d[u] + w(u,v)$
2. **then** $d[v] \coloneqq d[u] + w(u,v)$
3. $\pi[v] \coloneqq u$

La figura 2.1 muestra dos ejemplos de relajación de un arco, uno en el cual el peso estimado del camino más corto decrece y el otro en el que no hay cambios en el peso estimado.

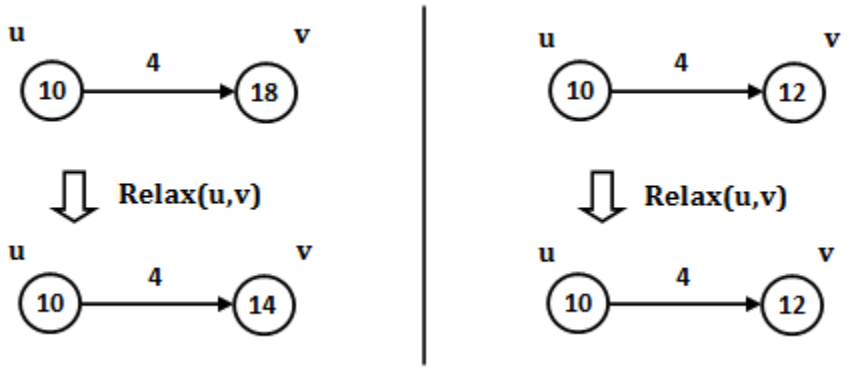



**Figura 2.1** *Casos de relajación*
(Elaboración propia)

### 2.4.3 Algoritmo Dijkstra

Dado un grafo dirigido $G = (V, E)$ y ponderado con pesos no negativos; dado un nodo origen s ∈ $V$, el algortimo Dijkstra permite obtener los caminos más cortos al resto de nodos $V$ (Rosen et al., 1998).

El algoritmo Dijkstra es el siguiente:

**Dijkstra's algorithm (G,w,s)**

```
1.  for each vertex v in V[G]      // Initializations
2.     d[v]:= infinity
3.     previous[v]:= undefined
4.  d[s]:= 0
5.  S:= empty set
6.  Q:= set of all vertices

7.  while Q is not an empty set     // The algorithm itself
8.  u:= Extract_Min(Q)
9.  S:= S union {u}

10.    for each edge(u,v) outgoing from u
11.        if d[v] > d[u] + w(u,v)   // Relax (u,v)
12.           d[v]:= d[u] + w(u,v)
13.           previous[v]:= u
```

El algoritmo Dijkstra corre sobre un grafo dirigido $G = (V, E)$ al que se asocia una función de peso no negativo $w$ y un nodo origen $s \in V$, y termina con $d[v] = \delta(s, v)$ para todo $v \in V$.

## 2.5 Método de Bisección

El método de bisección es un método numérico sencillo y versátil que permite encontrar la raíz real de una función continua. Se basa en el teorema de Bolzano que establece que si se tiene una función $y = f(x)$ de variable real y continua en el intervalo $[a, b]$, y el signo de la función en el extremo $a$ es distinto al signo de la función en el extremo $b$ del intervalo, existe por lo menos un valor $c$ dentro de dicho intervalo $[a, b]$, en el que se cumple $f(c) = 0$; $c$ es la raíz buscada (Chapra y Canale, 2006). Tal y como se observa en la figura 2.2.

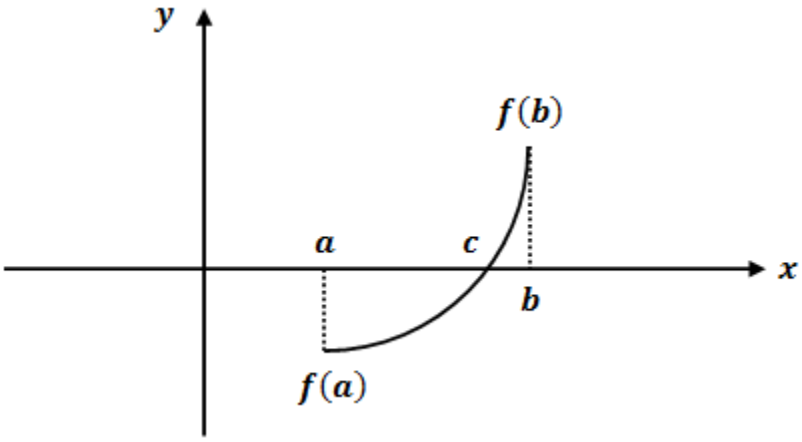



**Figura 2.2** *Método de bisección*
(Elaboración propia)

A continuación se proporciona el algoritmo de bisección:

**Método de Bisección**

1. Elegir valores iniciales para $a$ y $b$ de forma que: $f(a) * f(b) < 0$
2. Encontrar la primera aproximación a la raíz con la formula: $c_n = (a + b)/2$
3. Determinar en qué intervalo se encuentra la raíz:

   3.1 Si $f(a) * f(c_n) = 0$ , $c_n$ es la raíz.
   3.2 Si $f(a) * f(c_n) < 0$, $b = c_n$
   3.3 Si $f(a) * f(c_n) > 0$, $a = c_n$

4. Calcular una nueva aproximación a la raíz: $c_{n+1} = (a + b)/\ 2$
5. Evaluar la aproximación relativa

   3.1 Si $|c_{n+1} - c_n| < tolerancia$
   3.2 (Verdadero) Entonces $c_{n+1}$ es la raíz.
   3.3 (Falso) Repetir pasos 3, 4 y 5.

## 2.6 Funciones de tiempo de viaje incluidas en la implementación del algoritmo Frank Wolfe

Los procesos de carga se realizan en la red de prueba descrita en el punto anterior (2.6); para ello se emplea una matriz origen-destino conocida, asignada mediante el algoritmo Frank Wolfe o de las combinaciones convexas. En cada proceso de asignación se utilizan funciones de tiempo de viaje apropiadas para cada arco. Dichas funciones de tiempo de viaje $t_a(x_a)$ se caracterizan por ser positivas, crecientes y diferenciables. Adicionalmente la función de costo total $z(x_a)$ es creciente, diferenciable y convexa. Por último, es posible resolver el subproblema de optimización en el algoritmo de búsqueda del mínimo costo de la red. Lo anterior para garantizar condiciones de existencia y único resultado. Las funciones $t_a(x_a)$ empleadas se describen a continuación.

### 2.6.1 BPR

El proceso de carga con la función BPR, aplica de igual forma para los arcos de acceso controlado y aquellos semaforizados.

- **Función de tiempo**

  La función de tiempo BPR se encuentra descrita en la ecuación 2.5 del presente capitulo.

- **Aporte del arco en la sumatoria del costo del sistema $z(x_a)$**

$$z(x_a) = \sum \left[ t_a^0 x_a^n + \frac{t_a^0 \alpha (x_a^n)^{\beta+1}}{(\beta+1) Q_a^\beta} \right]$$

$Z(x)$: costo del sistema, en unidades de tiempo en el periodo que se da la demanda, es decir si la demanda está en veh/h, el costo del sistema se da en $[(h/veh) * (veh/h)]$

- **Aporte del arco en la sumatoria de minimización**

  Se quiere buscar un λ mediante el método de bisección, tal que:

$$\sum_a (y_a^n - x_a^n)\, t_a^0 \left\{1 + \frac{\alpha}{Q_a^\beta}[x_a^n + \lambda(\mathrm{y_a^n} - \mathrm{x_a^n})]^\beta\right\} = 0$$

  Siendo $0 \leq \lambda \leq 1$

  $x_a^n$: flujo de demanda que se carga con el método "todo o nada" cuando la red está vacía

  $y_a^n$: flujo de demanda que se carga con el método "todo o nada", en la que los costos han sido actualizados con $x_a^n$, es decir, con $f(x_a^n) = t_a^0(1 + \alpha\left(\frac{x_a^n}{Q}\right)^\beta)$

  $n$: Iteración del algoritmo Frank Wolfe

  El lamba se busca para encontrar un nuevo flujo para cargar la red, es decir:
  $x_a^{n+1} = x_a^n + \lambda_n(y_a^n - x_a^n), \forall a$ (Paso 4 del algoritmo Frank Wolfe)

## 2.6.2 BPR-Webster

Estas funciones varían según el tipo de arco, en la red se cuenta con dos tipos de arcos: "Acceso controlado"(como los que integran Periférico) y "Semaforizado" (como los que integran Revolución). Estos datos son definidos por la tabla de atributos de la red vial de prueba dentro de la columna "Acceso".

Por lo que, ahora se tiene que programar para cada tipo de arco:

- Una función de tiempo $\boldsymbol{t_a(x_a)}$
- El aporte del arco en la sumatoria del costo del sistema $\boldsymbol{z(x_a)}$
- El aporte del arco en la sumatoria del problema de minimización

$$min_{0\leq 1} \sum_a \int_0^{x_a^n + \lambda_n(y_a^n + x_a^n)} t_a(\omega) d\omega$$

  El cual se soluciona con el algoritmo de bisección y se usan las $\frac{dz[x_a^n + \lambda(y_a^n - x_a^n)]}{d\lambda}$

A continuación se indican las expresiones matemáticas:

**1. Arco de acceso controlado**

- **Función de tiempo**

  La función de tiempo BPR se encuentra descrita en la ecuación 2.5 del presente capitulo.

- **Aporte del arco en la sumatoria del costo del sistema $z(x_a)$**

$$z(x_a) = \sum \left[ t_a^0 x_a^n + \frac{t_a^0 \alpha (x_a^n)^{\beta+1}}{(\beta+1) Q_a^\beta} \right] + z(x_a^n)$$

  $z(x_a^n)$: aporte de otro tipo de arcos

  $Z(x)$: costo del sistema, en unidades de tiempo en el periodo que se da la demanda, si la demanda está en $(veh/h)$, el costo del sistema se da en $[(h/veh) * (veh/h)]$

- **Aporte del arco en la sumatoria de minimización**

  Se quiere buscar un λ mediante el método de bisección, tal que:

$$\sum_a (y_a^n - x_a^n)\, t_a^0 \left\{ 1 + \frac{\alpha}{Q_a^\beta} [x_a^n + \lambda(\mathrm{y_a^n} - \mathrm{x_a^n})]^\beta \right\} + \frac{dz[x_a^n + \lambda(y_a^n - x_a^n)]}{d\lambda} = 0$$

  $\frac{dz[x_a^n + \lambda(y_a^n - x_a^n)]}{d\lambda}$: otro tipo de arco

**2. Arco de acceso semaforizado**

Las expresiones matemáticas para el tipo de arcos de acceso semaforizado no pueden ser descritas dentro de esta tesis, ya que forman parte del artículo próximo a publicarse: "Suitable cost functions for signalized arterials and freeways, in the user equilibrium assignment problem" escrito por Gloria Elena Londoño Mejía y Angélica Lozano, del Laboratorio de Transporte y Sistemas Territoriales (LTST) del Instituto de Ingeniería de la UNAM (Londoño y Lozano, 2012).

### 2.6.3 Akcelik-Webster

Al igual que con las ecuaciones BPR-Webster, estas funciones varían según el tipo de arco de la red vial de prueba, pero ahora se ha agregado una nueva función en la que la demora 2 (demora promedio por sobreflujo) se caracteriza por tomar en cuenta el sobreflujo de Akcelik.

Las expresiones matemáticas tanto para el acceso controlado como el semaforizado no pueden ser descritas dentro de esta tesis, por la misma razón que se indica en el punto anterior (2.6.2).

## 2.7 Red vial de prueba

La implementación del Algoritmo de Frank-Wolfe es probada en una red con más de un tipo de arco (vialidad). La red vial de prueba es una pequeña subred de red vial de la Zona Metropolitana del Valle de México, posee un origen y un destino, y es posible seleccionar una de las dos rutas disponibles. La primera ruta corresponde a un corredor arterial semaforizado con planes coordinados llamado *"Revolución"*; la otra ruta es una vía de acceso controlado llamada *"Segundo piso Periférico"*.

La red vial está formada por lo siguiente:

- Red con 32 arcos, 32 nodos, un origen, un destino y dos rutas.
- Atributos de arco: id, nombre, nodo origen, nodo destino, longitud, sentido de circulación, velocidad a flujo libre, capacidad de arco, número de carriles de circulación, ancho de carril.
- Atributos de intersecciones: faseamiento, tipo de control, duración del ciclo, tiempo verde efectivo, flujo de saturación, tasa de llegada, grado de saturación, demora fija.
- Matriz O-D conocida.

A continuación se proporciona:

- Imagen geográfica de la red vial de prueba (figura 2.3).
- Características y parámetros de los arcos que conforman la red vial de prueba (tabla 2.1).
- Nodos existentes en la red de prueba (tabla 2.2).

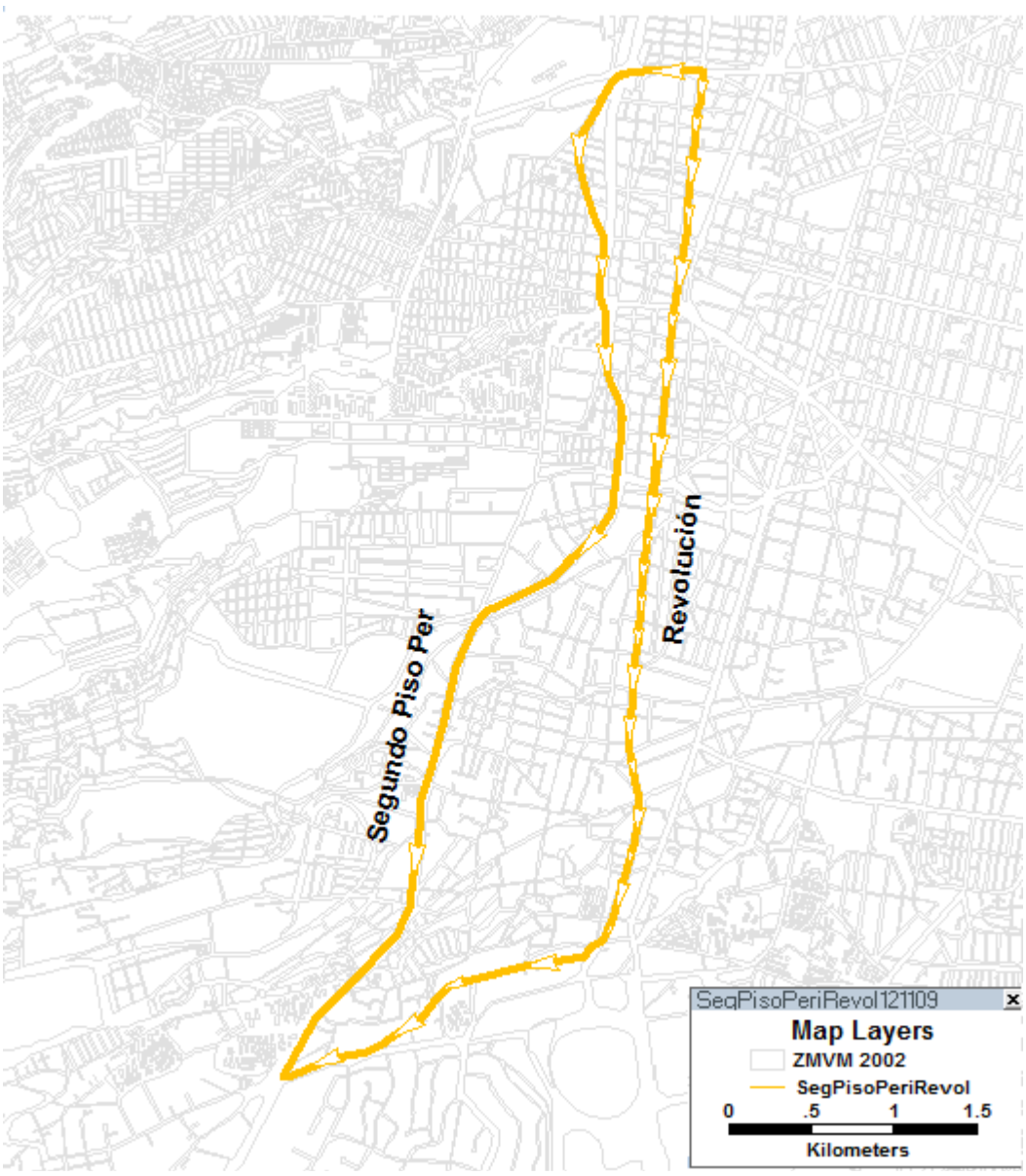



**Figura 2.3** *Red vial de prueba*
(Elaboración propia)

**Tabla 2.1** *Características y parámetros de los arcos de la red de prueba: Revolución y Segundo Piso Periférico* (Elaboración propia)

| Arco Id | Nombre | From nodo | To nodo | Longitud [km] | Dir | Vel_FF [km/h] | Acceso | Capacidad | Ciclo [h] | Verde [s] | Alfa | Beta | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Av. San Antonio | 30 | 75 | 0.4366 | 1 | 60 | Controlado | 4400 | | | 0.04 | 3 | 0.1 |
| 75 | SegundoPisoPer | 77 | 78 | 0.87891 | 1 | 60 | Controlado | 4400 | | | 0.04 | 3 | 0.1 |
| 76 | SegundoPisoPer | 78 | 13 | 1.46402 | 1 | 60 | Controlado | 4400 | | | 0.04 | 3 | 0.1 |
| 77 | SegundoPisoPer | 13 | 57 | 3.10463 | 1 | 60 | Controlado | 4400 | | | 0.04 | 3 | 0.1 |
| 78 | Revolucion | 79 | 47 | 0.08414 | 1 | 60 | Semaforizado | 3867.13 | 120 | 61.06 | 0.95 | 2 | 0.4 |
| 74 | SegundoPisoPer | 75 | 76 | 1.18813 | 1 | 60 | Controlado | 4400 | | | 0.04 | 3.4 | 0.1 |
| 29 | Revolucion | 30 | 60 | 0.22767 | 1 | 60 | Semaforizado | 7611.36 | 120 | 60.31 | 0.55 | 2 | 0.4 |
| 58 | Revolucion | 61 | 64 | 0.22236 | 1 | 60 | Semaforizado | 5523.46 | 120 | 69.77 | 0.6 | 2.4 | 0.4 |
| 31 | Revolucion | 32 | 61 | 0.30646 | 1 | 60 | Semaforizado | 8213.07 | 120 | 64.84 | 0.55 | 2.3 | 0.4 |
| 59 | Revolucion | 62 | 38 | 0.36157 | 1 | 60 | Semaforizado | 5236.4 | 120 | 55.12 | 0.4 | 2.5 | 0.4 |
| 60 | Revolucion | 64 | 62 | 0.05862 | 1 | 60 | Semaforizado | 5523.46 | 120 | 69.77 | 0.6 | 2.4 | 0.4 |
| 66 | Revolucion | 70 | 71 | 0.32382 | 1 | 60 | Semaforizado | 4825.37 | 120 | 76.19 | 0.6 | 2.3 | 0.4 |
| 65 | Revolucion | 69 | 70 | 0.19604 | 1 | 60 | Semaforizado | 4849.43 | 120 | 76.57 | 0.8 | 2.4 | 0.4 |
| 61 | Revolucion | 65 | 41 | 0.46325 | 1 | 60 | Semaforizado | 5036.27 | 120 | 79.52 | 0.42 | 2.6 | 0.4 |
| 62 | Revolucion | 38 | 65 | 0.19977 | 1 | 60 | Semaforizado | 4053.33 | 120 | 64 | 0.6 | 2.4 | 0.4 |
| 64 | Revolucion | 68 | 69 | 0.18014 | 1 | 60 | Semaforizado | 3930.47 | 120 | 62.06 | 0.75 | 2 | 0.4 |
| 63 | Revolucion | 66 | 67 | 0.13175 | 1 | 60 | Semaforizado | 5381.43 | 120 | 84.97 | 0.9 | 3 | 0.4 |
| 40 | Revolucion | 41 | 42 | 0.40226 | 1 | 60 | Semaforizado | 3461.8 | 120 | 54.66 | 0.42 | 2 | 0.4 |
| 41 | Revolucion | 42 | 66 | 0.28311 | 1 | 60 | Semaforizado | 4446.63 | 120 | 70.21 | 0.59 | 2.3 | 0.4 |
| 42 | Revolucion | 67 | 68 | 0.21205 | 1 | 60 | Semaforizado | 4905.8 | 120 | 77.46 | 0.8 | 2.3 | 0.4 |
| 67 | Revolucion | 71 | 46 | 0.26995 | 1 | 60 | Semaforizado | 6627.83 | 120 | 104.7 | 0.4 | 4.8 | 0.4 |
| 68 | Revolucion | 72 | 49 | 0.16887 | 1 | 60 | Semaforizado | 4731 | 120 | 74.7 | 0.85 | 2.4 | 0.4 |
| 45 | Revolucion | 46 | 79 | 0.19489 | 1 | 60 | Semaforizado | 4051.43 | 120 | 63.97 | 0.7 | 2.2 | 0.4 |
| 46 | Revolucion | 47 | 72 | 0.2814 | 1 | 60 | Semaforizado | 4578.37 | 120 | 72.29 | 0.6 | 2.3 | 0.4 |
| 69 | RioDeLaMagdal | 73 | 74 | 0.23544 | 1 | 60 | Semaforizado | 6887.5 | 120 | 87 | 0.7 | 3 | 0.4 |
| 48 | Revolucion | 49 | 50 | 0.2839 | 1 | 60 | Semaforizado | 3780.37 | 120 | 59.69 | 0.6 | 1.9 | 0.4 |
| 49 | Revolucion | 50 | 51 | 0.14243 | 1 | 60 | Semaforizado | 3306.63 | 120 | 52.21 | 0.7 | 1.8 | 0.4 |
| 71 | Eje 10 Sur | 53 | 57 | 0.53387 | 1 | 60 | Semaforizado | 3541.13 | 120 | 44.73 | 0.32 | 2 | 0.4 |
| 70 | Canoa | 74 | 53 | 0.54466 | 1 | 60 | Semaforizado | 6887.5 | 120 | 87 | 0.35 | 3 | 0.4 |
| 50 | Eje 10 Sur | 51 | 73 | 0.83211 | 1 | 60 | Semaforizado | 6887.5 | 120 | 87 | 0.25 | 3.3 | 0.4 |
| 57 | Revolucion | 60 | 32 | 0.22684 | 1 | 60 | Semaforizado | 8213.07 | 120 | 64.84 | 0.55 | 2 | 0.4 |
| 73 | SegundoPisoPer | 76 | 77 | 0.20127 | 1 | 60 | Controlado | 4400 | | | 0.04 | 3 | 0.1 |

**Tabla 2.2** *Nodos existentes en la red de prueba: Revolución y Segundo Piso Periférico*
(Elaboración propia)

| Nodo Id | Nombre |
|---|---|
| 75 | Salida Prolong. |
| 77 | Salida Miranda |
| 78 | |
| 13 | |
| 79 | |
| 30 | |
| 61 | Antonio Van Dick |
| 32 | Rembrandt |
| 62 | |
| 69 | R. Boker - R. Ca |
| 70 | Tlacopac |
| 64 | Andrea del Sarto |
| 65 | Castañeda |
| 38 | Molinos |
| 67 | Corregidora |
| 66 | Macedonio Alcalá |
| 41 | M. Gómez |
| 42 | Barranca del Mue |
| 68 | Sofia |
| 71 | María Luisa |
| 72 | Dr Gálvez |
| 46 | Desierto de Los |
| 47 | Av. La Paz |
| 73 | Iglesia |
| 49 | Rey Cuauhtémoc |
| 50 | Altamirano |
| 51 | Río Magdalena |
| 53 | |
| 74 | Fraternidad |
| 57 | |
| 60 | Andrea del Casta |
| 76 | Converg Periféri |

La implementación del algoritmo Frank Wolfe para resolver el problema de equilibrio del usuario, junto a las expresiones matemáticas expuestas, genera satisfactoriamente patrones de flujo. Una de las principales características de esta tesis reside en la conexión de la aplicación con el SIG TransCAD; mediante su uso es posible representar geográficamente el patrón de flujo. Dada la relevancia que posee dicha tecnología en el trabajo realizado, a continuación se describen los *"Sistemas de Información Geográfica"*.

# Capítulo 3

# Sistemas de información Geográfica

Los Sistemas de Información Geográfica (SIG) se han convertido en una tecnología esencial que es ampliamente utilizada en diversas áreas de estudio, tales como: transporte, urbanismo, comercio, logística, entre otras. Su capacidad para generar representaciones del mundo real a partir de bases de datos geográficas constituye un instrumento altamente eficaz que permite analizar aquellos fenómenos que poseen patrones, relaciones y tendencias en la información.

Ya que en esta tesis se emplea el SIG TransCAD para representar geográficamente el patrón de flujo vehicular generado a partir de la implementación del algoritmo que resuelve el problema del modelo de equilibrio del usuario, este capítulo presenta un panorama general sobre los SIG; haciendo énfasis en su componente de software y justificando la utilización del SIG TransCAD.

## 3.1 Definición de SIG

El término SIG procede del acrónimo de Sistema de Información Geográfica (en ingles GIS, Geographic Information System); ha sido definido en múltiples ocasiones, sin embargo, no se ha llegado a un consenso claro y unánime que permita determinar cuál es la más adecuada (Grinderud, 2009).

Algunos autores se enfocan en su aspecto informático:

1. Tipo especializado de bases de datos, que se caracteriza por su capacidad para manejar datos geográficos, mismos que pueden ser representados gráficamente como imágenes (Bracken y Webster, 1990).

2. Tecnología informática capaz de realizar las tareas para manejar datos georeferenciados, como su entrada, almacenamiento, recuperación, manipulación, análisis y representación (Aronoff, 1989).

Otros autores dan importancia a sus componentes:

3. Es un sistema de hardware, software y procedimientos elaborados que facilita la obtención, gestión, manipulación, análisis, modelado, representación y salida de datos espacialmente referenciados, con el objetivo de resolver problemas complejos de planificación y gestión (National Center for Geographic Information and Analysis, 1990).

Finalmente, se menciona la definición que más se ha difundido en muchos años:

4. Es un conjunto de herramientas para reunir, introducir, almacenar, recuperar, transformar y cartografiar datos espaciales sobre el mundo real para un conjunto particular de objetivos (Burrough, 1990).

De forma inmediata, para el desarrollo de esta tesis se considera a un SIG como:

5. Un modelo de una parte de la realidad referido a un sistema de coordenadas terrestres que se ha construido con el propósito de satisfacer necesidades concretas de información (Longley, Goodchild y Maguire).

## 3.2 Componentes

Muchas veces se identifica de forma errónea a los SIG con el software diseñado para trabajar con los datos geográficos, pero en realidad, son más que aplicaciones instaladas en un equipo.

Un SIG se compone por cinco elementos fundamentales (Harmon y Anderson, 2003):

1. Personas
2. Hardware
3. Software
4. Datos
5. Métodos

Los elementos anteriores, interactúan en equilibrio, siempre bajo una administración central, que posee las relaciones definidas para alcanzar los objetivos propuestos.

### 3.2.1 Personas

Las personas que trabajan con los SIG constituyen una pieza clave en su funcionamiento. Entre sus principales funciones se encuentra: operar, desarrollar y administrar el sistema; además, en muchas ocasiones crean los planes que permiten llevar a cabo su correcta implementación (Gutiérrez, 2000).

Son clasificadas dentro de las siguientes categorías (Spatial Information Clearinghouse, 2004):

1. **Especialistas SIG**

   Son los encargados de la conceptualización, diseño, aplicación y uso de los SIG; además, dan mantenimiento a la base de datos geográfica y proporcionan la ayuda técnica necesaria. Incluye programadores, analistas de sistemas, administradores de base de datos, etcétera.

2. **Usuarios del sistema**

   Es un grupo multidisciplinario de profesionales que se encuentra capacitado para utilizar el software SIG con el propósito de analizar y dar solución a los problemas que se presentan. Se conforma por ingenieros, científicos, arquitectos, planificadores, urbanistas, etcétera.

3. **Usuarios finales**

   Son las personas u organizaciones que requieren de la información digital que se encuentra almacenada o es producida por los SIG, esto con el objeto de aplicarla en su trabajo diario o para el desarrollo de proyectos específicos; constituyen la clase más grande de usuarios.

### 3.2.2 Software

El software SIG es el motor de procesamiento y un componente vital para la parte operativa. Se compone de una colección integrada de programas informáticos que implementan funciones de procesamiento geográfico. Las tres partes fundamentales de un sistema de software SIG son: la interfaz de usuario (GUI), las herramientas y el sistema de gestión de bases de datos (DBMS) (Longley et al., 2005).

Los paquetes de software SIG que existen en el mercado son relativamente fáciles de utilizar, y poseen la capacidad de reconocer información geográfica estructurada en diversos formatos.

Su precio llega a variar enormemente; sin embargo, en la actualidad existen diversos proyectos que proveen productos de software SIG libre, tales como: Quantum SIG, gvSIG, MapServer, etc. (Gutiérrez et al., 2000).

Existen varias categorías de software SIG, aquí se toma como base la clasificación que realizó Paul Longley en el libro *Geographic Information Systems and Science* (2005): SIG para escritorio, DBMS espacial, servidor WebMap, servidor SIG, cliente WebSIG, SIG móvil, y librerías y extensiones. En del punto 3.4, se lleva a cabo un análisis más detallado sobre el software SIG.

### 3.2.3 Hardware

El hardware consiste en el equipo de cómputo en cual se ejecuta el software SIG. Actualmente, es posible realizar complejas operaciones SIG desde computadoras personales, razón por la cual, no es necesario contar con un equipo altamente sofisticado; sin embargo, se deben considerar aspectos como: velocidad, seguridad, escalabilidad, administración, costo y soporte. (Gutiérrez et al., 2000).

Dentro de este componente, también se consideran los periféricos requeridos para la entrada y salida de datos, tales como: escáneres, tabletas digitalizadoras, receptores GPS e impresoras.

### 3.2.4 Datos

Los datos constituyen el componente más importante para cualquier sistema de información. Es por ello que el éxito y la eficacia de un SIG se miden por el tipo, la calidad y vigencia de los datos con los que se opera; se requiere de buenos datos para obtener resultados acertados (Longley et al., 2005).

Los esfuerzos, la investigación y la inversión necesaria para crear las bases de datos que permiten tener un SIG eficiente y funcional no son pequeños; además, en algunas ocasiones el tiempo que ocupa dicho proceso es muy largo, situación que repercute directamente en el presupuesto asignado para la implementación (usualmente absorberá entre el 60% y 70%).

Las personas que implementan el SIG tienen dos opciones:

1. Capturar los datos.
2. Adquirirlos en el mercado.

La primera opción es larga y laboriosa, los datos pueden ser capturados a través de la digitalización de material cartográfico o empleando tecnologías como teledetección y GPS. Una vez capturados es de vital importancia su documentación, para ello se recurre a los metadatos geográficos; dichos elementos, permiten describir diversos aspectos, tales como: calidad, actualización, referencia aeroespacial, autor, etcétera; es importante mencionar que la norma internacional que regula los metadatos geográficos se define en la ISO 19115:2003 (Sánchez, 2008).

En cuanto a la segunda opción, cada vez es mayor la cantidad de datos que se encuentran disponibles en el mercado, por lo que es posible acudir a empresas como: Navteq y TeleAtlas; principales proveedores de datos para la creación de cartografía digital, tráfico y localización. Lamentablemente, el precio de adquisición es elevado, y en algunas ocasiones los datos geográficos no tienen el nivel de calidad requerido para ser empleados en ciertas aplicaciones (GPSReview, 2011).

Una vez que se ha concluido con el proceso de captura de los datos geográficos, resulta fundamental controlar su organización, mantenimiento y manejo; para cumplir con dichas tareas la mayoría de los SIG emplean sistemas de gestión de bases de datos (en inglés, DBMS).

### 3.2.5 Procedimientos

Para que un SIG funcione de forma adecuada, debe contar con una serie de procedimientos que definan la forma en cómo se llevaran a cabo los procesos de ingreso, almacenamiento, manejo, análisis y salida de los datos de acuerdo a las características del software y hardware disponible, la estructura administrativa de cada organización y las reglas que existen en cada disciplina (Harmon et al., 2003).

## 3.3 Datos geográficos

Los datos geográficos son más que imágenes electrónicas de mapas; constituyen entidades espacio-temporales que cuantifican la distribución, el estado y los vínculos de los distintos fenómenos u objetos naturales o sociales (Domech y Mosqueras, 1995).

Con ellos es posible crear un modelo de la realidad que permite conocer lo que ocurre (qué), sobre una determinada posición del espacio (dónde), y para un tiempo especifico (cuándo) (Sánchez et al., 2008).

### 3.3.1 Componente espacial

La componente espacial define la localización geográfica y las propiedades espaciales de los objetos; además, es responsable de conservar las relaciones espaciales que existen entre ellos (CIAF, 2011).

1. **Localización geográfica**

   Se expresa mediante un sistema de coordenadas; es importante que dicho sistema sea el mismo para las distintas capas de información con las que se representa el área de estudio.

2. **Propiedades espaciales**

   Los objetos utilizados para representar la realidad poseen ciertas propiedades espaciales. Ejemplo, una línea se encuentra compuesta por su longitud, forma, pendiente y orientación.

3. **Relaciones espaciales**

   Los objetos espaciales mantienen relaciones entre sí que basan en el espacio, tales como: conectividad, contigüidad, proximidad, etc.

### 3.3.2 Componente temática

La componente temática son los atributos o características asociadas a los objetos con los que se modela el mundo real (nombre, valor, etc.); no tienen extensión especifica, son almacenados en tablas, y se encuentran relacionados a los objetos que modelan a través de identificadores comunes que se almacenan en la base de datos. También se le llama componente descriptiva (CIAF et al., 2011).

### 3.3.3 Componente temporal

La componente temporal se refiere al instante en el que fue adquirido el dato o respecto al que se ha estimado; es importante mencionar que las distribuciones espaciales varían con el transcurso del tiempo, por lo que, puede llegar a afectar la componente temática o espacial (CIAF et al., 2011).

La consideración de la dimensión temporal en un SIG, demanda almacenar y tratar grandes volúmenes de datos, ya que cada estrato, capa o nivel de información se debe almacenar tantas veces como momentos temporales se consideren para efectuar el análisis del área de estudio (CIAF et al., 2011).

## 3.4 Modelos de datos

La principal diferencia del software SIG existente en el mercado es la manera en cómo realizan la representación de los datos geográficos; sin embargo, la gran mayoría se caracteriza por emplear uno o dos modelos de representación: vectorial y raster (Gutiérrez et al., 2000).

El modelo de datos elegido es fundamental a la hora de implementar un SIG, ya que de ello depende el modo de visualización de los datos y el tipo operaciones de análisis que se realicen (Longley et al., 2005).

### 3.4.1 Modelo vectorial

En el modelo de datos vectorial la representación de los objetos del mundo real se efectúa con tres elementos geométricos: el punto, la línea y el polígono (ver figura 3.1).

1. **Puntos**

   Los puntos se utilizan para representar entidades geográficas que pueden ser expresadas por un único punto de referencia, tales como ubicaciones o zonas a escalas muy pequeñas. Dichos elementos no pueden ser medidos debido a que la dimensión de un punto es cero.

2. **Líneas**

   Las líneas unidimensionales o polilíneas son empleadas para representar rasgos lineales: ríos, caminos, ferrocarriles, líneas topográficas, curvas de nivel, entre otros; además, se utilizan para representar polígonos. En los elementos lineales se puede medir su distancia.

3. **Polígonos**

   Los polígonos bidimensionales son utilizados para representar los elementos geográficos que cubren un área de la superficie de la tierra; estas entidades pueden llegar a modelar: lagos, edificios, provincias, etcétera. En los polígonos se puede calcular su perímetro y área.

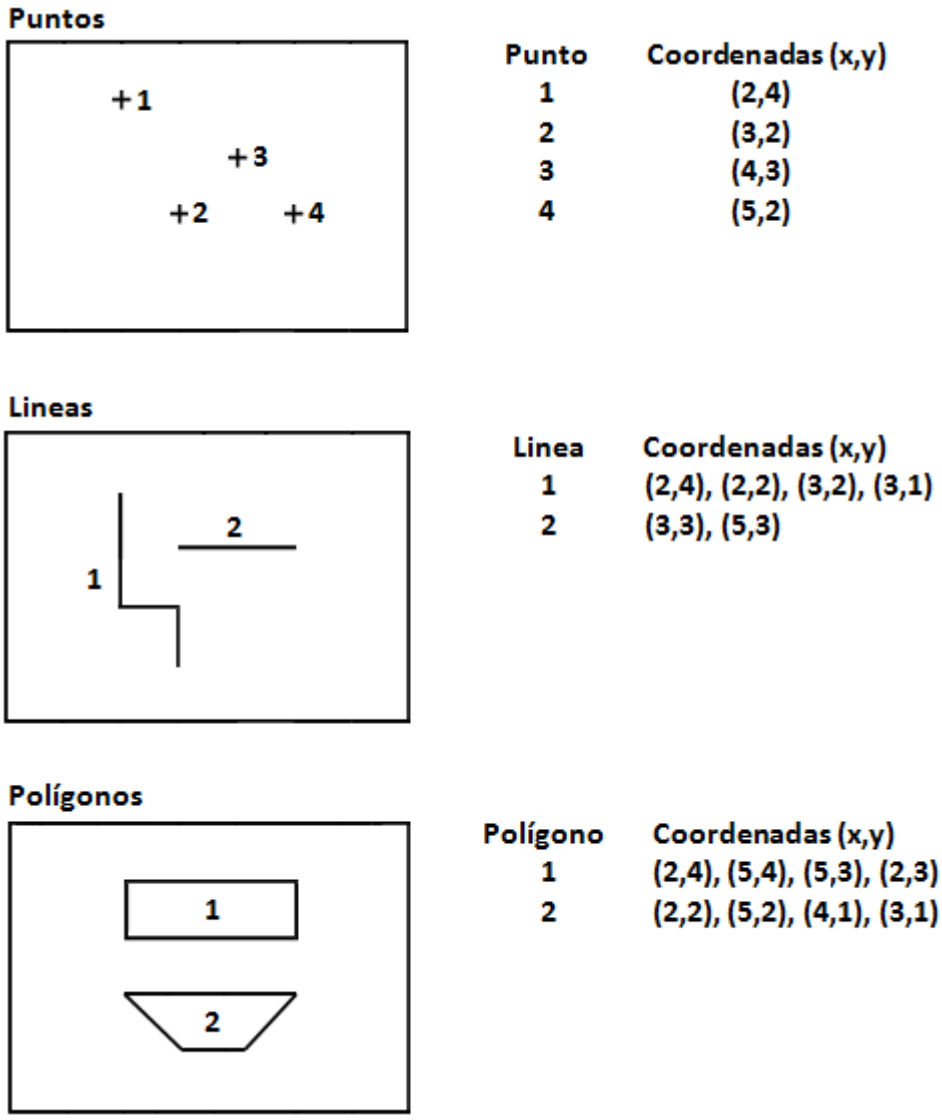


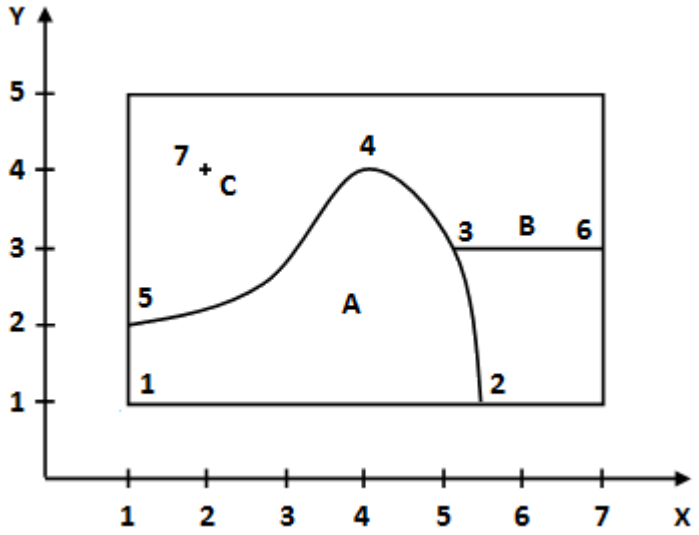

**Figura 3.1** *Representación de un punto, una línea y un polígono empleando el modelo vectorial* (Longley et al., 2005, p.184)

El modelo vectorial es adecuado cuando se trabaja con entidades que poseen límites establecidos (objetos discretos); en este modelo, el interés de las representaciones se centra principalmente en la precisión de la localización de los objetos geográficos sobre el espacio.

Los modelos vectoriales más comunes son: la lista de coordenadas o estructura espagueti, el diccionario de vértices y la topología arco-nodo; cada uno posee ciertas ventajas y desventajas. A continuación se describen en forma breve las principales características de dichos modelos (Longley et al., 2005).

## 1. Modelo espagueti

Consiste en una estructura de datos simple, sin topología, en la cual cada objeto espacial se almacena mediante un identificador, seguido por la lista de coordenadas de los vértices que definen la posición del objeto en el espacio; en caso de que los objetos sean polígonos, se deben repetir las coordenadas del primer vértice para indicar que es una figura cerrada (ver figura 3.2).

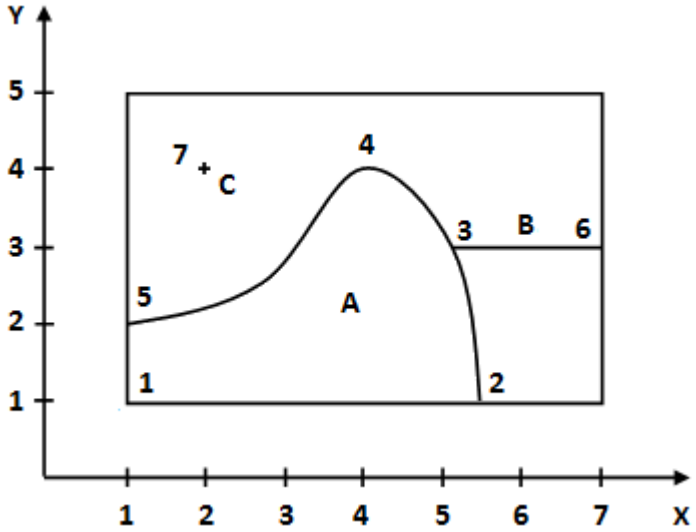



| **A, 5** | **Identificador del polígono y número de vértices** |
|---|---|
| 1, 1 | Coordenadas (X, Y) del vértice 1 |
| 5.5, 1 | Coordenadas (X, Y) del vértice 2 |
| 5.2, 3 | Coordenadas (X, Y) del vértice 3 |
| 4, 4 | Coordenadas (X, Y) del vértice 4 |
| 1, 2 | Coordenadas (X, Y) del vértice 5 |
| 1, 1 | Coordenadas (X, Y) del vértice 1 |
| **B, 2** | **Identificador de la línea y número de vértices** |
| 5.2, 3 | Coordenadas (X, Y) del vértice 3 |
| 7, 3 | Coordenadas (X, Y) del vértice 6 |
| **C, 1** | **Identificador del punto y número de vértices** |
| 2, 4 | Coordenadas (X, Y) del vértice 7 |

**Figura 3.2** *Ejemplo modelo espagueti*
(Elaboración propia)

## 2. Diccionario de vértices

Esta estructura representa los objetos a través de dos listas: la primera se conforma por las coordenadas de localización que posee cada vértice; la segunda lista, define los vértices que conforman cada objeto espacial. Aunque resuelve los problemas de repetición de vértices que se presentan en el modelo espagueti, es muy pobre desde el punto de vista topológico (ver figura 3.3).



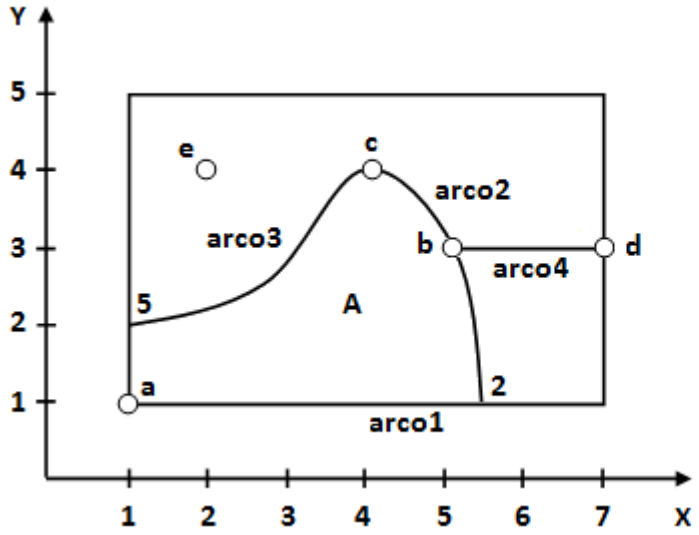

| COORDENADAS DE LOS VERTICES | |
|---|---|
| **Vértice** | **Coordenadas (X,Y)** |
| 1 | (1, 1) |
| 2 | (5.5, 1) |
| 3 | (5.2, 3) |
| 4 | (4, 4) |
| 5 | (1, 2) |
| 6 | (7, 3) |
| 7 | (2, 4) |

| VERTICES DE LOS OBJETOS | |
|---|---|
| **Estructura** | **Vertices** |
| polígono A | 1, 2, 3, 4, 5 |
| línea B | 3, 6 |
| punto C | 7 |

**Figura 3.3** *Ejemplo diccionario de vértices* (Elaboración propia)

## 3. Estructura arco-nodo

Consiste en dos entidades básicas: arcos y nodos. Los arcos son sucesiones de puntos que se encuentran unidos por líneas rectas, que inician y finalizan en nodos; por su parte, los nodos constituyen el punto de intersección entre dos o más arcos. Esta estructura se caracteriza por conservar las propiedades topológicas de conectividad, adyacencia y definición de área (ver figura 3.4).




| TOPOLOGIA DE NODOS | |
|---|---|
| Nodo | Arcos |
| a | 1,3 |
| b | 1,2,4 |
| c | 2,3 |
| d | 4 |
| e | - |

| TOPOLOGIA DE POLIGONOS | |
|---|---|
| Polígono | Arcos |
| A | 1,2,3 |

| TOPOLOGIA DE ARCOS | | | | |
|---|---|---|---|---|
| Arco | Nodo inicial | Nodo final | Polígono derecha | Polígono izquierda |
| 1 | a | b | Externo | A |
| 2 | b | c | Externo | A |
| 3 | c | a | Externo | A |
| 4 | b | d | Externo | Externo |

| COORDENADAS DE ARCO (X,Y) | | | |
|---|---|---|---|
| Arco | Nodo inicial | Vertices intermedios | Nodo final |
| 1 | (5.5, 1) | (5.5, 1) | (5.2, 3) |
| 2 | (5.2, 3) | - | (4, 4) |
| 3 | (4, 4) | (0, 3) | (0, 0) |
| 4 | (5.2, 3) | - | (7, 3) |

**Figura 3.4** *Ejemplo estructura arco-nodo*
(Elaboración propia)

### 3.4.2 Modelo raster

El modelo raster divide el espacio en una malla que se conforma por un conjunto de elementos que se denominan celdas o pixeles; en dicha malla, cada celda almacena las coordenadas de la localización y del valor temático que corresponden a un determinado aspecto del mundo real. Ya que registra el interior de los objetos en lugar de codificar sus fronteras, en el modelo raster los límites quedan implícitamente representados por las celdas que contienen el mismo valor (Longley et al., 2005).

El modelo de datos raster es adecuado cuando se trabaja con entidades que llegan a presentar variables continuas que varían de forma gradual en el espacio, tales como: dispersión de nubes de contaminantes, distribuciones de temperaturas, elevaciones en la superficie terrestre, etc.

Las estructuras raster más comunes son: la enumeración exhaustiva y la codificación por grupos de longitud variables; a continuación, se describen brevemente sus principales características.

**1. Enumeración exhaustiva**

En esta estructura de datos el valor de cada pixel se almacena individualmente, de forma que no se aplica ningún método de compresión para los elementos que aparecen repetidos (ver figura 3.5).

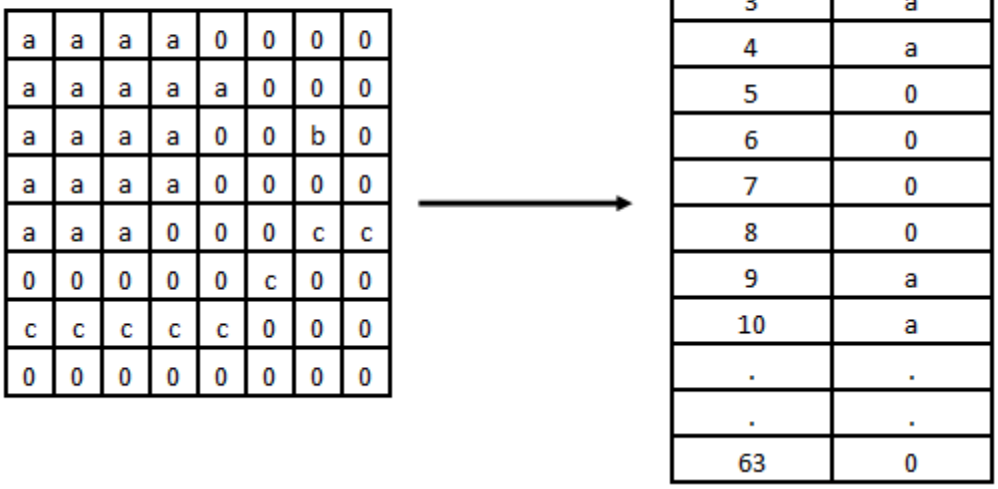



**Figura 3.5** *Ejemplo enumeración exhaustiva*
(Elaboración propia)

**2. Codificación por grupos de longitud variable**

Es una estructura muy simple de compresión de imágenes; en lugar de guardar de forma individual el valor de cada pixel, registra el valor de cada pixel y cuantas veces se repite (ver figura 3.6).

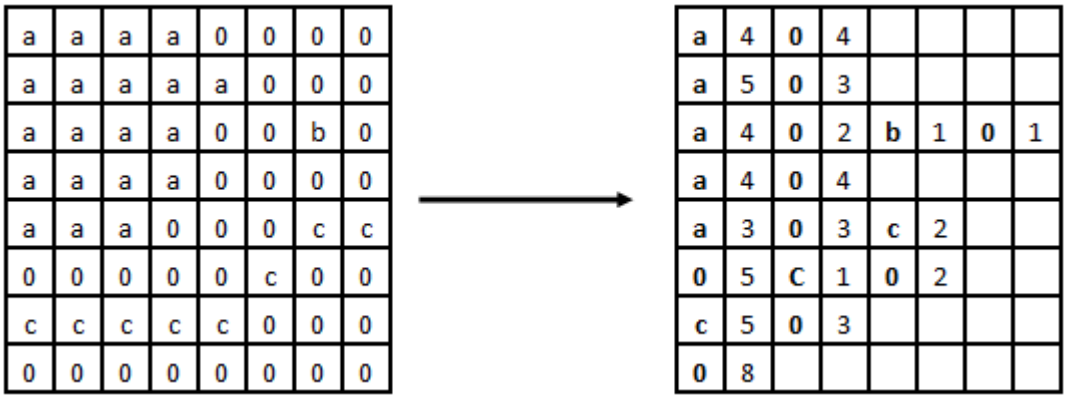



**Figura 3.6** *Ejemplo codificación por grupos de longitud variable*
(Elaboración propia)

## 3.4.3 Vectorial vs Raster

Al momento de representar la realidad tanto el modelo vectorial como el raster poseen ciertas ventajas y desventajas (ver tabla 3.1), por lo que, ningún modelo es mejor que el otro en términos absolutos (Longley et al., 2005).

**Tabla 3.1** *Ventajas y desventajas de los modelos vectorial y raster*
(Elaboración propia)

| | Raster | Vectorial |
|---|---|---|
| precisión gráfica | − | + |
| manejo de cartografía tradicional | − | + |
| volumen de datos | − | + |
| construcción topológica | − | + |
| operaciones de cálculo | + | − |
| actualización de datos geográficos | + | − |
| variación espacial continua | + | − |
| variación espacial discontinua | − | + |

Existen diversos artículos escritos y debates en torno a qué modelo utilizar; sin embargo, al implementar un SIG es fundamental considerar los siguientes puntos (Sánchez et al., 2008):

1. **Características del dato geográfico**

   Como se mencionó anteriormente, las variables continuas suelen representarse mediante el modelo raster; no obstante, nada impide que esto se lleve a cabo con el modelo vectorial. Lo que ocurre es lo siguiente: en los modelos vectoriales los objetos gráficos se definen por su contorno o borde, de manera que cuanto más continua sea la variable, más contornos o bordes deben ser dibujados para realizar la representación de forma óptima; por esta razón el modelo vectorial resulta mucho más adecuado para trabajar con las variables discretas.

2. **Actualización de los datos geográficos**

   En los modelos vectoriales la actualización de los datos se realiza de forma manual sobre los nuevos mapas o tomando como referencia imágenes sobre las que se representa la antigua información vectorial; ya que se realiza manualmente, constituye un procedimiento lento, tedioso y sujeto a errores humanos. Por el contrario, en los modelos raster la actualización de los datos se realiza de forma automática, a partir de imágenes digitales provenientes de sensores remotos, que tras su adecuado tratamiento, se convierten en datos geográficos.

3. **Velocidad de análisis geográfico**

   Por su estructura de almacenamiento en forma de listas de coordenadas, los SIG vectoriales necesitan recurrir constantemente a un largo proceso de comparación de listas para realizar el más sencillo análisis. Sin embargo, los SIG raster actúan por operación matricial, lo que permite obtener una mayor velocidad; una gran ventaja de esta propiedad es la posibilidad de poder utilizar equipos de computo más sencillos, como las PC comunes, lo que permite al usuario reducir el tiempo de aprendizaje, ya que se conoce el equipo en el que se trabaja.

4. **Topología**

   A través de la topología se define la relación espacial existente entre los datos geográficos. En el caso del modelo vectorial, la topología debe ser calculada explícitamente o definida implícitamente al introducir los datos. El primer caso es matemáticamente más perfecto y ocupa menos almacenamiento; sin embargo, se debe calcular cada vez que se agrega algún elemento. En el segundo caso, la topología implícitamente definida al introducir los datos, es más rápida y operativa; lamentablemente, solo existe en algunos sistemas SIG recientes.

Finalmente, es importante mencionar que actualmente la mayoría del software SIG existente, tanto comercial como libre, incorpora elementos de ambas técnicas de representación, y en algunos casos, posee diversas extensiones que permiten la conversión entre modelos de datos.

## 3.5 Software SIG

El software que se utiliza para crear, gestionar, analizar y visualizar datos geográficos se denota con el término "*software SIG*". Algunas de las aplicaciones comunes del software SIG incluyen la evaluación de los lugares para la ubicación de nuevas tiendas, la gestión de las líneas eléctricas y de gas, la creación de mapas, cálculos de rutas para tareas de transporte, la administración de bosques, parques e infraestructuras, tales como carreteras y líneas de agua; así como el análisis de riesgo ante desastres naturales, con los respectivos planes de emergencia y respuesta.

Para esta multitud de aplicaciones, existen diferentes paquetes de software SIG; todos proveen un conjunto de particular de funciones SIG para cumplir con ciertas tareas de gestión de datos.

### 3.5.1 Arquitectura

La arquitectura de un software SIG posee tres capas: presentación, lógica de negocio y datos (ver figura 3.7).

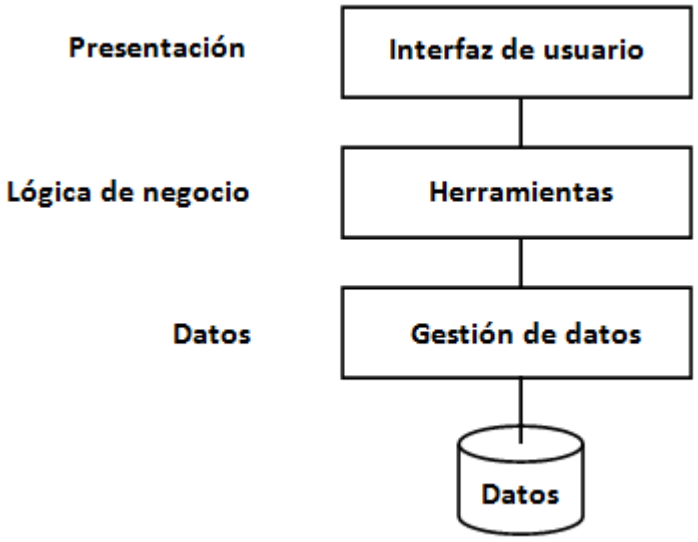



**Figura 3.7** *Arquitectura de tres capas del software SIG*
(Longley et al., 2005, p.160)

1. **Capa de presentación**

   En la capa de presentación el usuario interactúa con el sistema a través de la interfaz gráfica de usuario (GUI), una colección integrada de menús, barras de tareas y otros controles. La GUI proporciona el acceso a las herramientas SIG. El conjunto de herramientas define las funciones que el software SIG posee para el procesamiento de datos geográficos. Además, esta capa es la encargada de representar (mostrar) e interactuar con los objetos gráficos.

2. **Capa de lógica de negocio**

   La capa de lógica de negocio es responsable de realizar el conjunto de operaciones que el usuario selecciono en la GUI, por ejemplo, cálculos de distancias, agregaciones de atributos, superposiciones de distintos mapas, entre otras. Además, es en esta capa donde se realiza la implementación del modelo de datos. Cabe mencionar, que los paquetes de software SIG que existen en el mercado se diferencian por su nivel y orientación de lógica de negocio.

3. **Capa de datos**

   La capa de datos es responsable de almacenar, ordenar y recuperar los datos. Los datos son almacenados en archivos o en bases de datos, y se encuentran organizados por un DBMS. El DBMS funciona como interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Sus principales funciones son: reducir la redundancia de datos, integrar datos de múltiples archivos, recuperar los datos con rapidez, y garantizar la seguridad de los datos.

Con el fin de maximizar el rendimiento del sistema, se recomienda optimizar el hardware y las configuraciones de los sistemas operativos de forma diferente para cada una de las capas. Por ejemplo, para generar los mapas digitales se requieren grandes cantidades de memoria y potentes procesadores; mientras que realizar consultas a las bases de datos requiere de veloces unidades de disco y bus para el almacenamiento y movimiento de grandes cantidades de datos.

### 3.5.2 Categorías de software SIG

Los productos de software SIG existentes en el mercado proveen funcionalidades específicas. De acuerdo a su funcionalidad, la tabla 3.2 muestra las categorías de software SIG existentes.

Los denominados SIG para escritorio (Desktop GIS) permiten llevar a cabo todas las tares SIG. Además, usualmente se divide en las subcategorías: SIG visualizador, SIG editor, SIG analista.

Los sistemas de gestión de bases de datos espaciales (DBMS), casi siempre son utilizados para almacenar datos, pero en algunas ocasiones proporcionan (de forma limitada) las funciones de análisis y manipulación de datos. Los servidores WebMap son utilizados para distribuir mapas y datos a través de internet. Del mismo modo, los clientes WebSIG se utilizan para mostrar los datos y para acceder a las funciones de análisis y consulta que se encuentran en servidores SIG mediante internet o intranet. Las librerías y extensiones proveen funciones adicionales para el software SIG. Por último, los SIG móviles se utilizan normalmente para la recolección de datos (Steiniger y Weibel, 2010).

**Tabla 3.2** *Funciones disponibles para las categorías de software SIG*
*● Funcionalidad estándar, ◊ Funcionalidad opcional*
(Steiniger et al., 2010, p.3)

| Funciones SIG vs Categorías SIG | Visualizar | Crear | Editar | Almacenar | Integrar | Transformar | Consultar | Analizar | Crear mapas |
|---|---|---|---|---|---|---|---|---|---|
| **Sig para escritorio** | | | | | | | | | |
| visualizador | ● | | | ● | | | ● | | ◊ |
| editor | ● | ● | ● | ● | | ◊ | ● | | ● |
| analista | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| **DBMS espacial** | | | | ● | | ● | ● | ◊ | |
| **Servidor WebMap** | ● | ◊ | ◊ | | | | ● | | ● |
| **Servidor SIG** | | | | ● | ● | ● | ● | ● | ● |
| **Cliente WebSIG** | | | | | | | | | |
| ligero | ● | | | | | | ● | | |
| pesado | ● | ● | ● | ● | | | ● | ● | ● |
| **SIG móvil** | ● | ● | ● | ● | | | ● | | |
| **Librerías y extensiones** | | | | ● | ● | ● | | ● | ● |

### 3.5.3 Fabricantes de software SIG y proyectos

El software SIG no solo es creado por compañías privadas; actualmente, es posible encontrar diversos proyectos de software SIG libre y de código abierto. Mientras las compañías privadas ofrecen productos para todas las categorías que se han descrito; los proyectos abiertos suelen enfocarse casi siempre en las categorías de software SIG para escritorio y servidor WebMap (Steiniger et al., 2010).

Las empresas más importantes que proveen software SIG hoy en día son: AutoDesk, Bentley, ESRI Inc., GE (Smallworld), Pitney Bowes (MapInfo), e Intergraph. Las empresas de software SIG tienden a centrarse en áreas específicas. Por ejemplo, ArcGIS de ESRI se utiliza principalmente para el análisis de negocios, planificación y aplicaciones ambientales; mientras que AutoDesk, GE y los productos de Bentley, se utilizan en la gestión de servicios públicos e infraestructura (Steiniger et al., 2010).

En los proyectos de software SIG libre destacan las aplicaciones para servidor MapServer y GeoServer, el DBMS espacial PostGIS, y dentro del software SIG para escritorio Quantum GIS y gvSIG. El software SIG libre complementa el software propietario en lugar de competir con él.

## 3.6 TransCAD

TransCAD es un sistema de información geográfica (SIG) diseñado especialmente para los profesionales de transporte con el propósito principal de almacenar, mostrar, y analizar datos de transporte. A diferencia de los demás paquetes informáticos de transporte, combina en una sola plataforma las propiedades de un SIG y las capacidades de modelación del transporte (Caliper, 2010).

Dentro de esta tesis, TransCAD permite visualizar la representación geográfica del patrón de flujo que ha sido generada mediante la implementación del modelo de equilibrio del usuario. Para ello, la aplicación de escritorio accede a TransCAD a través de los servicios que provee .NET.

### 3.6.1 GISDK$^{tm}$

El kit de desarrollo de sistema de información geográfica (GISDK$^{tm}$), provee un conjunto de herramientas y documentación necesaria para crear: Macros (add-ins) que permiten generar nuevas funcionalidades o automatizar tareas, interfaces adaptadas que amplían o modifican la interfaz estándar de TransCAD, aplicaciones de escritorio que utilizan los servicios de TransCAD para generar mapas y ver los datos asociados con sus características en formularios tabulares (Caliper et al., 2010).

El componente principal del GISDK$^{tm}$ es el lenguaje de programación llamado Caliper Script. La principal razón para utilizar Caliper Script es interactuar con TransCAD. Existen cerca de mil funciones GISDK en TransCAD; todas ellas pueden ser llamadas con el lenguaje Caliper Script.

A su vez, Caliper Script se conforma por un compilador, un debugger y un toolbox. El compilador GISDK toma el código Caliper Script y crea la UIDatabase, que a su vez, puede ser ejecutada por la plataforma de TransCAD; cualquier error del código es notificado por el compilador, dando detalles sobre el tipo de error y su localización. El debugger GISDK ejecuta el código en modo de prueba para asegurarse de que no hay errores. El toolbox GISDK posee botones para ejecutar el compilador y el debugger, y herramientas para interactuar fácil y rápidamente con el programa.

### 3.6.2 Funciones GISDK

Como se mencionó, en GISDK$^{tm}$ existen cerca de mil funciones que pueden ser llamadas mediante CaliperScript para trabajar con elementos de TransCAD, tales como: ventanas, mapas, capas, editores, tablas y vistas; de igual forma existen funciones que permiten analizar información geográfica, trabajar con redes, sistemas de rutas, matrices y realizar otras tareas disponibles.

A continuación se listan las funciones GISDK que han sido utilizadas para trabajar con TransCAD:

1. **AddLayer()**

   Agrega una capa a un mapa desde un archivo geográfico.

2. **CreateContinuosTheme()**

   Crea un tema continuo (escala un símbolo) de la capa actual.

3. **CreateLegend()**

   Crea la leyenda del mapa.

4. **GetLayerPosition()**

   Obtiene la posición de una capa en el mapa.

5. **GetView()**

   Obtiene el nombre de la vista actual.

6. **JoinViews()**

   Genera un join ligando campos de vistas existentes.

7. **OpenMap()**

   Abre un archivo .MAP y lo despliega como mapa dentro de una ventana.

8. **OpenTable()**

   Abre una tabla desde un archivo del disco y crea una vista.

9. **RunMacro()**

   Ejecuta un macro.

10. **SetArrowheads()**

    Establece el estilo de flechas que se dibujan en las líneas.

11. **SetLabels()**

    Activa y desactiva opciones para el etiquetado automático para una capa.

12. **SetLayer()**

    Establece la capa actual (y la vista actual).

13. **SetLayerPosition()**

    Establece la posición de una capa en el mapa.

14. **SetLineColor()**

    Establece el color de las líneas que se dibujan.

15. **SetTheme()**

    Despliega el tema en la capa actual.

### 3.6.3 Accediendo a TransCAD desde .NET

CALIPERFORM.DLL debe agregarse como referencia en la carpeta del proyecto de la aplicación. Las clases .NET incluidas en CALIPERFORM.DLL permiten acceder al entorno GISDK™ desde una aplicación de escritorio de Windows (Windows Forms) escrita en cualquier lenguaje de .NET.

La clase CaliperForm.Connection se encarga de realizar la conexión con el entorno GISDK™, permitiendo ejecutar funciones GISDK y macros, crear mapas y seleccionar datos geográficos. Ademas, esta clase incluye métodos para convertir datos entre los entornos GISDK™ y .NET.

Ya que la implementación del modelo de equilibrio del usuario se desarrolló con el lenguaje C#, el código para acceder a TransCAD y ejecutar una Macro desde una aplicación de escritorio es el siguiente:

```
1.   using CaliperForm;
2.   CaliperForm.Connection Gisdk;
2.   Gisdk = new CaliperForm.Connection();
3.   Gisdk.MappingServer = "TransCAD";
4.   Gisdk.Open();
5.   Gisdk.DoMacro(String NameMacro, String UIDatabase,
                   params object[] Arguments);
6.   Gisdk.Close();
```

Durante el proceso de implementación del algoritmo y los módulos encargados de la conexión de la aplicación con el SIG TransCAD, se empleó un modelo para estabilizar, controlar y organizar las diferentes actividades relacionadas con la creación, presentación y mantenimiento del software. El modelo que ha sido empleado en esta tesis es el "*Proceso Unificado de Desarrollo de Software*", propuesto por Ivar Jacobson, Grady Booch y James Rumbaugh, por lo que en el siguiente capítulo serán abordadas sus principales características.

# Capítulo 4

# Proceso Unificado

Un modelo de software proporciona un conjunto de normas para el desarrollo eficiente de éste. En la actualidad es posible encontrar diferentes modelos para el desarrollo de software: Programación Extrema (XP), Diseño Rápido de Aplicaciones (RAD), Proceso Unificado (UP), Desarrollo Guiado por Funcionalidades (FDD), entre otros; lo que origina la siguiente pregunta: ¿qué modelo de desarrollo de software debe ser utilizado para llevar a cabo la implementación?

El objetivo de este capítulo es describir el Proceso Unificado y justificar su utilización para la correcta implementación del algoritmo que da solución al problema de equilibrio del usuario.

## 4.1 Uso del proceso Unificado

Debido al carácter de investigación de este proyecto y a la existente necesidad de modificar los requisitos que surgirían según se vayan evaluando y probando las distintas posibilidades con las que se cuenta para la construcción, un modelo de desarrollo de software pesado no se ajusta de manera adecuada. Sin embargo, un modelo puramente ágil necesita de un equipo de desarrollo con experiencia para ser llevado a cabo de manera satisfactoria; ya que el LTST carece de él, tampoco representa el caso más adecuado para llevar a cabo la implementación.

Ante dicha situación, se ha optado por un modelo que combina características de ambas orientaciones, y además, proporciona un enfoque iterativo e incremental, dicho modelo es: "*El Proceso Unificado de Desarrollo de Software",* propuesto por Ivar Jacobson, Grady Booch y James Rumbaugh (2003).

El Proceso Unificado se ha convertido en un modelo de desarrollo de software ampliamente utilizado para la construcción de aplicaciones orientadas a objetos; se caracteriza por estar dirigido por casos de uso, estar centrado en la arquitectura y por ser iterativo e incremental (Jacobson et al., 2003). Cabe destacar que hace uso del Lenguaje Unificado de Modelado (UML) para representar visualmente todos los artefactos que conforman el sistema de software.

## 4.2 La idea más importante del UP: desarrollo iterativo

El UP fomenta muchas buenas prácticas, pero una destaca sobre todas: *el desarrollo iterativo*. Dentro de este enfoque el desarrollo se lleva a cabo a través de una serie de proyectos cortos de duración fija, a los que se les conoce como iteraciones; el resultado de cada uno de ellos es un sistema que puede ser probado, integrado y ejecutado, por lo que, cada iteración debe contar con sus propias actividades de análisis de requisitos, diseño, implementación y pruebas (Larman et al., 2003).

El ciclo de vida iterativo se basa en la ampliación y refinamiento sucesivos del sistema mediante múltiples iteraciones, con retroalimentación cíclica y adaptación como elementos principales que dirigen para converger hacia un sistema adecuado. En la figura 4.1 es posible observar como el sistema crece incrementalmente a lo largo del tiempo, iteración tras iteración, y por ello, este enfoque también es conocido como: *desarrollo iterativo e incremental* (Larman, 2003).

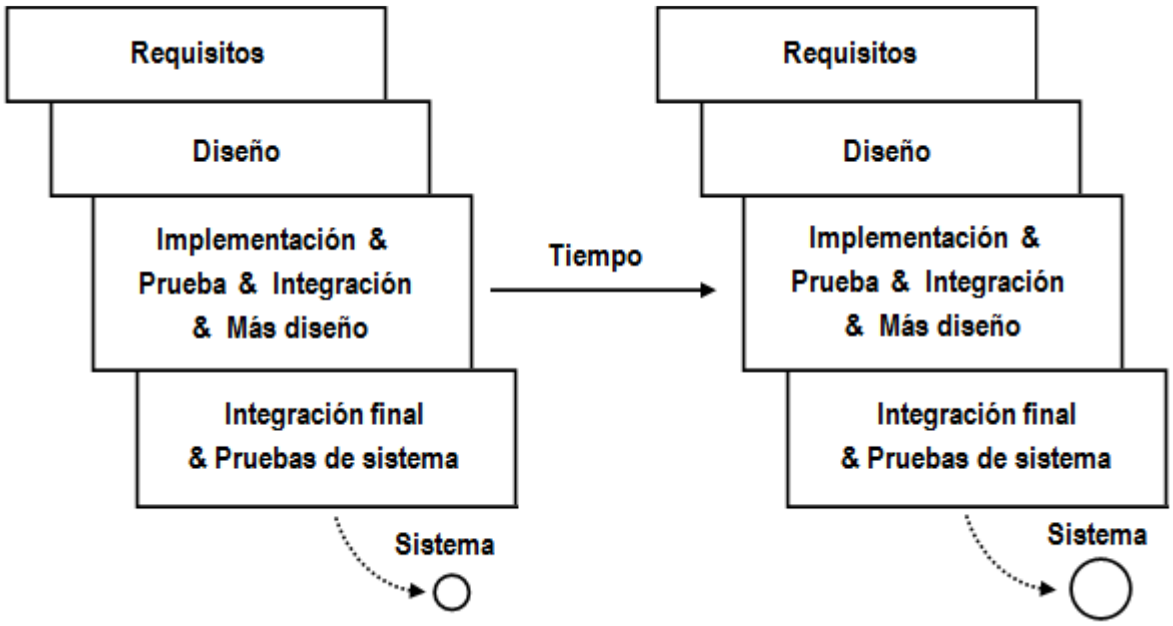



**Figura 4.1** *Desarrollo iterativo e incremental*
(Larman et al., 2003, p.14)

Como se mencionó anteriormente, el resultado de cada iteración es un sistema ejecutable, pero incompleto, ya que no está preparado para ser puesto en producción. El sistema podría estar listo para su puesta en producción después de muchas iteraciones, por ejemplo, 10 o 15.

### 4.2.1 Aceptando los cambios: retroalimentación y adaptación

Una característica clave del desarrollo iterativo es aceptar el cambio. En lugar de luchar contra el inevitable cambio que ocurre en el desarrollo de software, intentando sin éxito, especificar y definir de forma completa los requisitos antes de la implementación. El desarrollo iterativo se basa en la aceptación del cambio y en la adaptación como motores inevitables y esenciales (Larman et al., 2003).

Esto no quiere decir que el UP fomenta un proceso dirigido por una adición de características de manera incontrolada y reactiva; para evitar esta situación, el UP llega a un equilibrio entre la necesidad de estabilizar un conjunto de requisitos, y por otro lado considerar la realidad de los requisitos cambiantes (siendo aquellos que se modifican cuando el personal clarifica su visión).

Cada iteración trabaja con un pequeño conjunto de requisitos, lo que permite rápidamente diseñar, implementar y probar. En las primeras iteraciones, la elección de los requisitos y el diseño podrían no ser exactamente lo que se desea al final, pero la acción de dar un pequeño paso con rapidez, antes de capturar todos los requisitos, y haber definido el diseño de forma especulativa, proporciona una rápida retroalimentación entre usuarios y desarrolladores.

Tener retroalimentación en una etapa temprana del desarrollo posee vital importancia; más que las especulaciones sobre los requisitos y diseños correctos, la retroalimentación generada a partir de la construcción y prueba realista de algo, aporta un conocimiento práctico y crucial, además de la oportunidad de modificar o adaptar la comprensión de los requisitos o el diseño.

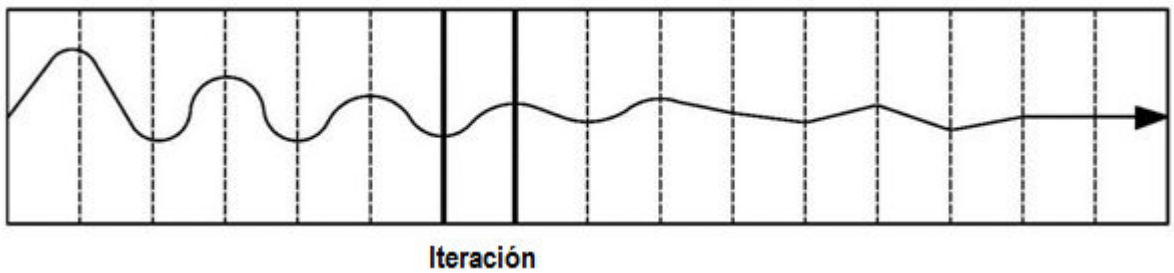



**Figura 4.2** *La retroalimentación y adaptación conducen hacia el sistema deseado*
*La inestabilidad de los requisitos disminuye a lo largo del tiempo*
(Larman et al., 2003, p.16)

Considerando lo anterior, el trabajo se realiza mediante una serie de ciclos estructurados de construir-retroalimentar-adaptar; no sorprende que la desviación del sistema del verdadero camino (en términos de sus requisitos y diseños finales) en las primeras iteraciones es mayor que en las últimas. La figura 4.2 ilustra como el sistema converge hacia el verdadero camino.

### 4.2.2 Beneficios del desarrollo iterativo

Los beneficios del sistema iterativo incluyen (Larman et al., 2003,):

- Mitigación temprana de altos riesgos (técnicos, requisitos, objetivos, usabilidad y demás).
- Progreso visible en primeras etapas.
- Temprana retroalimentación, compromiso de los usuarios y adaptación.
- Gestión de la complejidad; el equipo no se ve abrumado por pasos muy largos y complejos.
- El conocimiento adquirido en una iteración se puede utilizar metódicamente para mejorar el propio proceso de desarrollo, iteración a iteración.

### 4.2.3 Longitud de una iteración y fijación de la duración

El UP y los desarrolladores con experiencia en aplicar procesos iterativos, recomiendan que la longitud de una iteración sea de dos a seis semanas. Pasos pequeños, rápida retroalimentación y adaptación son las dos ideas fundamentales del desarrollo iterativo; las iteraciones largas destruyen la motivación principal del desarrollo iterativo e incrementan el riesgo del proyecto (Larman et al., 2003).

Fijar la duración de las iteraciones es clave. Por ejemplo, si se elige que una iteración dure cuatro semanas, el sistema debe integrarse, probarse y estabilizarse en la fecha planificada. Si parece que será difícil cumplir con el plazo fijado, se recomienda eliminar tareas o requisitos de la iteración, e incluirlos en una iteración posterior, para evitar retrasar la fecha de terminación.

Rara vez, equipos de trabajo muy grandes (por ejemplo, varios cientos de desarrolladores), podrían requerir iteraciones de más de seis semanas para recuperar el tiempo perdido en la coordinación y comunicación; aun así, no es recomendable que dure más de seis semanas. Un equipo formado por 10 o 20 desarrolladores puede dividir su trabajo en varias iteraciones (Larman et al., 2003).

## 4.3 Fases del UP

Un ciclo en el modelo UP organiza el trabajo y las iteraciones en cuatro fases fundamentales (Jacobson et al., 2003):

1. **Inicio**

   Visión aproximada, análisis del negocio, alcance y estimaciones imprecisas.

2. **Elaboración**

   Visión refinada, implementación iterativa del núcleo central de la arquitectura, resolución de los altos riesgos, estimaciones más realistas e identificación de más requisitos.

3. **Construcción**

   Implementación iterativa del resto de requisitos de menor riesgo y elementos más fáciles, preparación para el despliegue.

4. **Transición**

   Pruebas beta, despliegue.

Esto no corresponde con el antiguo ciclo en cascada o secuencial, en el que primero se definían todos los requisitos y después se realizaba todo o la mayoría del diseño. Por ejemplo, la fase de diseño no es una fase de requisitos; sino una especie de fase de viabilidad, donde se realiza sólo el estudio suficiente para decidir si continuar o no. De igual forma, la fase elaboración no es la fase de diseño; sino constituye una fase donde se disminuyen las cuestiones de alto riesgo, además de implementar de manera iterativa, la arquitectura que constituye el núcleo central.

Cada fase termina con un hito. De manera informal, un hito es un punto de terminación que se determina por la disponibilidad de un conjunto de artefactos; es decir, ciertos modelos o documentos han sido desarrollados hasta alcanzar un estado predefinido. Los hitos tienen muchos objetivos; el más importante es que el grupo de trabajo debe tomar ciertas decisiones cruciales antes de que el trabajo pueda continuar con la siguiente fase (Larman et al., 2003).

Cada ciclo finaliza con una nueva versión del sistema. Una versión del sistema es un producto preparado para su entrega, consta de código fuente, manuales y otros artefactos de trabajo.

### 4.3.1 Flujos de trabajo del UP

Un flujo de trabajo (disciplina) es un conjunto de actividades (y artefactos relacionados) que se desarrollan en un área determinada; en el UP, un artefacto es el término general que se emplea para nombrar cualquier producto de trabajo: código, texto, diagramas, modelo, manuales, etc. (Jacobson et al., 2003).

Existen varios flujos de trabajo en el UP; sin embargo, esta tesis se centrará en los siguientes:

1. **Requisitos**

   Es el análisis de los requisitos para una aplicación, como escritura de casos de uso e identificación de los requisitos no funcionales.

2. **Diseño**

   Todos los aspectos de diseño, incluyendo arquitectura global, objetos, datos, entre otros.

3. **Implementación**

   Constituye la acción de programar y construir el sistema.

4. **Prueba**

   Ejecuta una objetiva evaluación que permite validar que el sistema funciona de acuerdo a los requerimientos establecidos.

Una lista más extensa de los flujos de trabajo que conforman el UP se muestra en la figura 4.3.

### 4.3.2 Disciplinas y fases

Como se ilustra en la figura 4.3, durante una iteración, el trabajo se desarrolla en la mayoría o todos los flujos de trabajo; sin embargo, el esfuerzo aplicado en ellos cambia a lo largo del tiempo. Por ejemplo, en las primeras iteraciones se tiende a aplicar un mayor esfuerzo a los requisitos y al diseño, y en las posteriores disminuye debido a que los requisitos y el diseño del núcleo central de la arquitectura son estabilizados mediante retroalimentación y adaptación.

Relacionando esto con las fases del UP, la figura 4.4 muestra el esfuerzo relativo de cambio con respecto a las fases; es importante tomar dicho grafico como una sugerencia, y no literalmente.

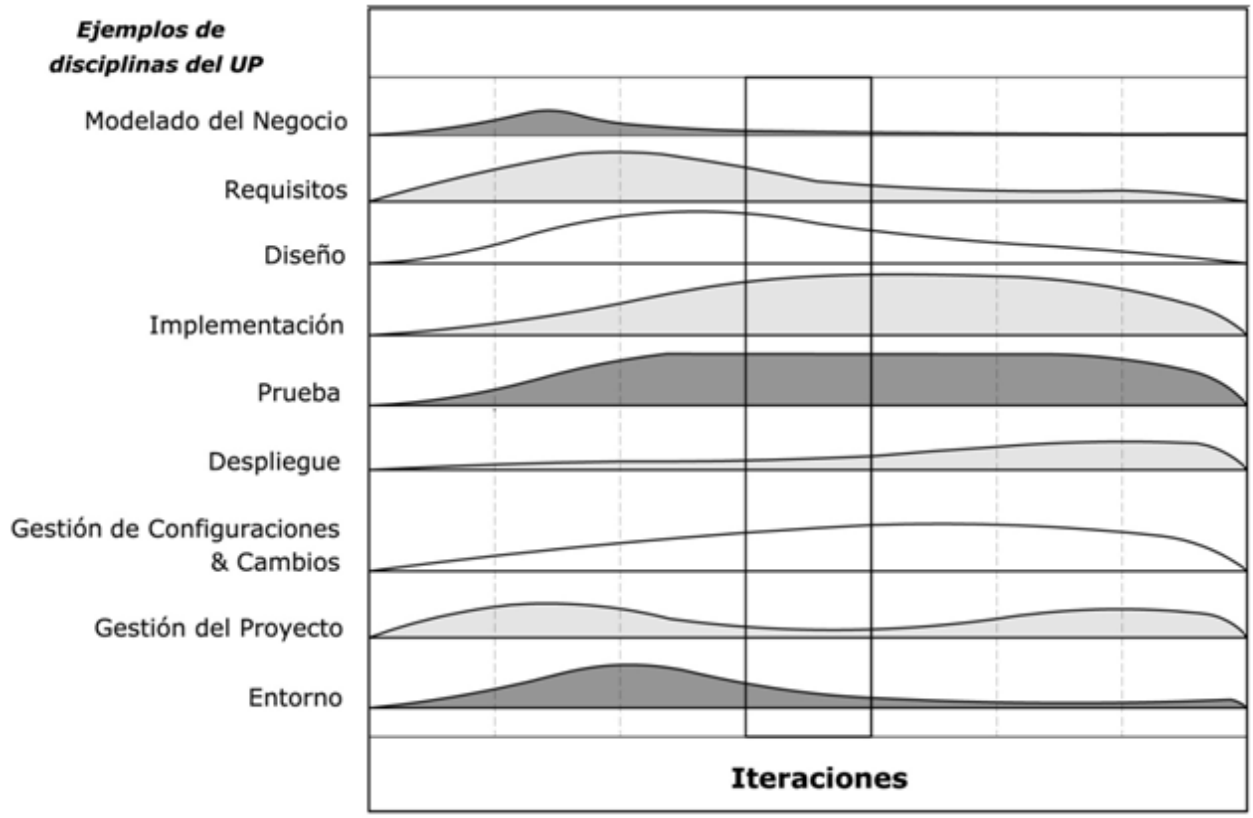



**Figura 4.3** *Flujos de trabajo del UP*
(Jacobson et al., 2003, p.11)

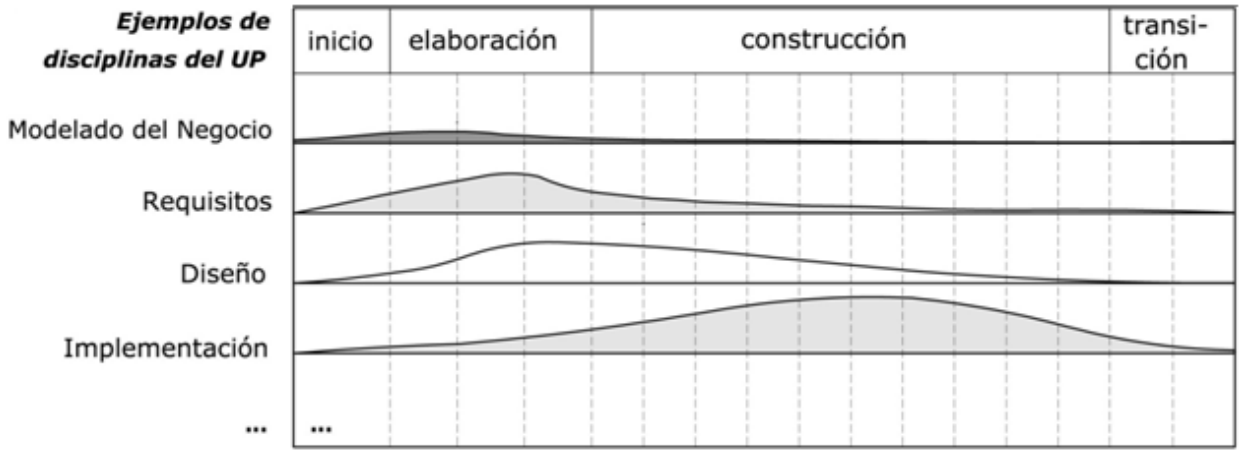



**Figura 4.4** *Flujos de trabajo y fases del UP*
(Jacobson et al., 2003, p.11)

## 4.4 Fase de inicio

La mayoría de los proyectos de software requieren una etapa inicial breve en la que se estudian los siguientes tipos de preguntas (Larman et al., 2003):

- ¿Cuál es la visión y el análisis del negocio para este proyecto?
- ¿Es viable?
- ¿Se debería desarrollar o es preferible no seguir?
- ¿Cuál es una aproximación de su costo?

Para poder definir la visión y obtener una estimación aproximada de la magnitud del proyecto es necesario llevar a cabo alguna exploración de los requisitos; sin embargo, el propósito de la fase de inicio, no es definir todos los requisitos, o generar una estimación creíble o plan de proyecto. Por lo que, aun a riesgo de simplificar demasiado, la idea es efectuar la investigación justa para formar una opinión racional y justificable del propósito global del nuevo sistema.

Por lo tanto, la fase de inicio debería ser relativamente corta en la mayoría de los proyectos, una o dos semanas; de hecho, en muchos proyectos, si la duración es superior a una semana se pierde la idea fundamental de la fase de inicio: "*decidir si es necesario realizar una investigación profunda durante la fase de elaboración"*, no haber llevado a cabo dicha investigación. Si se ha decidido que el proyecto se hará sin ninguna duda, y es claramente viable, la fase de inicio se caracteriza por ser especialmente breve; puede tener una duración de tan solo dos o tres días.

### 4.4.1 Objetivos de la fase de inicio

El objetivo principal de la fase de inicio es desarrollar el análisis del negocio hasta el punto que se permita justificar la puesta en marcha del proyecto (Larman et al., 2003).

Los objetivos secundarios son (Larman et al., 2003):

1. Establecer el ámbito del proyecto y sus límites.
2. Encontrar los casos de uso críticos del sistema.
3. Definir una arquitectura candidata para los escenarios principales.
4. Identificar los riesgos y las fuentes de incertidumbre.

### 4.4.2 Requisitos

En la fase de inicio, el mayor esfuerzo reside principalmente dentro de este flujo de trabajo. El objetivo de los requisitos es encontrar, comunicar y registrar lo que se necesita realmente, de manera que el significado sea totalmente claro para el cliente y el equipo de desarrollo (Larman et al., 2003).

La captura de requisitos se divide en dos grupos (Larman et al., 2003):

- **Requisitos funcionales**

  No todos los casos de uso se escriben en formato completo durante esta fase, tan solo entre el 10% y el 20% de los casos de uso que representan las funciones principales o que son especialmente arriesgadas en alguna dimensión, son escritos en formato completo; para ello el equipo debe efectuar una investigación cuidadosa y con cierta profundidad. El propósito es entender mejor la magnitud, complejidad y los riesgos ocultos del proyecto.

  Para el análisis de requisitos de cualquier aplicación, es necesario enfocarse en los casos de uso a nivel de procesos del negocio elementales (EBPs, Elementary Business Processes); ya que un error típico en la captura de requisitos funcionales es definir muchos casos de uso a un nivel muy bajo; es decir, como un paso simple, subfunción o subtarea dentro de un EBP.

- **Requisitos no funcionales**

  Si un requisito no funcional de calidad o restricción se relaciona de manera específica con un caso de uso, el UP recomienda registrar dichos requisitos con el caso de uso en cuestión. Sin embargo, muchos expertos prefieren reunirlos en un documento especial llamado especificación complementaria; ya que favorece la gestión del contenido, comprensión y legibilidad, al momento de efectuar el análisis de arquitectura en la etapa de elaboración.

### 4.4.3 Diseño

El objetivo principal del flujo de trabajo de diseño dentro de esta fase, es crear el modelo de la arquitectura candidata para soportar los casos de uso críticos del sistema; esto implica diseñar un boceto inicial de la arquitectura del sistema considerando los elementos más significativos, tales como: su organización, clases de análisis y su interacción para la realización de los casos de uso, entre otros, todo con un alto nivel de abstracción (Jacobson et al., 2003).

El modelo realizado debe llevar a cabo tanto los requisitos funcionales como los no funcionales.

### 4.4.4 Implementación

La actividad en este flujo de trabajo depende la decisión que debe tomar el jefe de proyecto: "*¿Debe finalizar la fase de inicio con la descripción de una arquitectura candidata?"* (Jacobson et al., 2003).

En situaciones normales, se finaliza con la descripción de la arquitectura candidata, en cuyo caso, el flujo de implementación no es necesario. Pero en algunos casos, es necesario verificar que la arquitectura candidata funciona, por lo que se requiere construir un prototipo de prueba (prototipo desechable); de ser así, el proyecto entra en un flujo de implementación pequeño.

### 4.4.5 Pruebas

Mientras se desarrollan las actividades de los demás flujos de trabajo, se debe empezar a considerar qué pruebas serán requeridas. Puede que se lleguen a desarrollar algunos planes provisionales de prueba; sin embargo, no se realiza un trabajo significativo de pruebas dentro de esta fase, ya que el prototipo creado, por lo general, es ilustrativo, más que operativo.

### 4.4.6 Artefactos

Es importante recordar que los artefactos deben ser considerados opcionales. Se deben elegir solo aquellos que añaden valor al proyecto y descartarlos si no se prueba que merecen la pena. La tabla 4.1 presenta una lista de los artefactos comunes de la fase de inicio:

**Tabla 4.1** *Artefactos creados en la fase de inicio*
(Larman et al., 2003)

| Artefactos | Comentario |
|---|---|
| Visión y Análisis del negocio | Describe los objetivos y restricciones de alto nivel |
| Modelo de casos de uso | Describe los requisitos funcionales |
| Especificación complementaria | Describe otros requisitos |
| Glosario | Terminología clave del dominio |
| Lista de riesgos | Describe los riesgos del negocio y del proyecto |
| Marco de desarrollo | Descripción de los pasos y artefactos del UP |

Estos artefactos se completan parcialmente en esta fase (se concluyen en futuras iteraciones).

## 4.5 Fase de elaboración

La elaboración es la serie inicial de iteraciones durante las que (Jacobson et al., 2003):

- Se descubren y estabilizan la mayoría de los requisitos.
- Se reducen o eliminan los riesgos importantes.
- Se implementan y prueban los elementos básicos de la arquitectura.

La elaboración se conforma por dos o cuatro iteraciones; el UP recomienda que cada iteración dure entre dos y seis semanas, a menos que el equipo de trabajo sea muy grande (Jacobson et al., 2003, p.11). Es importante mencionar, que durante la elaboración ya no se van a crear prototipos de prueba, por lo que todo el código y el diseño constituyen porciones del sistema final con calidad de producción.

Como se mencionó en el punto 4.3, la fase de elaboración no corresponde con una fase de diseño o una fase en la que se desarrollan completamente los modelos preparándolos para que sean implementados en la etapa de construcción; lo anterior, representa un ejemplo claro de superposición de las ideas del desarrollo en cascada sobre el desarrollo iterativo y el UP.

En la parte final de la fase de elaboración se posee la información necesaria para planificar la etapa de construcción; de igual forma, ya es posible realizar con certeza el análisis de negocio.

### 4.5.1 Objetivos de la fase de elaboración

El objetivo principal de la fase de elaboración es construir el núcleo central de la arquitectura para guiar el trabajo durante las fases de construcción y transición (Jacobson et al., 2003).

Los objetivos secundarios son (Jacobson et al., 2003):

1. Recopilar la mayor parte de los requisitos que aún quedan pendientes; formulando los requisitos funcionales como casos de uso.

2. Continuar la observación y control de riesgos críticos, e identificar los riesgos significativos para poder estimar su impacto en el proyecto.

3. Estimar la planificación y los recursos globales del proyecto.

### 4.5.2 Requisitos

En la fase de elaboración, el flujo de requisitos aún posee gran importancia para el proyecto. Sus principales funciones son: identificar, establecer la prioridad y estructurar los casos de uso (Jacobson et al., 2003).

- **Encontrar e identificar los casos de uso**

  Se debe identificar entre el 80% y el 90% de los casos de uso; no es necesario detallar toda esa cantidad, es posible describir solo aquellos que son arquitectónicamente significativos.

- **Determinar la prioridad de los casos de uso**

  Al principio, se debe trabajar en identificar los casos de uso; posteriormente, se genera una clasificación de aquellos que están relacionados con el núcleo de central de la arquitectura.

- **Estructurar el modelo de casos de uso**

  Se deben buscar similitudes, simplificaciones y oportunidades que permitan mejorar la estructura del modelo de casos de uso; además, es posible emplear mecanismos de extensión y generalización para lograr un modelo mejor estructurado y fácil de entender.

- **Desarrollar prototipos de las interfaces de usuario**

  Durante la elaboración solo se deben considerar las interfaces de usuario que poseen importancia desde un punto de vista de la arquitectura.

### 4.5.3 Diseño

Usualmente, en esta fase se llega a diseñar e implementar al menos el 10% de los casos de uso; este pequeño porcentaje constituye solo una fracción del total de casos de uso identificados. El diseño es llevado a cabo desde un punto de vista de la arquitectura; esto significa que todo el diseño que se realiza en la elaboración, se va a enfocar solamente en los casos de uso, clases y subsistemas que son arquitectónicamente significativos (Jacobson et al., 2003).

Cabe destacar, que los paquetes durante el análisis, y los subsistemas durante el diseño poseen gran importancia para definir las vistas de la arquitectura.

- **Diseño de la arquitectura**

  Es necesario identificar la arquitectura en capas, por lo que se considera la capa de software del sistema, la capa de subsistemas intermedios, y se seleccionan los productos que serán utilizados al final; es posible emplear sistemas heredados para reutilizar ciertas partes.

  A continuación, se debe trabajar en los niveles más altos de la arquitectura para poder identificar los subsistemas y las interfaces que deben incluirse dentro del modelo de diseño. Los paquetes de análisis de alto nivel se convierten en subsistemas en el modelo de diseño.

- **Diseñar un caso de uso**

  Cuando han sido identificados los casos de uso arquitectónicamente más significativos, se diseñan en términos de subsistemas de diseño, subsistemas de servicio y clases del diseño.

- **Diseñar una clase**

  Se deben diseñar las clases que tienen participación dentro de los casos de uso descritos en el paso anterior; por lo general, se completan en el desarrollo de posteriores iteraciones.

- **Diseño un subsistema**

  Se diseñan los subsistemas resultantes del diseño de la arquitectura; durante esta fase, la vista de la arquitectura del modelo de diseño se debe actualizar solo si llega a ser necesario.

### 4.5.4 Implementación

Este flujo de trabajo adquiere importancia debido a que es en él, y precisamente durante esta fase, donde se implementan y prueban los componentes del núcleo central de la arquitectura (línea base de la arquitectura), que es generada a partir de menos del 10% de los casos de uso (Jacobson et al., 2003).

- **Implementación de la arquitectura**

  Basándose en la vista de la arquitectura del modelo de diseño y la vista de la arquitectura del modelo de despliegue, se identifican los componentes necesarios para implementar los subsistemas de servicio; los componentes ejecutables son asignados a los nodos de la red.

- **Implementación de clases y subsistemas**

  Esta actividad genera una versión ejecutable preliminar del sistema que se va a construir. El equipo de trabajo debe implementar las clases que son relevantes para la creación del núcleo central de la arquitectura. Las clases se implementan en términos de componentes.

- **Integrar el sistema**

  Tomando como base el pequeño porcentaje de casos de uso que van a ser implementados, se debe establecer un plan de integración para definir la secuencia de integración de los subsistemas y los componentes que forman parte del núcleo de la arquitectura ejecutable.

### 4.5.5 Pruebas

El objetivo de este flujo de trabajo en la fase de elaboracion es verificar que los subsistemas de todos los niveles y de todas las capas funcionan de forma óptima; aunque solo se pueden probar los componentes ejecutables, si estos funcionan, se obtiene cierta seguridad de que otras cosas (en otros modelos) también van a funcionar correctamente (Jacobson et al., 2003).

- **Planificar pruebas**

  El equipo debe seleccionar los objetivos que evaluarán el núcleo central de la arquitectura.

- **Diseñar las pruebas**

  Basándose en los objetivos, se identificarán los casos de prueba necesarios y se prepararán los procedimientos de prueba para comprobar la correcta integración de los subsistemas.

- **Realizar pruebas de integración**

  En cada construcción que se realice se debe comprobar la integración de cada componente.

- **Realizar pruebas al sistema**

  El núcleo central de la arquitectura debe ser sometido a diversas pruebas para verificar que cumple con los objetivos; en caso contrario, se notifican los defectos para su corrección.

### 4.5.6 Artefactos

La tabla 4.2 presenta una lista de los artefactos comunes de la fase de elaboración.

**Tabla 4.2** *Artefactos creados en la fase de elaboración*
(Larman et al., 2003)

| Artefactos | Comentario |
|---|---|
| Modelo del dominio | Es una visualización de los conceptos del dominio |
| Modelo de diseño | Son los diagramas que describen el diseño lógico |
| Modelo de análisis | Es una especificación más precisa de los requisitos |
| Modelo de pruebas | Es una descripción de lo que se probará y cómo. |
| Modelo de la implementación | Implementación real (código fuente, ejecutables) |
| Prototipos UI | Es una descripción de la interfaz de usuario |
| Descripción de la arquitectura | Es un resumen del diseño de la arquitectura |

Al final de la fase de elaboración, estos modelos se encontrarán completos al menos en un 10% (exceptuando los modelos de casos de uso y análisis).

## 4.6 Fase de construcción

La construcción va más allá del núcleo central de la arquitectura (Jacobson et al., 2003):

- Se identifica y detalla la totalidad de los casos de uso.
- Se termina de construir y de integrar el sistema para obtener su versión operativa inicial.
- Se realizan pruebas y se empieza a prepara el despliegue operacional del sistema.

Si bien la fase de inicio y la fase elaboración podrían ser consideradas como investigación, la fase de construcción corresponde al desarrollo, por lo que, el trabajo se traslada a la creación del sistema o aplicación dentro de los parámetros establecidos para el costo, esfuerzo y tiempo.

En esta fase, la construcción parte del núcleo central de la arquitectura, a través de una serie de iteraciones e incrementos para desarrollar un producto de software listo para su operación inicial en el entorno de usuario; dicho producto, posee la calidad adecuada para su aplicación y debe asegurarse de cumplir los requisitos que se han establecido (Jacobson et al., 2003).

### 4.6.1 Objetivos de la fase de construcción

El objetivo principal de la fase de construcción es crear la versión operativa inicial del sistema; también se le conoce como versión beta (Jacobson et al., 2003).

Los objetivos secundarios son (Jacobson et al., 2003):

1. Finalizar la disminución de riesgos (exceptuando aquellos que se derivan de la operación).
2. Realizar pruebas de integración y del sistema (alpha, beta).
3. Preparar el despliegue mediante actividades tales como la escritura de la guías de usuario.

### 4.6.2 Requisitos

En la fase de construcción la mayoría de los requisitos funcionales y no funcionales ya han sido estabilizados de manera iterativa y adaptable; esto no significa que los requisitos se congelan o el estudio termina, sino que el grado de cambio es mucho menor (Jacobson et al., 2003).

- **Encontrar e identificar los casos de uso**

  Casi siempre, solo resta identificar un pequeño porcentaje de casos de uso (menos del 20%). Si es necesario, se deben actualizar los casos de uso y actores en el modelo de casos de uso.

- **Determinar la prioridad de los casos de uso**

  En la fase de elaboración, se creó una clasificación de los casos de usos relacionados con la arquitectura; en esta fase, se añaden los casos de uso restantes para establecer su prioridad.

- **Estructurar el modelo de casos de uso**

  Es posible mejorar el modelo de casos de uso; pero el sistema ya posee una arquitectura estable, por lo que, cualquier cambio debe referirse a casos de uso que aún se desarrollan.

- **Desarrollar prototipos de las interfaces de usuario**

  Por regla, se construye un prototipo que debe ser probado por los usuarios; considerando las respuestas dadas, el prototipo se modifica para cumplir las necesidades de los usuarios.

### 4.6.3 Diseño

Durante la fase de construcción se diseña e implementa el 90% restante de los casos de uso. Constituyen los casos de uso no utilizados para desarrollar el núcleo central de la arquitectura (Jacobson et al., 2003).

- **Diseño de la arquitectura**

  No es necesario añadir subsistemas de diseño ni subsistemas de servicio, ya que estos existen en el núcleo central de la arquitectura en forma de esqueleto; solo se deben añadir subsistemas si son similares o alternativos a los que se ya se encuentran en la arquitectura.

  El comportamiento de un subsistema de servicio puede derivarse de partes de un caso de uso o de un conjunto de casos de uso relacionados; casi siempre, entre el 40% y el 60% de responsabilidades de un subsistema de servicio provienen de un caso de uso con este estilo.

  Si el comportamiento de un subsistema posee un alto porcentaje de responsabilidades a partir de un caso de uso o de un conjunto de casos de usos relacionados (más del 80%), el UP recomienda autorizar el desarrollo completo del servicio en la misma construcción (Jacobson et al., 2003).

### 4.6.4 Implementación

En este flujo de trabajo se lleva a cabo la mayor parte del trabajo de la fase de construcción. Aquí se implementa y se realizan las pruebas de unidad de todos los componentes trabajando principalmente a partir del modelo de diseño; el resultado, después de varias iteraciones, y de la integración y pruebas del sistema, es la versión operativa inicial (Jacobson et al., 2003).

- **Implementar la arquitectura**

  La arquitectura se encuentra estable, se debe actualizar solo en caso de que sea necesario.

- **Implementar clases y subsistemas**

  Se implementan las clases y subsistemas en su totalidad de acuerdo al orden establecido.

- **Realizar pruebas de unidad**

  Se realizan pruebas de unidad; de ser necesario, se corregirá el diseño y la implementación.

- **Integrar el sistema**

  Se debe crear un plan de integración que permita generar una secuencia de construcciones; este plan mostrará los casos de uso y escenarios que van a ser implementados en esta fase.

### 4.6.5 Pruebas

Este flujo es fundamental en esta fase, ya que es responsable de realizar pruebas de integración y de sistema, con el objeto de comprobar cada construcción realizada dentro de las iteraciones, y finalmente, en la última instancia de la construcción final (Jacobson et al., 2003).

- **Planificar las pruebas**

  El equipo de trabajo debe seleccionar los objetivos que van a permitir comprobar las construcciones de cada iteración y de la versión final del sistema.

- **Diseñar las pruebas**

  El equipo de trabajo establece como probar los requisitos en el conjunto de construcciones; por lo que, se prepararán casos y procedimientos de prueba para satisfacer este propósito.

- **Realizar pruebas de integración**

  Para cada construcción que se integra se generan pruebas; si se superan las pruebas, se añaden construcciones adicionales, en caso contrario, se debe registrar y notificar el fallo.

- **Realizar pruebas del sistema**

  En cada iteración se obtiene una versión parcial del sistema que debe ser probado con los procedimientos de prueba del sistema. Al final, se comprobará la versión operativa inicial.

- **Evaluar las pruebas**

  Conforme se realizan las pruebas, se deben evaluar para revisar los resultados obtenidos; el propósito de evaluar las pruebas es verificar que cumplen con los objetivos establecidos.

### 4.6.6 Artefactos

La tabla 4.3 presenta una lista de los artefactos comunes de la fase de construcción.

**Tabla 4.3** *Artefactos creados en la fase de diseño*
(Larman et al., 2003)

| Artefactos | Comentario |
|---|---|
| Sistema software ejecutable | Versión con capacidad operativa inicial |
| Manual de usuario | Versión preliminar, sirve como guía a los usuarios |
| Descripción de la arquitectura | Mínimamente modificada y actualizada |
| Análisis del negocio | Refleja la situación al final de la fase |
| Plan de transición | Describe que hacer durante la fase de transición |

Al final de la fase de construcción, todos los artefactos del sistema, incluyendo los modelos (casos de uso, análisis, diseño, implementación, despliegue) deben encontrarse completos.

## 4.7 Fase de transición

La transición es la serie final de iteraciones durante las que (Jacobson et al., 2003):

- Se prepara la versión beta (o de pruebas de aceptación) producida en la fase construcción; con el objeto de llevar a cabo la instalación del sistema en el entorno operativo del usuario.
- Se procede de acuerdo a la información que se ha obtenido en las pruebas.
- Se completan los artefactos que conforman el proyecto (código, diagramas, modelos, etc.).
- Se realiza una evaluación que determina cuándo debe finalizar el proyecto.

Cuando se inicia la fase de transición el sistema ya ha alcanzado la capacidad operativa inicial; sin embargo, en la mayoría de las ocasiones, aun existen riesgos, problemas y defectos que no se han puesto en evidencia, por lo que, es muy posible que estos se manifiesten al momento de realizar las pruebas del sistema dentro del entorno operativo del usuario.

En otras ocasiones, el usuario puede descubrir la necesidad de determinadas características, si son muy importantes y no afectan el sistema, se pueden añadir. Cualquier cambio considerable, se debe efectuar en un nuevo ciclo de desarrollo, es decir, para la siguiente versión del sistema (Jacobson et al., 2003). De acuerdo a lo anterior, en esta fase no se busca reformular el producto,

tanto el cliente como el equipo de trabajo ya han incorporado los cambios significativos en los requisitos a lo largo de las fases anteriores; por el contrario, el equipo de desarrollo solamente identifica pequeñas deficiencias que han pasado desapercibidas durante la fase de construcción y que se pueden corregir sin llegar a modificar el núcleo central de la arquitectura del sistema.

En ciertas ocasiones el equipo puede proporcionar ayuda para crear un entorno apropiado para el sistema. De igual forma, puede participar en la formación de usuarios que utilizan el sistema.

### 4.7.1 Objetivos de la fase de transición

El objetivo principal de la fase de transición es implantar el sistema en el entorno de operación (Jacobson et al., 2003).

Los objetivos secundarios son (Jacobson et al., 2003):

1. Cumplir los requisitos establecidos previamente hasta lograr la satisfacción de los usuarios.
2. Gestionar todos los aspectos relativos a la operación del sistema en el entorno del usuario; esto incluye la corrección de los errores experimentados por los usuarios de la versión beta o por los encargados de las pruebas de aceptación.

### 4.7.2 Flujos de trabajo en la fase de transición

En esta fase, la actividad que se realiza en los flujos de trabajo es baja (observe la Figura 4.4). Dentro del UP casi todo el trabajo se ha llevado a cabo en la fase de construcción, por lo que, en la fase de transición el esfuerzo se centra en corregir los problemas encontrados durante las pruebas del entorno del usuario (Jacobson et al., 2003).

La mayor parte del trabajo se enfoca en actividades que no recaen dentro los flujos de trabajo. En los siguientes puntos se menciona básicamente en qué consisten dichas actividades del UP (Jacobson et al., 2003):

- **Preparación de la versión beta**

  El equipo selecciona el conjunto inicial de usuarios encargados de realizar las pruebas beta. Además, recoge la documentación creada con anterioridad. Esta debe complementarse con instrucciones específicas sobre como notificar los resultados de las pruebas y observaciones.

- **Instalación de la versión beta**

  Durante las pruebas beta el equipo no estará presente, por lo que, debe proporcionarse la documentación que se ha recogido, las instrucciones de cómo operar e instalar el sistema, y en qué aspectos y problemas deben centrarse los usuarios para llevar a cabo las pruebas.

- **Reacción ante los resultados de las pruebas**

  El equipo recopila y analiza los resultados de las pruebas con el objeto de realizar acciones. Por lo general, los resultados se clasifican en: fallos de codificación relativamente menores y problemas importantes que pueden presentar ramificaciones extensas dentro del sistema.

- **Adaptación del producto al entorno del usuario**

  Una vez que se que concluyen las pruebas beta se debe preparar el entorno de operación. Es posible que se requieran actividades adicionales, por ejemplo: migraciones de datos, conversiones de base de datos del sistema viejo al nuevo, configuraciones particulares, etc.

### 4.7.3 Artefactos

La tabla 4.4 presenta una lista de los artefactos comunes de la fase de transición.

**Tabla 4.4** *Artefactos creados en la fase de transición*
(Larman et al., 2003)

| Artefactos | Comentario |
|---|---|
| Sistema software ejecutable | Versión final, se incluye software de instalación |
| Modelos del sistema | Versión completa y corregida de todos los modelos |
| Manuales, material de formación | Muestra al usuario la forma de utilizar el sistema |
| Referencias de ayuda | Como informar de defectos, actualizaciones, etc. |

Al final de la fase de transición se han completado todos artefactos, incluyendo los modelos y la descripción de la arquitectura; de igual forma, se verifica que sean consistentes unos con otros. Cabe mencionar que el conjunto de modelos debería estar completo al final de la construcción.

### 4.7.4 Finalización del proyecto

La fase de transición no termina cuando se han completado todos los artefactos; sino cuando el cliente queda satisfecho. Lo que origina la pregunta ¿cuándo están los usuarios satisfechos? Como lo menciona Jacobson, Booch y Rumbaugh (2003): "*Determinar este momento depende de la relación mercantil en la cual se realiza el proyecto*".

Si el producto de software se va a lanzar al mercado, los usuarios quedan satisfechos cuando el proyecto reacciona de forma positiva ante los resultados que se obtienen en las pruebas beta; por lo que, la fase de transición finaliza cuando el mantenimiento pasa a otra organización. En el caso de los productos que son contratados por un cliente en específico, el cliente queda satisfecho cuando el sistema pasa las pruebas de aceptación; esto depende de la interpretación de los requisitos detallados en el contrato original, pero la transición como tal habrá acabado.

La implementación del algoritmo que soluciona el problema de equilibrio del usuario se realizó con éxito siguiendo las normas que provee el *"Proceso Unificado de Desarrollo de Software"*. Con ello se logró estabilizar, controlar y organizar las diferentes tareas durante la construcción. En el siguiente capítulo se proveen los artefactos fundamentales utilizados en la presente tesis. Cabe mencionar que se eligieron solo aquellos que añaden valor a la implementación realizada.

# Capítulo 5

# Artefactos del Sistema

Un sistema de software no solo se compone por el código máquina y los archivos ejecutables. Un sistema de software es el conjunto de artefactos necesarios para que su representación sea comprensible para las máquinas, los trabajadores y todos aquellos interesados en el proyecto. Un artefacto es un término general utilizado para nombrar cualquier tipo de información que es creada, producida, cambiada o utilizada por los trabajadores durante el desarrollo del sistema.

En este capítulo se muestran los artefactos de ingeniería utilizados por el Proceso Unificado durante la implementación del algoritmo que resuelve el problema de equilibrio del usuario.

## 5.1 Flujo de trabajo de los requisitos

Los artefactos fundamentales utilizados por el flujo de trabajo de los requisitos se describen a continuación (Jacobson et al., 2003).

1. **Modelo de dominio**

   El modelo de dominio es la representación de los conceptos (objetos) significativos en el contexto del problema. Es construido utilizando las reglas que proporciona UML e incluye: clases de objetos, asociaciones entre clases de objetos y atributos de las clases de objetos.

2. **Modelo de casos de uso**

   El modelo de casos de uso es la combinación de casos de uso y sus correspondientes diagramas a través de los cuales se describe la funcionalidad propuesta de la nueva aplicación.

### 5.1.1 Modelo de dominio

A continuación se proporciona el modelo del dominio para la implementación del algoritmo que resuelve el problema de equilibrio del usuario (PEU). Como se observa en la figura 5.1 el objeto principal es el algoritmo solución PEU, el cual basa su funcionamiento en el objeto Frank Wolfe. A su vez, Frank Wolfe se descompone en cuatro objetos: el método de asignación de tráfico Todo o Nada, el algoritmo de ruta mínima Dijkstra, el algoritmo de Bisección y las funciones de tiempo de viaje implementadas. El objeto Red Vial lleva a cabo el modelado de la red de tráfico vehicular sobre la cual el algoritmo solución PEU realiza la asignación de flujo a los arcos que la conforman.

Es importante señalar que en este punto del desarrollo, los objetos no constituyen clases de software (aunque algunos objetos del Modelo de Dominio pueden terminar siéndolo).

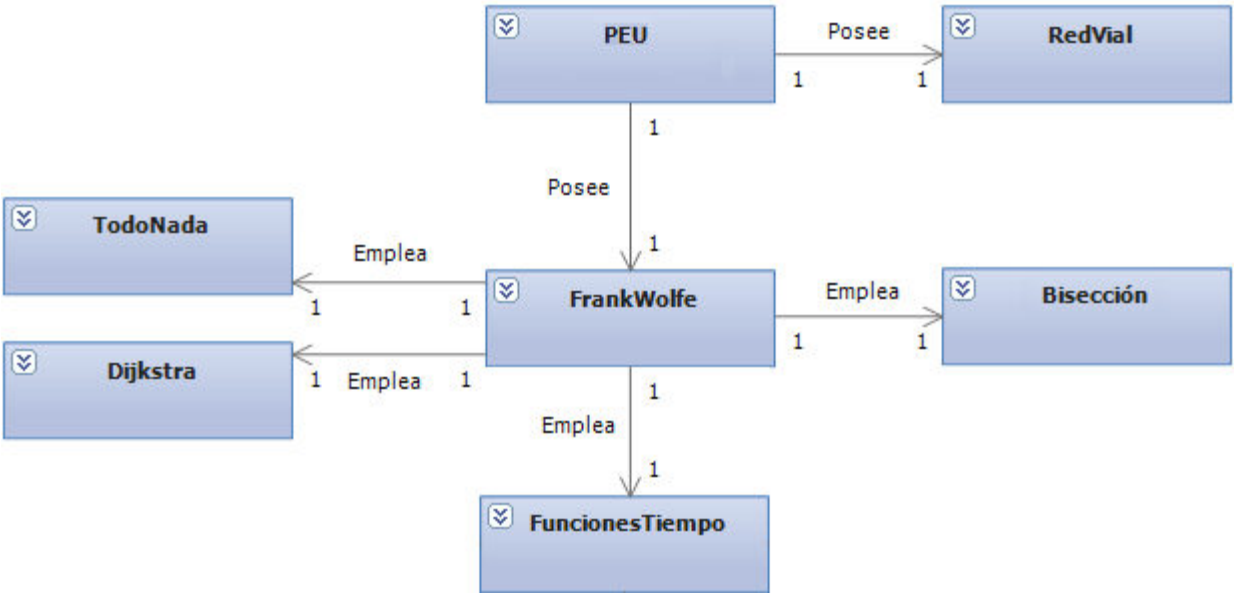



**Figura 5.1** *Modelo del dominio del algoritmo que resuelve el PEU*
(Elaboración propia)

### 5.1.2 Modelo de casos de uso

Como se ha mencionado en el punto 5.1, el modelo de casos de uso es la combinación de casos de uso y los correspondientes diagramas que describen la funcionalidad del sistema en desarrollo. Es importante recordar que el modelo de casos de uso es una pieza clave en el Proceso Unificado ya que es el encargado de guiar el proceso de implementación a través de las diferentes fases.

**1. Diagrama de casos de uso**

El diagrama de casos de uso utiliza los elementos de UML para especificar la comunicación y el comportamiento del sistema mediante su interacción con los usuarios y/u otros sistemas. En la figura 5.2 se ilustra el diagrama de casos de uso para el algoritmo que soluciona el PEU. Como se observa, existen cuatro casos de uso principales: introducir matriz Origen-Destino (O-D) de los viajes, seleccionar funciones, estimar flujo vehicular y visualizar resultado. Por su parte, el caso de uso estimar flujo vehicular incluye los casos de uso algoritmo Frank Wolfe y el modelado de la red vial. De igual forma se aprecia que el actor que inicia la interacción con el sistema es el usuario.

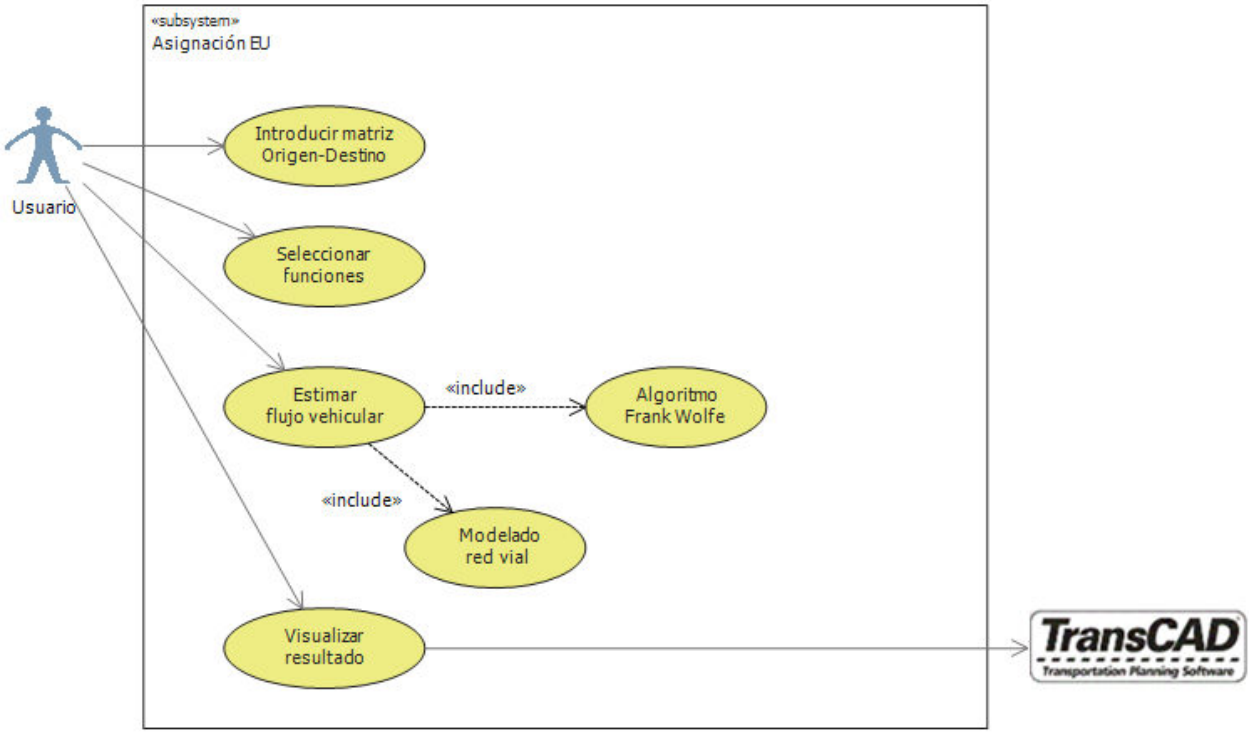



**Figura 5.2** *Diagrama de casos de uso del algoritmo que resuelve el PEU*
(Elaboración propia)

## 2. Casos de uso

Un caso de uso es una descripción de los pasos necesarios para llevar a cabo algún proceso. Considerando los casos de uso que proporciona el diagrama de casos de uso de la figura 5.2, la serie de figuras 5.3 – 5.8 detalla la secuencia necesaria para efectuar cada uno de ellos.

| **Caso de Uso:** | 1. Introducir viajes de la matriz Origen-Destino. |
|---|---|
| **Actor** | Usuario |
| **Descripción:** | El usuario proporciona a la aplicación el número de viajes de vehículos que serán cargados en la red vial. |
| **Precondiciones:** | 1. La aplicación se ha cargado correctamente. |

**Flujo normal de eventos**

| **Actor** | | **Sistema** | | **Excepción** |
|---|---|---|---|---|
| **Paso** | **Acción** | **Paso** | **Acción** | |
| 1 | El usuario ingresa en la aplicación el número de viajes de vehículos. | | | |
| | | 2 | La aplicación verifica la validez del valor que ha ingresado el usuario. | E-1, E-2 |
| | | 3 | La aplicación guarda temporalmente el número de viajes de vehículos. | |

**Flujo excepcional de eventos**

| **Excepción** | **Descripción** | **Acción** |
|---|---|---|
| E-1 | No se ingresa ningún valor. | Se notifica al usuario. |
| E-2 | No se ingresa un valor entero > 0. | Se notifica al usuario. |
| **Postcondiciones:** | Se obtiene el número de viajes de vehículos que serán cargados en la red vial. | |

**Figura 5.3** *Caso de uso introducción de la matriz O-D* (Elaboración propia)

| **Caso de Uso:** | 2. Seleccionar funciones. |
|---|---|
| **Actor** | Usuario |
| **Descripción:** | El usuario elige las funciones de tiempo de viaje que emplea el algoritmo Frank Wolfe para trabajar con la red vial. |
| **Precondiciones:** | 1. La aplicación se ha cargado correctamente. |

| **Flujo normal de eventos** | | | | |
|---|---|---|---|---|
| **Actor** | | **Sistema** | | **Excepción** |
| **Paso** | **Acción** | **Paso** | **Acción** | |
| | | 1 | La aplicación despliega el nombre de las funciones existentes. | |
| 2 | El usuario selecciona una opción. | | | E-3 |
| | | 3 | La aplicación guarda temporalmente las funciones elegidas. | |

| **Flujo excepcional de eventos** | | |
|---|---|---|
| **Excepción** | **Descripción** | **Acción** |
| E-3 | No se selecciona ninguna opción. | Se notifica al usuario. |

| **Postcondiciones:** | Se obtienen las funciones que serán utilizadas por el algoritmo Frank Wolfe. |
|---|---|

**Figura 5.4** *Caso de uso seleccionar funciones*
(Elaboración propia)

| **Caso de Uso:** | 3. Estimar flujo vehicular. |
|---|---|
| **Actor** | Usuario |
| **Descripción:** | La aplicación distribuye el flujo vehicular sobre la red vial utilizando el algoritmo Frank Wolfe. |
| **Precondiciones:** | 1. La demanda vehicular ha sido especificada.<br>2. Las funciones de tiempo de viaje han sido especificadas. |

**Flujo normal de eventos**

| **Actor** | | **Sistema** | | **Excepción** |
|---|---|---|---|---|
| **Paso** | **Acción** | **Paso** | **Acción** | |
| 1 | El usuario elige estimar flujo vehicular. | | | |
| | | 2 | La aplicación obtiene los datos que el usuario ingresó con anterioridad. | |
| | | 3 | La aplicación crea la red vial. | E-5 |
| | | 4 | La aplicación ejecuta el algoritmo Frank Wolfe. | |

**Flujo excepcional de eventos**

| **Excepción** | **Descripción** | **Acción** |
|---|---|---|
| E-5 | No existen datos sobre la red vial. | Se notifica el error al usuario. |

| **Postcondiciones:** | Se obtiene el patrón estimado de flujo vehicular. |
|---|---|

**Figura 5.5** *Caso de uso estimar flujo (asignar tráfico) vehicular*
(Elaboración propia)

| **Caso de Uso:** | 4. Modelado red vial. |
|---|---|
| **Actor** | Aplicación |
| **Descripción:** | La aplicación crea la red vial. |
| **Precondiciones:** | 1. El usuario ha elegido la opción estimar flujo vehicular. |

**Flujo normal de eventos**

| **Actor** | | **Sistema** | | **Excepción** |
|---|---|---|---|---|
| **Paso** | **Acción** | **Paso** | **Acción** | |
| | | 1 | La aplicación obtiene los datos que corresponden a la red vial. | E-6 |
| | | 2 | La aplicación construye la red vial. | |

**Flujo excepcional de eventos**

| **Excepción** | **Descripción** | **Acción** |
|---|---|---|
| E-6 | No existen datos sobre la red vial. | Se notifica el error al usuario. |

| **Postcondiciones:** | Se obtiene la red vial sobre la cual se estima el flujo vehicular. |
|---|---|

**Figura 5.6** *Caso de uso modelado red vial*
(Elaboración propia)

| **Caso de Uso:** | 5. Algoritmo Frank Wolfe. |
|---|---|
| **Actor** | Aplicación |
| **Descripción:** | La aplicación PEU ejecuta el Algoritmo Frank Wolfe. |
| **Precondiciones:** | 2. La red vial se ha construido correctamente. |

**Flujo normal de eventos**

| **Actor** | | **Sistema** | | **Excepción** |
|---|---|---|---|---|
| **Paso** | **Acción** | **Paso** | **Acción** | |
| | | 1 | La aplicación proporciona la red vial al algoritmo Frank Wolfe. | |
| | | 2 | La aplicación se encarga de ejecutar el algoritmo Frank Wolfe. | |

**Flujo excepcional de eventos**

| **Excepción** | **Descripción** | **Acción** |
|---|---|---|
| Ninguna | - | - |

| **Postcondiciones:** | El flujo vehicular es estimado sobre la red vial. |
|---|---|

**Figura 5.7** *Caso de uso algoritmo Frank Wolfe*
(Elaboración propia)

| **Caso de Uso:** | 6. Visualizar resultado. |
|---|---|
| **Actor** | TransCAD |
| **Descripción:** | La aplicación se conecta con el SIG TransCAD para enviar el patrón de flujo. TransCAD genera su representación geográfica. |
| **Precondiciones:** | 1. Patrón de flujo vehicular estimado. |

**Flujo normal de eventos**

| **Actor** | | **Sistema** | | **Excepción** |
|---|---|---|---|---|
| **Paso** | **Acción** | **Paso** | **Acción** | |
| 1 | El usuario selecciona la opción para conectarse con TransCAD. | | | |
| | | 2 | La aplicación se encarga de enviar la petición de conexión con TransCAD. | |
| 3 | TransCAD recibe la petición y realiza la conexión con la aplicación. | | | E-6 |
| 4 | El usuario selecciona la opción para visualizar el resultado en TransCAD. | | | |
| | | 5 | La aplicación envía el patrón de flujo vehicular estimado a TransCAD. | |
| 6 | TransCAD representa gráficamente el patrón de flujo vehicular. | | | |

**Flujo excepcional de eventos**

| **Excepción** | **Descripción** | **Acción** |
|---|---|---|
| E-7 | No es posible llevar a cabo la conexión. | Se notifica al usuario. |
| **Postcondiciones:** | Se genera la representación geográfica del patrón de flujo estimado. | |

**Figura 5.8** *Caso de uso visualizar resultado* (Elaboración propia)

## 5.2 Flujo de trabajo de diseño

En el diseño se ha modelado el sistema y se ha encontrado su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales y restricciones. Los artefactos fundamentales utilizados por el flujo de diseño son descritos a continuación (Jacobson et al., 2003).

### 5.2.1 Modelo de diseño

El modelo de diseño consiste en un plano de la implementación que describe cómo se efectúan los casos de uso. Para ello se basa en los siguientes elementos:

- **Diagrama de clases**

  El diagrama de clases es un diagrama estático que muestra la estructura del sistema en base a sus clases, atributos y las relaciones entre ellas.

- **Diagrama de secuencia**

  El diagrama de secuencia es un diagrama que muestra una secuencia detallada de la interacción entre los objetos de diseño en la realización de un caso de uso en específico.

El modelo de diseño del algoritmo que resuelve el PEU representado a través del diagrama de clases y los diagramas de secuencia de las realizaciones de casos, se encuentra almacenado en formato digital PDF y XPS en el CD que se anexa en la parte final de esta tesis. Esto debido al tamaño que poseen las imágenes.

### 5.2.2 Descripción de la arquitectura (vista modelo de diseño)

La descripción de la arquitectura contiene una vista de la arquitectura del modelo de diseño. En esta tesis se utiliza un diagrama de capas para visualizar el modelo de diseño del sistema. Como se ilustra en la figura 5.9, la implementación posee una arquitectura de tres capas:

1. **Capa de presentación**

   La capa de presentación es la interfaz gráfica de usuario (GUI) conformada por un conjunto de ventanas o cuadros de diálogos (Windows forms) que capturan los datos que se ingresan. Además, permite visualizar el resultado dentro de la aplicación o exportarlo al SIG TransCAD.

**2. Capa de lógica de negocio**

La capa de lógica de negocio se encarga de implementar el algoritmo que resuelve el PEU. Es aquí donde reside la mayor parte del trabajo efectuado, ya que es donde se ejecutan los algoritmos y las funciones de tiempo desarrolladas por Londoño y Lozano (2012).

**3. Capa de datos**

La capa de datos accede a los datos de la red vial que están almacenados en archivos CSV. Se encuentra formada por un gestor de archivos CSV responsable de efectuar dicha tarea. Cabe mencionar que la capa de datos se conecta solamente con la capa de lógica de negocio.

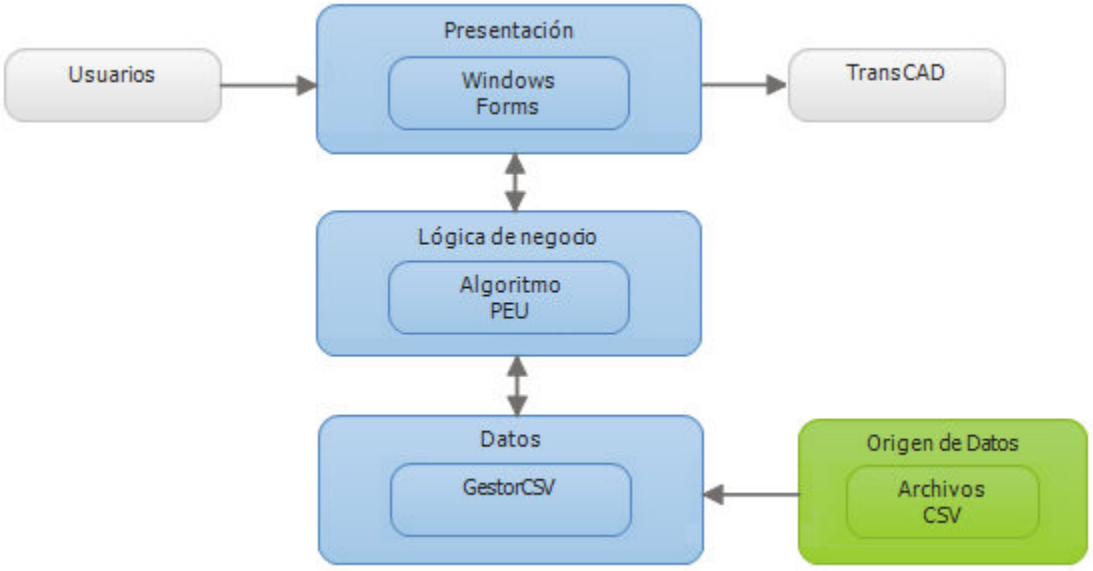



**Figura 5.9** *Arquitectura de tres capas de la implementación del algoritmo que resuelve el PEU* (Elaboración propia)

## 5.2.3 Modelo de despliegue

El modelo de despliegue es un modelo de objetos que permite describir la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. En la figura 5.10 se provee el modelo de despliegue del algoritmo que resuelve el PEU.

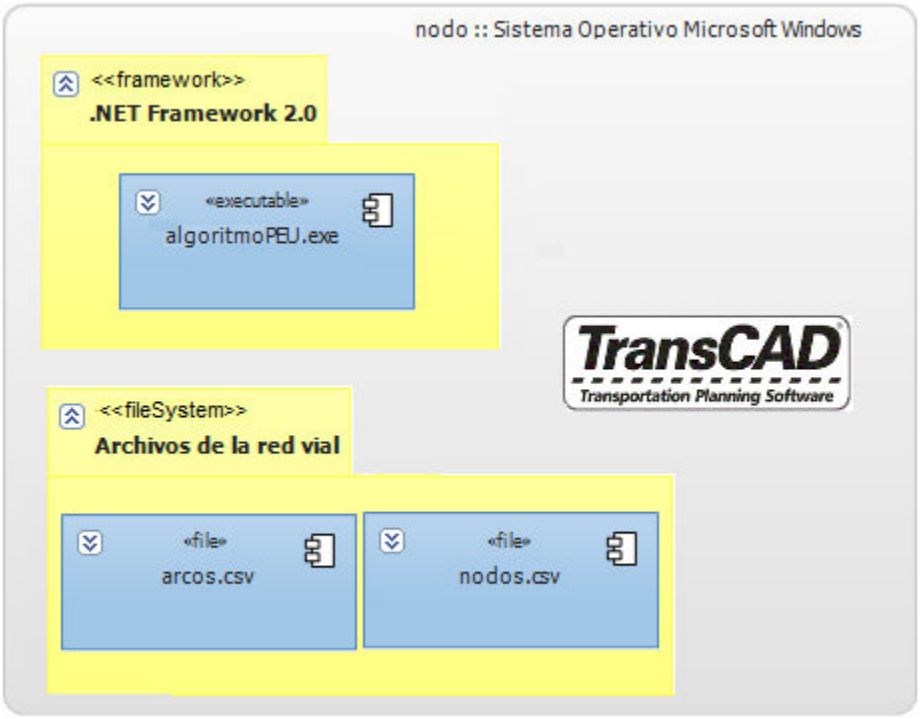



**Figura 5.10** *Modelo de despliegue de la implementación del algoritmo que resuelve el PEU*
(Elaboración propia)

### 5.2.4 Prototipos GUI

Los prototipos GUI son los objetos gráficos que permiten al usuario comunicarse con el sistema. La interfaz gráfica de la aplicación se muestra en la serie de figuras 5.5 – 5.8.

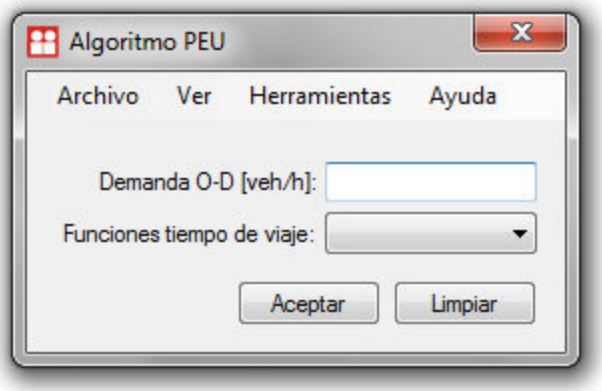



**Figura 5.11** *Ventana principal de la aplicación*
(Elaboración propia)

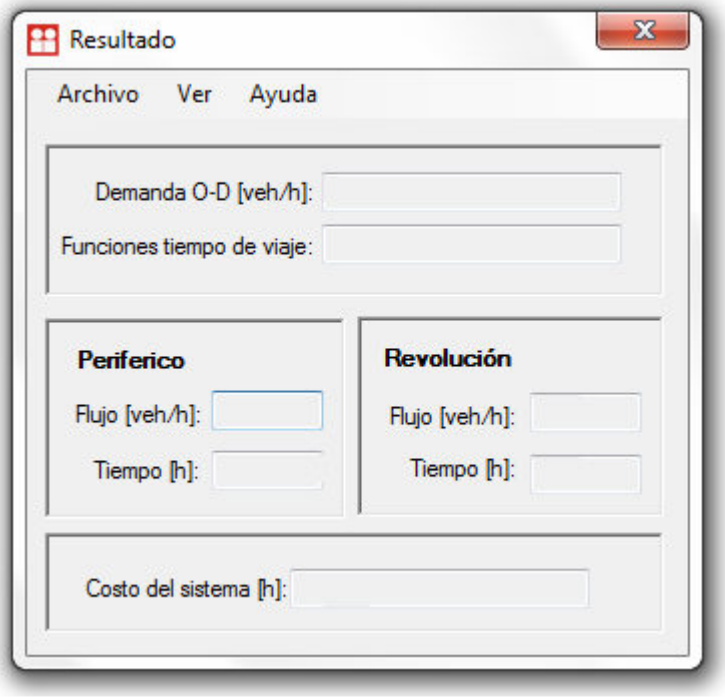



**Figura 5.12** *Ventana que muestra el resultado de la aplicación*
(Elaboración propia)

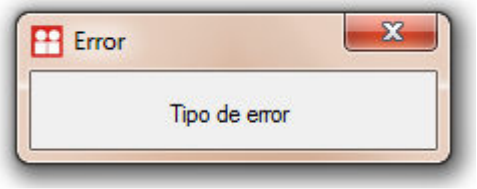



**Figura 5.13** *Ventana que muestra errores en la aplicación*
(Elaboración propia)

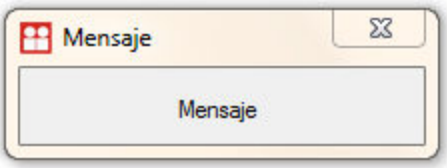



**Figura 5.14** *Ventana que muestra el mensaje de la aplicación*
(Elaboración propia)

## 5.3 Flujo de trabajo de implementación

En la implementación, el diseño llevado a cabo es implementado en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares. El modelo de implementación constituye el artefacto fundamental durante este flujo de trabajo (Jacobson et al., 2003).

### 5.3.1 Modelo de implementación

El modelo de implementación muestra cómo los elementos del modelo de diseño, como las clases, son implementadas en términos de componentes, como ficheros de código fuente, ejecutables, etc. Además, ilustra la organización de los componentes y los lenguajes de programación que han sido utilizados.

En la figura 5.9 se muestra el modelo de implementación del algoritmo que resuelve el PEU haciendo uso del diagrama de componentes.

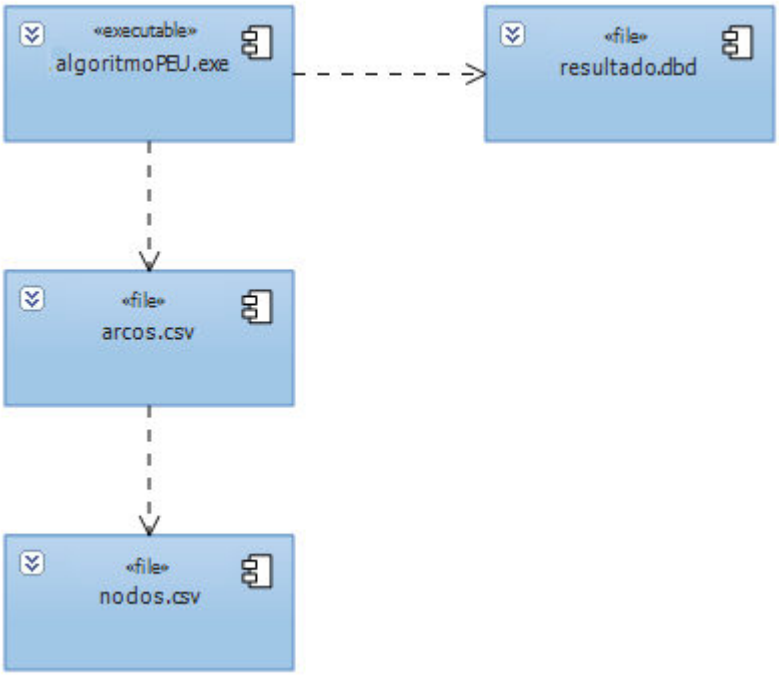



**Figura 5.15** *Diagrama de componentes de la implementación del algoritmo que resuelve el PEU*
(Elaboración propia)

Como se mencionó en el flujo de trabajo de diseño, se decidió seguir una arquitectura de tres capas: presentación, lógica de negocio y datos (ver punto 5.2.2). Dicha arquitectura se ha implementado manteniendo la independencia existente entre los componentes de cada capa.

A continuación se describen los paquetes de la aplicación:

- **Paquete de presentación**

  El paquete de presentación reúne todos los componentes de la aplicación que forman parte de la interfaz de usuario (ver figura 5.16).

- **Paquete de lógica de negocio**

  El paquete de lógica de negocio posee los componentes que forman parte de la capa de lógica del negocio de la aplicación (ver figura 5.17).

- **Paquete de datos**

  El paquete de datos se conforma únicamente por el componente que accede a los datos de los archivos CSV (ver figura 5.18).

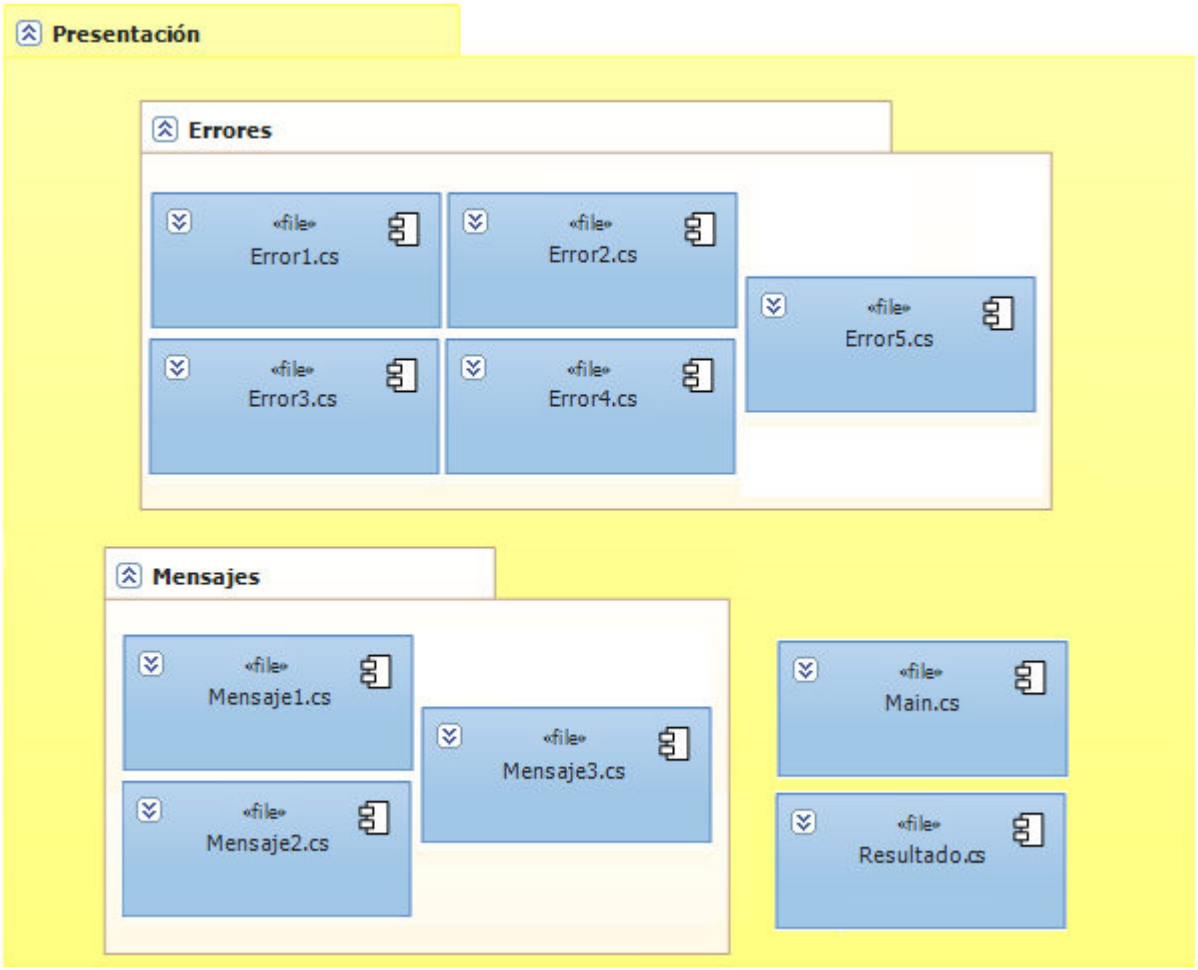



**Figura 5.16** *Paquete de presentación de la implementación del algoritmo que resuelve el PEU* (Elaboración propia)

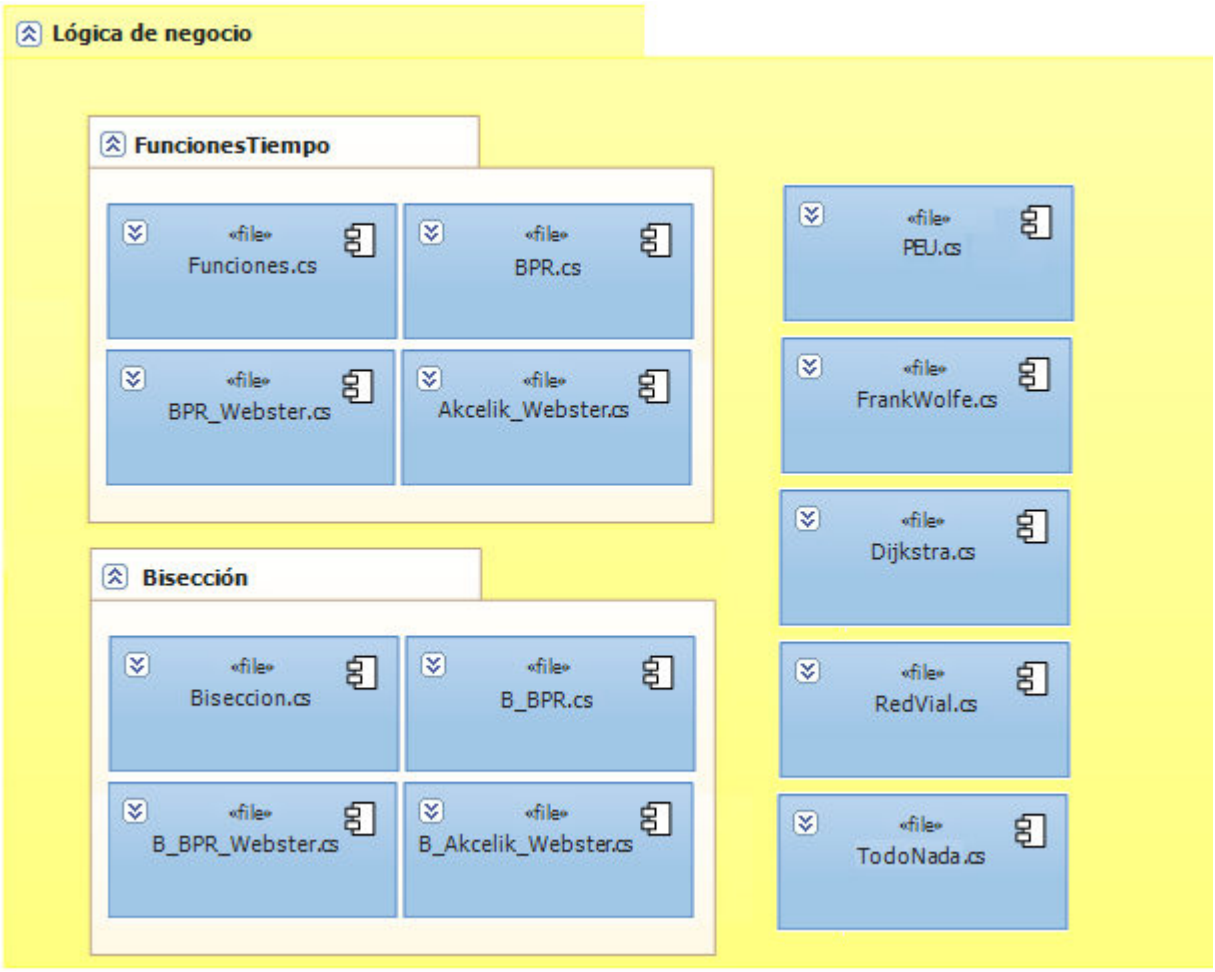



**Figura 5.17** *Paquete de lógica de negocio de la implementación del algoritmo que resuelve el PEU*
(Elaboración propia)


**Figura 5.18** *Paquete de datos de la implementación del algoritmo que resuelve el PEU*
(Elaboración propia)

## 5.4 Flujo de trabajo de pruebas

En el flujo de trabajo de pruebas se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como la versión final (Jacobson et al., 2003).

### 5.4.1 Modelo de pruebas

El modelo de pruebas describe cómo se prueban los diversos componentes ejecutables que existen en el modelo de implementación haciendo uso de pruebas de integración o de sistema (ver figura 5.19).

- **Caso de prueba**

  Los casos de prueba especifican una forma de probar el sistema, incluyendo la entrada o el resultado con el que se ha de probar y las condiciones bajo las que ha de probarse.

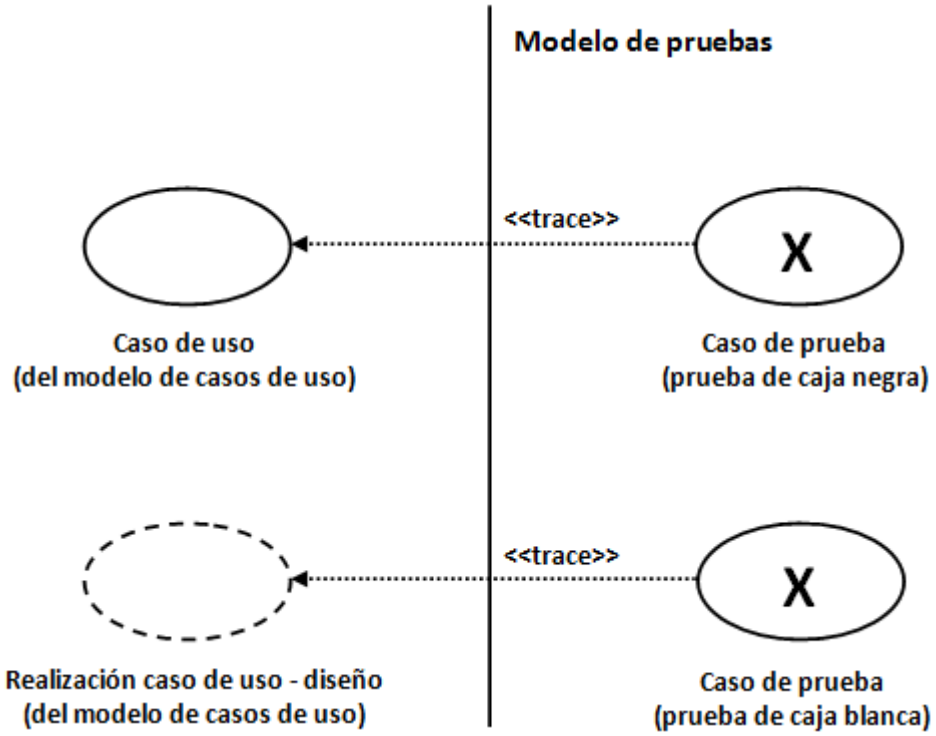



**Figura 5.19** *Un caso de prueba puede derivarse de un caso de uso o de una realización de cada caso de uso en el modelo de diseño*
(Elaboración propia)

- **Procedimiento de prueba**

  El procedimiento de prueba específica cómo realizar uno o varios casos de prueba.

En las figuras 5.20 – 5.23 se presenta el modelo de pruebas para la implementación del algoritmo que resuelve el PEU. Cada figura integra el caso de prueba y procedimiento de prueba correspondiente para cada caso de uso descrito en el modelo de casos de uso (ver punto 5.2.1).

| | | | |
|---|---|---|---|
| **Caso de prueba:** | 1. Introducir matriz Origen-Destino. | | |
| **Tipo de prueba:** | Funcional | | |
| **Descripción:** | En esta prueba el usuario se encarga de introducir el valor de la demanda vehicular (matriz O-D) en la interfaz gráfica de la aplicación. | | |
| **Fecha de realización:** | 23 de Marzo de 2011 | | |
| **Condiciones:** | 1. La aplicación debe cargarse correctamente. | | |
| **Datos de entrada:** | 1. Demanda vehicular [veh/h]: {entero >= 0} | | |
| **Procedimiento de prueba** | | | |
| **Paso** | **Detalles** | **Resultado esperado** | **Estado** |
| 01 | El usuario ejecuta la aplicación. | Se despliega en la pantalla la interfaz gráfica de usuario (GUI). | Aprobado. |
| 02 | El usuario ingresa en la aplicación el valor de la demanda vehicular (matriz O-D). | El valor se almacena temporalmente. | Aprobado. |
| **Notas** | | | |
| 1. El usuario ingresa la demanda vehicular escribiendo el valor en la interfaz gráfica de la aplicación. | | | |

**Figura 5.20** *Caso y procedimiento de prueba introducir matriz Origen-Destino*
(Elaboración propia)

| | |
|---|---|
| **Caso de prueba:** | 2. Seleccionar funciones. |
| **Tipo de prueba:** | Funcional |
| **Descripción:** | En esta prueba el usuario debe seleccionar las funciones de tiempo de viaje que utiliza el algoritmo Frank Wolfe. |
| **Fecha de realización:** | 23 de Marzo de 2011 |
| **Condiciones:** | 1. La aplicación debe cargarse correctamente. |
| **Datos de entrada:** | 1. Funciones tiempo de viaje: {BPR, BPR-Webster, Akcelik-Webster} |

**Procedimiento de prueba**

| Paso | Detalles | Resultado esperado | Estado |
|---|---|---|---|
| 01 | El usuario ejecuta la aplicación. | Se despliega en la pantalla la interfaz gráfica de usuario (GUI). | Aprobado. |
| 02 | El usuario elige el tipo de función de tiempo de viaje a ser utilizada. | El tipo de función de tiempo de viaje se almacena temporalmente. | Aprobado. |

**Notas**

1. El usuario solo puede seleccionar una de las tres combinaciones de las funciones desplegadas.

**Figura 5.21** *Caso y procedimiento de prueba seleccionar función*
(Elaboración propia)

| | |
|---|---|
| **Caso de prueba:** | 3. Estimar flujo vehicular. |
| **Tipo de prueba:** | Funcional |
| **Descripción:** | En esta prueba se ejecuta el algoritmo que resuelve el PEU obteniendo el patrón de flujo estimado. |
| **Fecha de realización:** | 23 de Marzo de 2011 |
| **Condiciones:** | 1. La aplicación debe cargarse correctamente. |
| **Datos de entrada:** | 1. Demanda vehicular [veh/h]: {entero >= 0}<br>2. Funciones tiempo de viaje: {BPR, BPR-Webster, Akcelik-Webster} |

**Procedimiento de prueba**

| Paso | Detalles | Resultado esperado | Estado |
|---|---|---|---|
| 01 | El usuario ejecuta la aplicación. | Se despliega en la pantalla la interfaz gráfica de usuario (GUI). | Aprobado. |
| 02 | El usuario ingresa en la aplicación el valor de la demanda vehicular (matriz O-D). | El valor se almacena temporalmente. | Aprobado. |
| 03 | El usuario elige el tipo de función de tiempo de viaje a ser utilizada. | El tipo de función de tiempo de viaje se almacena temporalmente. | Aprobado. |
| 04 | El usuario ejecuta el algoritmo. | Se despliega una nueva ventana que muestra los resultados obtenidos. | Aprobado. |
| 05 | El usuario verifica los resultados. | El flujo se ha estimado dentro de la red vial de prueba implementada. | Aprobado. |

**Notas**

1. Las funciones de tiempo de viaje poseen un valor de demanda mínimo para equilibrar el flujo.

**Figura 5.22** *Caso y procedimiento de prueba equilibrar flujo* (Elaboración propia)

| | |
|---|---|
| **Caso de prueba:** | 4. Visualizar resultado. |
| **Tipo de prueba:** | Funcional |
| **Descripción:** | En esta prueba el usuario se conecta con TransCAD para generar la representación geográfica del patrón de flujo estimado. |
| **Fecha de realización:** | 23 de Marzo de 2011 |
| **Condiciones:** | 1. Patrón de flujo estimado.<br>2. TransCAD se ha ejecutado y permanece abierto. |
| **Datos de entrada:** | Ninguno |

**Procedimiento de prueba**

| Paso | Detalles | Resultado esperado | Estado |
|---|---|---|---|
| 01 | El usuario elige la opción para conectarse con el TransCAD. | Se despliega mensaje de éxito. | Aprobado. |
| 02 | El usuario elige visualizar resultado. | Se genera la representación geográfica dentro de TransCAD. | Aprobado. |

**Notas**

1. El usuario puede finalizar la conexión establecida desde la aplicación o cerrando TransCAD.

**Figura 5.23** *Caso y procedimiento de prueba visualizar resultado*
(Elaboración propia)

Antes de proveer las conclusiones finales de este trabajo, en el siguiente capítulo se presentan los resultados más relevantes de la implementación del algoritmo que resuelve el problema de equilibrio del usuario para diferentes matrices Origen-Destino [veh/h]. De igual forma, se analiza el comportamiento del algoritmo Frank Wolfe para las funciones de viaje: BPR, BPR-Webster y Akcelik-Webster.

# Capítulo 6

# Análisis de Resultados

En la presente tesis se ha implementado con éxito el algoritmo Frank Wolfe para dar solución al problema de equilibrio del usuario en una red vial que se conforma por diferentes tipos de arcos. A continuación se proporciona el análisis de resultados para el trabajo que se ha llevado a cabo.

## 6.1 Ejecución del algoritmo

Para ejecutar el algoritmo es necesario: introducir la matriz Origen-Destino [veh/h] y elegir las funciones de tiempo de viaje que ocupará el algoritmo Frank Wolfe para trabajar con la red vial.

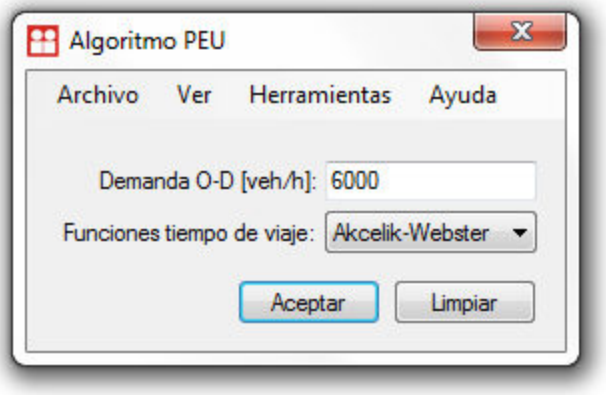



**Figura 6.1** *Parametros de entrada para ejecutar el algoritmo que soluciona el PEU*
(Elaboración propia)

El resultado desplegado dentro de la figura 6.2 muestra el flujo ya distribuido sobre la red vial. La ventana provee tanto los datos que ingresó el usuario como los datos que son generados a partir de la asignación efectuada. Cabe destacar que es posible abrir una o más ventanas de resultados para comparar procesos de carga con diferentes demandas vehiculares o combinaciones de funciones de tiempo de viaje.

Para las dos rutas que conforman la red vial (Segundo Piso de Periférico y Revolución), el flujo se ha equilibrado, es decir, los usuarios que se encuentran en la red vial no pueden cambiar su ruta sin incrementar su tiempo de viaje. En la ventana de resultados se muestra claramente el valor del flujo en [veh/h] que ha sido estimado en los arcos que conforman las rutas de Periférico y Revolución, así como los tiempos de viaje en [h] para dichas rutas; también es proporcionado el valor del costo del sistema en [h].

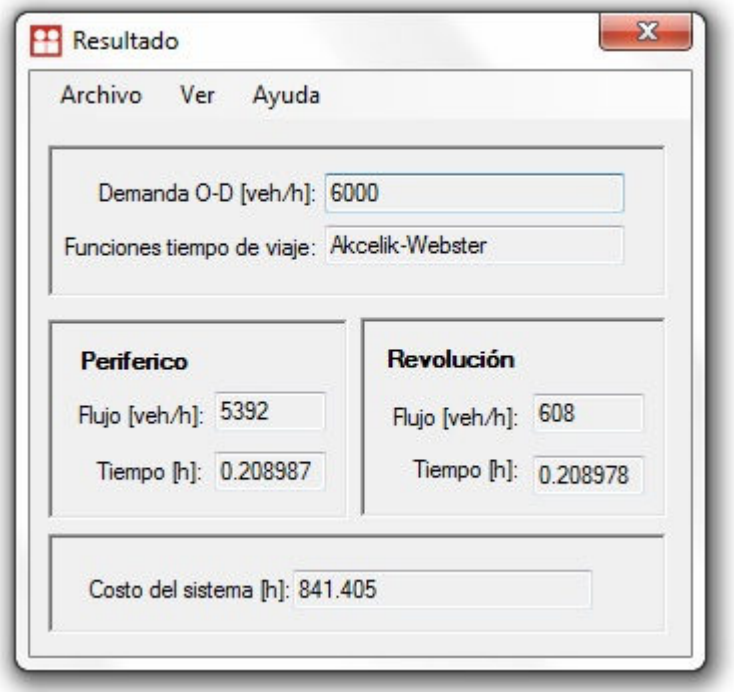



**Figura 6.2** *Información generada por la ejecución del algoritmo* (Elaboración propia)

Además, el usuario puede visualizar geográficamente el patrón de flujo estimado si se cuenta con versiones superiores o iguales del SIG TransCAD 4.8. Para ello, basta con ejecutar dicho SIG, establecer la conexión desde la pestaña de herramientas del menú que se encuentra en la ventana principal de la aplicación, y posteriormente, desde la ventana de resultados, elegir la opción para representar el patrón de flujo estimado que se encuentra en la pestaña “Ver”. Para conocer el proceso de conexión del SIG TransCAD con la aplicación ver las figuras 6.3 y 6.4.

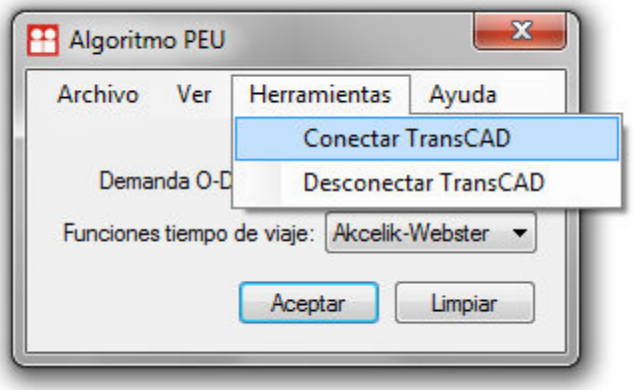



**Figura 6.3** *El usuario elige conectarse con TransCAD*
(Elaboración propia)

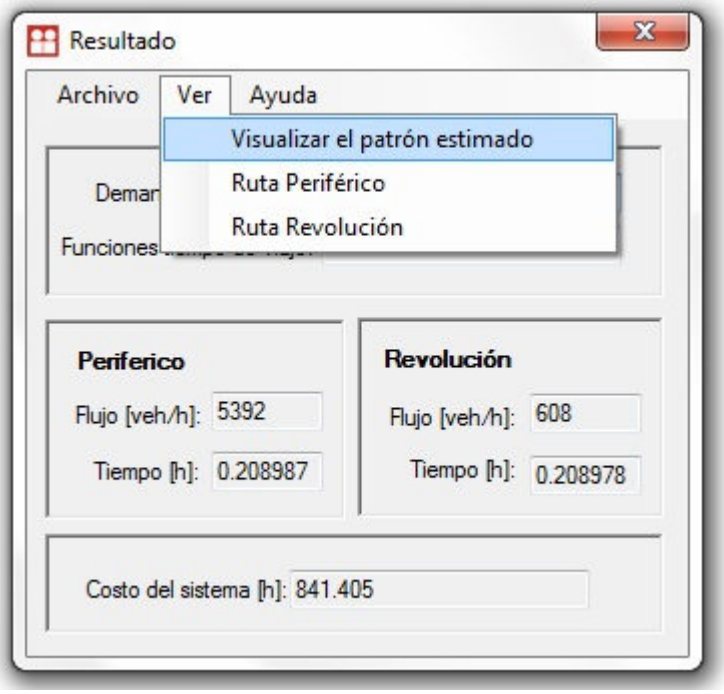



**Figura 6.4** *Entonces el usuario elige visualizar el patrón estimado de flujo en TransCAD*
(Elaboración propia)

Una vez que la conexión se ha establecido con éxito, se despliega una figura con el patrón de flujo. La figura (6.5) muestra el patrón de flujo estimado, en el SIG TransCAD. Como se observa, la red vial conformada por el corredor de Revolución y el segundo piso de Periférico de la ciudad de México, se ha cargado con una demanda vehicular de 7500 [veh/h], utilizando para ello, el algoritmo Frank Wolfe y las funciones de tiempo de viaje Akcelik-Webster.

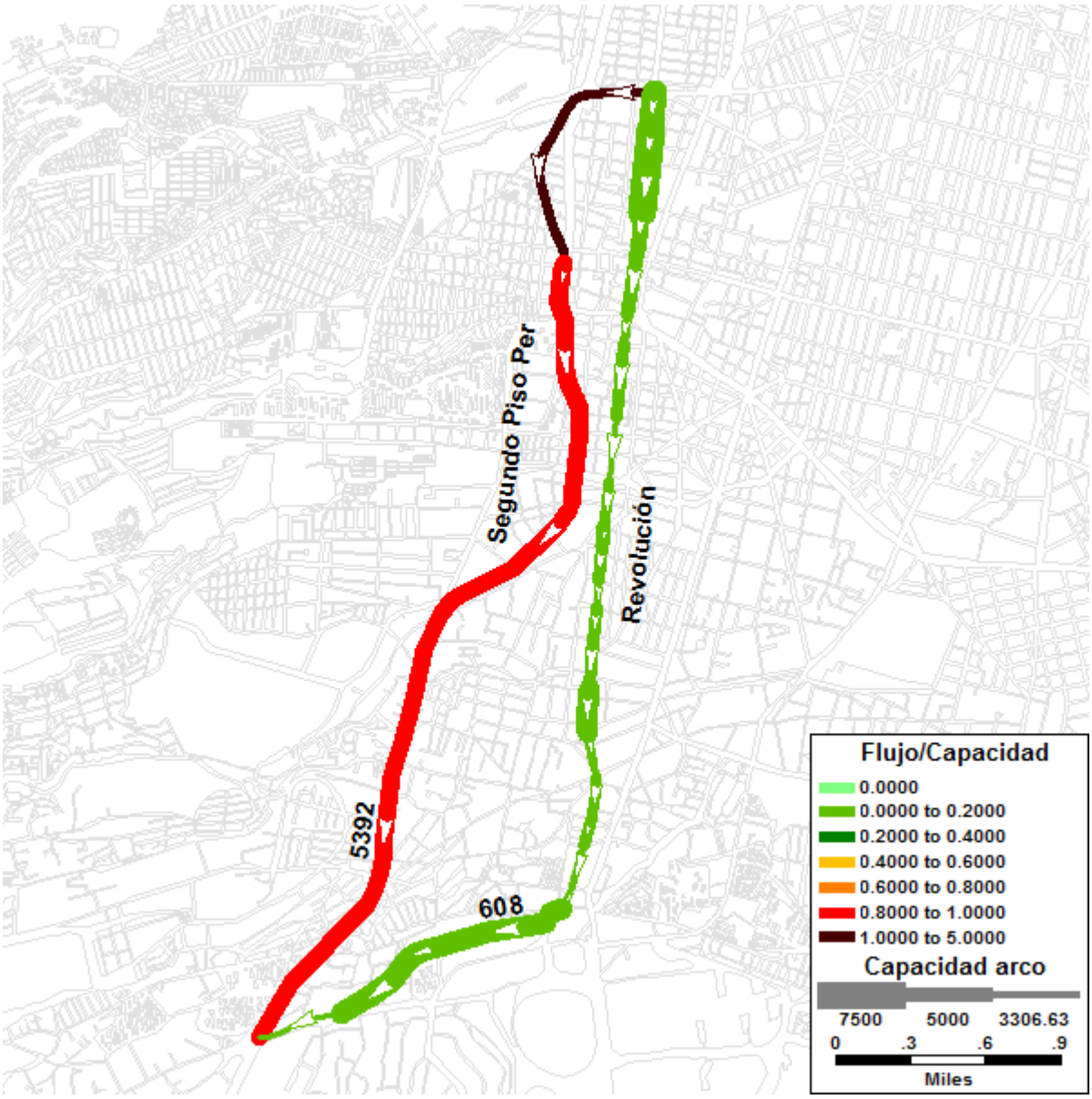



**Figura 6.5** *Representación geográfica del flujo vehicular dentro de TransCAD*
(Elaboración propia)

Es importante mencionar, que, el color de cada arco en la figura corresponde a la relación flujo/capacidad; dichos colores se muestran en la leyenda existente en la parte inferior derecha. Además, en la ventana de resultado es posible verificar para cada proceso de carga, los tiempos parciales y acumulados en [h] para las dos rutas que conforman el patrón de flujo estimado. Para ello, solo basta con seleccionar en el menu la pestaña “Ver” y posteriormente elegir la Ruta deseada (observar las figuras 6.6 y 6.7).

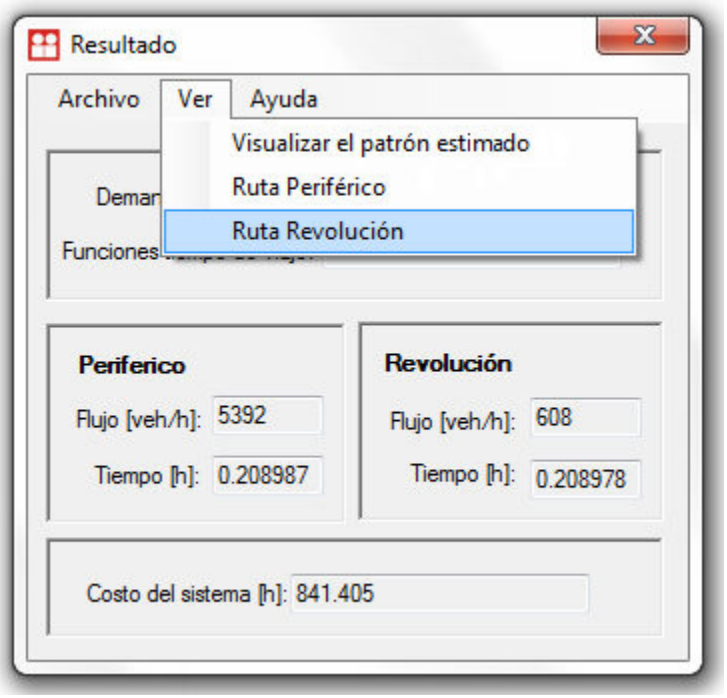
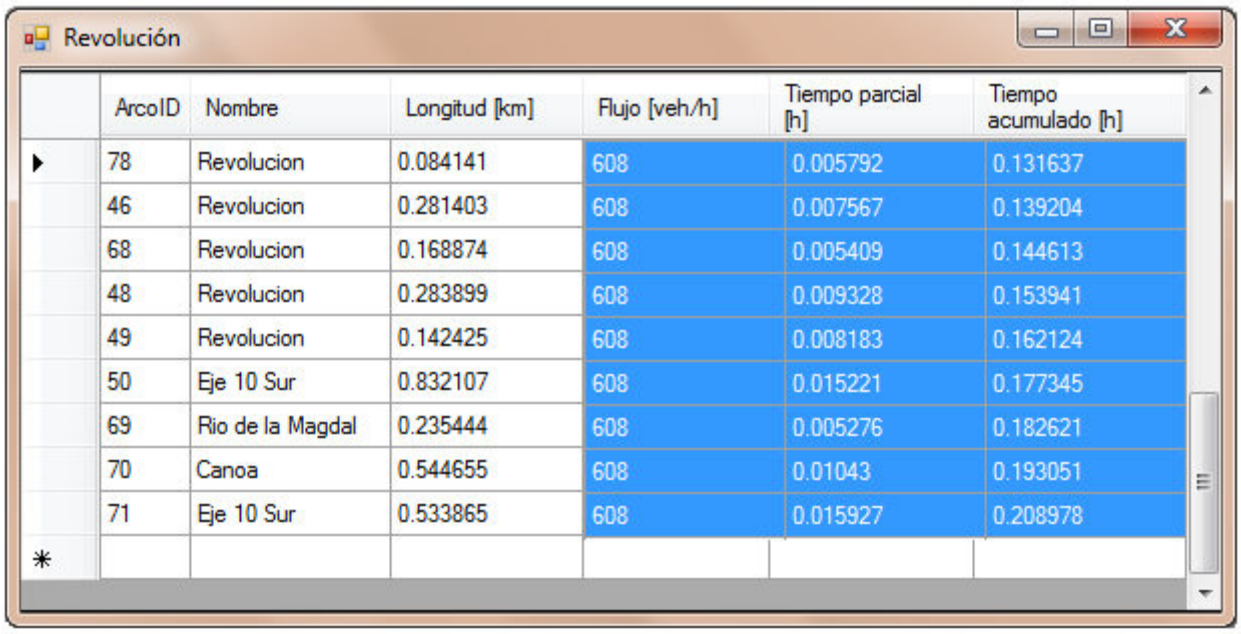



**Figura 6.6** *El usuario elige visualizar datos de la ruta de Revolución*
(Elaboración propia*)*




| ArcoID | Nombre | Longitud [km] | Flujo [veh/h] | Tiempo parcial [h] | Tiempo acumulado [h] |
|---|---|---|---|---|---|
| 78 | Revolucion | 0.084141 | 608 | 0.005792 | 0.131637 |
| 46 | Revolucion | 0.281403 | 608 | 0.007567 | 0.139204 |
| 68 | Revolucion | 0.168874 | 608 | 0.005409 | 0.144613 |
| 48 | Revolucion | 0.283899 | 608 | 0.009328 | 0.153941 |
| 49 | Revolucion | 0.142425 | 608 | 0.008183 | 0.162124 |
| 50 | Eje 10 Sur | 0.832107 | 608 | 0.015221 | 0.177345 |
| 69 | Rio de la Magdal | 0.235444 | 608 | 0.005276 | 0.182621 |
| 70 | Canoa | 0.544655 | 608 | 0.01043 | 0.193051 |
| 71 | Eje 10 Sur | 0.533865 | 608 | 0.015927 | 0.208978 |

**Figura 6.7** *Tiempos parciales y acumulados de la ruta de Revolución*
(Elaboración propia)

## 6.2 Comportamiento de las funciones de tiempo de viaje, en el algoritmo Frank Wolfe

La distribución del flujo vehicular sobre la red vial varía para cada una de las funciones de tiempo viaje utilizadas por el algoritmo Frank Wolfe durante el proceso de asignación de tráfico. Por lo cual, el resultado final es totalmente dependiente de las funciones elegidas. Esta situación genera diferentes escenarios que permiten analizar el comportamiento que presenta el algoritmo y la red vial para los diferentes procesos de carga. Con el propósito de verificar el funcionamiento de la implementación del algoritmo Frank Wolfe para cada una de las funciones de tiempo de viaje definidas: BPR, BPR-Webster y Akcelik-Webster, se presentan los resultados más representativos para el escenario que asigna una demanda vehicular de 7500 [veh/h].

### 6.2.1 Resultados para la función BPR

El comportamiento del algoritmo Frank Wolfe es sumamente estable al trabajar con la función BPR. Dicha función crece exponencialmente para la relación flujo/capacidad y considera que el tiempo de viaje en todos los arcos que componen cada corredor vial es el mismo (Segundo piso periférico y Revolución); por lo cual, ignora las características de la vía en términos del control semaforizado. Las tablas (6.1, 6.2 y, 6.3) contienen los resultados para la asignación de 7500 [veh/h].

**Tabla 6.1** *Resultados estimación 7500 [veh/h] con la función BPR* (Elaboración propia)

| Función de tiempo de viaje | Demanda vehicular [veh/h] | Flujo ruta Periférico [veh/h] | Tiempo ruta Periférico [h] | Flujo ruta Revolución [veh/h] | Tiempo ruta Revolución [h] | Costo del sistema [h] |
|---|---|---|---|---|---|---|
| BPR | 7500 | 5895 | 0.12832 | 1605 | 0.128332 | 924.653 |

**Tabla 6.2** *Segundo piso Periférico: demanda vehicular 7500 [veh/h] empleando la función BPR* (Elaboración propia)

| Arco ID | Nombre | Longitud [km] | Capacidad arco [veh] | Flujo [veh/h] | Tiempo parcial [h] | Tiempo acumulado [h] | Flujo / Capacidad |
|---|---|---|---|---|---|---|---|
| 1 | Av. San Antonio | 0.4366 | 4000 | 5895 | 0.008208 | 0.008208 | 1.47377 |
| 74 | Segundo Piso Per | 1.188127 | 4000 | 5895 | 0.022393 | 0.030601 | 1.47377 |
| 73 | Segundo Piso Per | 0.201274 | 6000 | 5895 | 0.003482 | 0.034083 | 0.98251 |
| 75 | Segundo Piso Per | 0.878909 | 6000 | 5895 | 0.015204 | 0.049287 | 0.98251 |
| 76 | Segundo Piso Per | 1.464018 | 6000 | 5895 | 0.025326 | 0.074613 | 0.98251 |
| 77 | Segundo Piso Per | 3.104629 | 6000 | 5895 | 0.053707 | 0.12832 | 0.98251 |

| Arco ID | Nombre | Longitud [km] | Capacidad arco [veh] | Flujo [veh/h] | Tiempo parcial [h] | Tiempo acumulado [h] | Flujo / Capacidad |
|---|---|---|---|---|---|---|---|
| 29 | Revolucion | 0.227672 | 7611.36 | 1605 | 0.003887 | 0.003887 | 0.21086 |
| 57 | Revolucion | 0.226836 | 8213.07 | 1605 | 0.00386 | 0.007747 | 0.19541 |
| 31 | Revolucion | 0.306455 | 8213.07 | 1605 | 0.005173 | 0.01292 | 0.19541 |
| 58 | Revolucion | 0.222364 | 5523.46 | 1605 | 0.003821 | 0.016741 | 0.29056 |
| 60 | Revolucion | 0.058622 | 5523.46 | 1605 | 0.001007 | 0.017748 | 0.29056 |
| 59 | Revolucion | 0.361574 | 5236.4 | 1605 | 0.006152 | 0.0239 | 0.30649 |
| 62 | Revolucion | 0.19977 | 4053.33 | 1605 | 0.003556 | 0.027456 | 0.39595 |
| 61 | Revolucion | 0.463248 | 5036.27 | 1605 | 0.007887 | 0.035343 | 0.31867 |
| 40 | Revolucion | 0.402258 | 3461.8 | 1605 | 0.00731 | 0.042653 | 0.46361 |
| 41 | Revolucion | 0.28311 | 4446.63 | 1605 | 0.004986 | 0.047639 | 0.36093 |
| 63 | Revolucion | 0.131752 | 5381.43 | 1605 | 0.002248 | 0.049887 | 0.29823 |
| 42 | Revolucion | 0.21205 | 4905.8 | 1605 | 0.003751 | 0.053638 | 0.32715 |
| 64 | Revolucion | 0.180144 | 3930.47 | 1605 | 0.003378 | 0.057016 | 0.40833 |
| 65 | Revolucion | 0.196042 | 4849.43 | 1605 | 0.003451 | 0.060467 | 0.33095 |
| 66 | Revolucion | 0.323824 | 4825.37 | 1605 | 0.005655 | 0.066122 | 0.3326 |
| 67 | Revolucion | 0.269949 | 6627.83 | 1605 | 0.004501 | 0.070623 | 0.24215 |
| 45 | Revolucion | 0.194886 | 4051.43 | 1605 | 0.003559 | 0.074182 | 0.39614 |
| 78 | Revolucion | 0.084141 | 3867.13 | 1605 | 0.001632 | 0.075814 | 0.41502 |
| 46 | Revolucion | 0.281403 | 4578.37 | 1605 | 0.004943 | 0.080757 | 0.35054 |
| 68 | Revolucion | 0.168874 | 4731 | 1605 | 0.003003 | 0.08376 | 0.33923 |
| 48 | Revolucion | 0.283899 | 3780.37 | 1605 | 0.005289 | 0.089049 | 0.42454 |
| 49 | Revolucion | 0.142425 | 3306.63 | 1605 | 0.002826 | 0.091875 | 0.48536 |
| 50 | Eje 10 Sur | 0.832107 | 6887.5 | 1605 | 0.013897 | 0.105772 | 0.23302 |
| 69 | Rio Magdal | 0.235444 | 6887.5 | 1605 | 0.003959 | 0.109731 | 0.23302 |
| 70 | Canoa | 0.544655 | 6887.5 | 1605 | 0.009118 | 0.118849 | 0.23302 |
| 71 | Eje 10 Sur | 0.533865 | 3541.13 | 1605 | 0.009483 | 0.128332 | 0.45322 |

**Tabla 6.3** *Revolución: demanda vehicular 7500 [veh/h] empleando la función BPR*
(Elaboración propia)

En la figura 6.8 se aprecia que el flujo se ha distribuido con éxito sobre la red vial, sin embargo, la asignación no representa adecuadamente la operación real de la red vial debido a que la función BPR, como ya se ha mencionado anteriormente, no considera los ciclos de semáforo para los arcos de Revolución.

Se observa también que ciertos arcos presentan una mayor congestión, esto es ocasionado por la reducción de la capacidad vehicular en algunos de ellos, ya que poseen menor número de carriles (recordar que el grosor de las líneas es proporcional a la capacidad de los arcos).

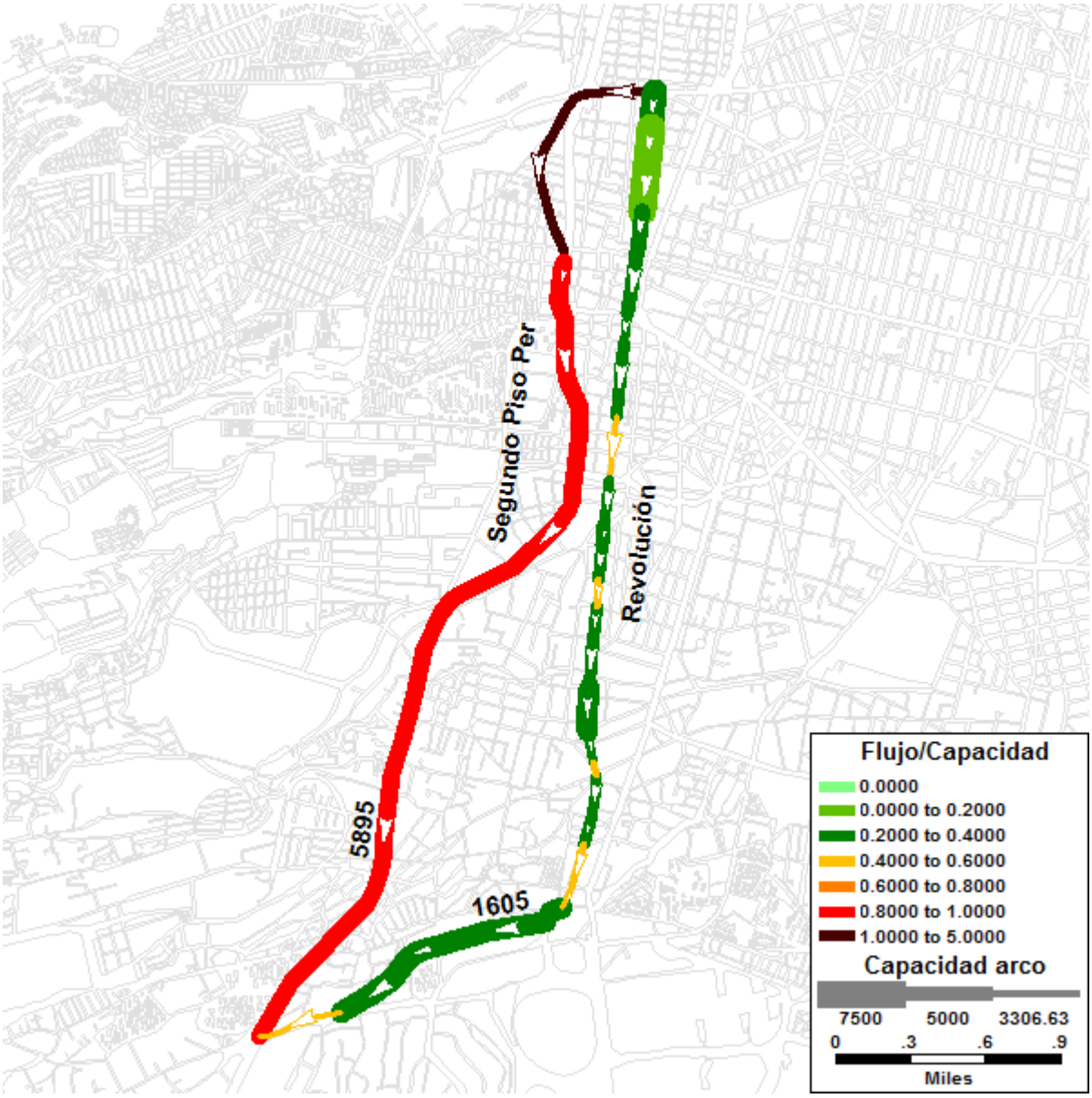



**Figura 6.8** *Comportamiento de la red vial: demanda de 7500 [veh/h] - función BPR*
(Elaboración propia)

### 6.2.2 Resultados para la combinación de las funciones BPR-Webster

El algoritmo Frank Wolfe emplea la combinación de funciones BPR y Webster con el propósito de obtener una mejor representación de la operación para los diferentes tipos de arco en la red vial. Para el Segundo piso de Periférico se emplea la función BPR, y para el corredor de Revolución se hace uso de la función de Webster, esto con el objeto de considerar los ciclos de los semáforos, los tiempos de verde y el tiempo de congestión en los arcos de tipo semaforizado. Las tablas 6.4, 6.5 y 6.6 contienen los resultados para la asignación de 7500 [veh/h].

**Tabla 6.4** *Resultados para la asignación de 7500 [veh/h] para la función BPR-Webster*
(Elaboración propia)

| Función de tiempo de viaje | Demanda vehicular [veh/h] | Flujo ruta Periférico [veh/h] | Tiempo ruta Periférico [h] | Flujo ruta Revolución [veh/h] | Tiempo ruta Revolución [h] | Costo del sistema [h] |
|---|---|---|---|---|---|---|
| BPR Webster | 7500 | 7500 | 0.136375 | 0 | 0.20266 | 936.597 |

**Tabla 6.5** *Segundo piso Periférico: demanda vehicular 7500 [veh/h] para la función BPR-Webster*
(Elaboración propia)

| Arco ID | Nombre | Longitud [km] | Capacidad arco [veh] | Flujo [veh/h] | Tiempo parcial [h] | Tiempo acumulado [h] | Flujo / Capacidad |
|---|---|---|---|---|---|---|---|
| 1 | Av. San Antonio | 0.4366 | 4000 | 7000 | 0.009195 | 0.009195 | 1.875 |
| 74 | Segundo Piso Per | 1.188127 | 4000 | 7000 | 0.025677 | 0.034872 | 1.875 |
| 73 | Segundo Piso Per | 0.201274 | 6000 | 7000 | 0.003617 | 0.038489 | 1.25 |
| 75 | Segundo Piso Per | 0.878909 | 6000 | 7000 | 0.015793 | 0.054282 | 1.25 |
| 76 | Segundo Piso Per | 1.464018 | 6000 | 7000 | 0.026307 | 0.080589 | 1.25 |
| 77 | Segundo Piso Per | 3.104629 | 6000 | 7000 | 0.055786 | 0.136375 | 1.25 |

**Tabla 6.6** *Revolución: demanda vehicular 7500 [veh/h] para la función BPR-Webster*
(Elaboración propia)

| Arco ID | Nombre | Longitud [km] | Capacidad arco [veh] | Flujo [veh/h] | Tiempo parcial [h] | Tiempo acumulado [h] | Flujo / Capacidad |
|---|---|---|---|---|---|---|---|
| 29 | Revolucion | 0.227672 | 7611.36 | 0 | 0.007964 | 0.007964 | 0 |
| 57 | Revolucion | 0.226836 | 8213.07 | 0 | 0.007302 | 0.015266 | 0 |
| 31 | Revolucion | 0.306455 | 8213.07 | 0 | 0.008629 | 0.023895 | 0 |
| 58 | Revolucion | 0.222364 | 5523.46 | 0 | 0.006626 | 0.030521 | 0 |
| 60 | Revolucion | 0.058622 | 5523.46 | 0 | 0.003897 | 0.034418 | 0 |
| 59 | Revolucion | 0.361574 | 5236.4 | 0 | 0.010898 | 0.045316 | 0 |
| 62 | Revolucion | 0.19977 | 4053.33 | 0 | 0.006959 | 0.052275 | 0 |
| 61 | Revolucion | 0.463248 | 5036.27 | 0 | 0.009617 | 0.061892 | 0 |
| 40 | Revolucion | 0.402258 | 3461.8 | 0 | 0.011646 | 0.073538 | 0 |
| 41 | Revolucion | 0.28311 | 4446.63 | 0 | 0.007588 | 0.081126 | 0 |
| 63 | Revolucion | 0.131752 | 5381.43 | 0 | 0.003616 | 0.084742 | 0 |
| 42 | Revolucion | 0.21205 | 4905.8 | 0 | 0.005629 | 0.090371 | 0 |
| 64 | Revolucion | 0.180144 | 3930.47 | 0 | 0.006888 | 0.097259 | 0 |
| 65 | Revolucion | 0.196042 | 4849.43 | 0 | 0.00545 | 0.102709 | 0 |
| 66 | Revolucion | 0.323824 | 4825.37 | 0 | 0.007619 | 0.110328 | 0 |
| 67 | Revolucion | 0.269949 | 6627.83 | 0 | 0.004772 | 0.1151 | 0 |
| 45 | Revolucion | 0.194886 | 4051.43 | 0 | 0.006882 | 0.121982 | 0 |
| 78 | Revolucion | 0.084141 | 3867.13 | 0 | 0.005423 | 0.127405 | 0 |
| 46 | Revolucion | 0.281403 | 4578.37 | 0 | 0.007325 | 0.13473 | 0 |
| 68 | Revolucion | 0.168874 | 4731 | 0 | 0.00519 | 0.13992 | 0 |
| 48 | Revolucion | 0.283899 | 3780.37 | 0 | 0.008941 | 0.148861 | 0 |
| 49 | Revolucion | 0.142425 | 3306.63 | 0 | 0.007693 | 0.156554 | 0 |
| 50 | Eje 10 Sur | 0.832107 | 6887.5 | 0 | 0.015129 | 0.171683 | 0 |
| 69 | Rio Magdal | 0.235444 | 6887.5 | 0 | 0.005184 | 0.176867 | 0 |
| 70 | Canoa | 0.544655 | 6887.5 | 0 | 0.010338 | 0.187205 | 0 |
| 71 | Eje 10 Sur | 0.533865 | 3541.13 | 0 | 0.015455 | 0.20266 | 0 |

Una vez que el algoritmo ha distribuido el flujo sobre la red, se observa que el total de la demanda vehicular que el usuario introdujo (7500 veh/h) ha sido asignada al Segundo piso de Periférico. Esto se debe fundamentalmente a que la función de Webster para el corredor de Revolución se encuentra tomando en cuenta los tiempos de congestión; en cambio, la función BPR aplicada al Segundo piso de Periférico ignora por completo dichos tiempos, lo que tiene

como resultado, que el algoritmo considere que el tiempo de viaje sea menor a través del Segundo piso de Periférico. Como se observa en la figura 6.6, por el segundo Piso de Periférico la demanda vehicular rebasa la capacidad de todos los arcos (relación flujo/capacidad > 1), esto no significa un alto total en el flujo a través de dicho corredor, pero si un nivel de congestión muy elevado o crítico que limita considerablemente la operación en la red. Caso contrario para Revolución, en donde se aprecia que no se ha asignado flujo a ninguno de los arcos (relación flujo/capacidad = 0).

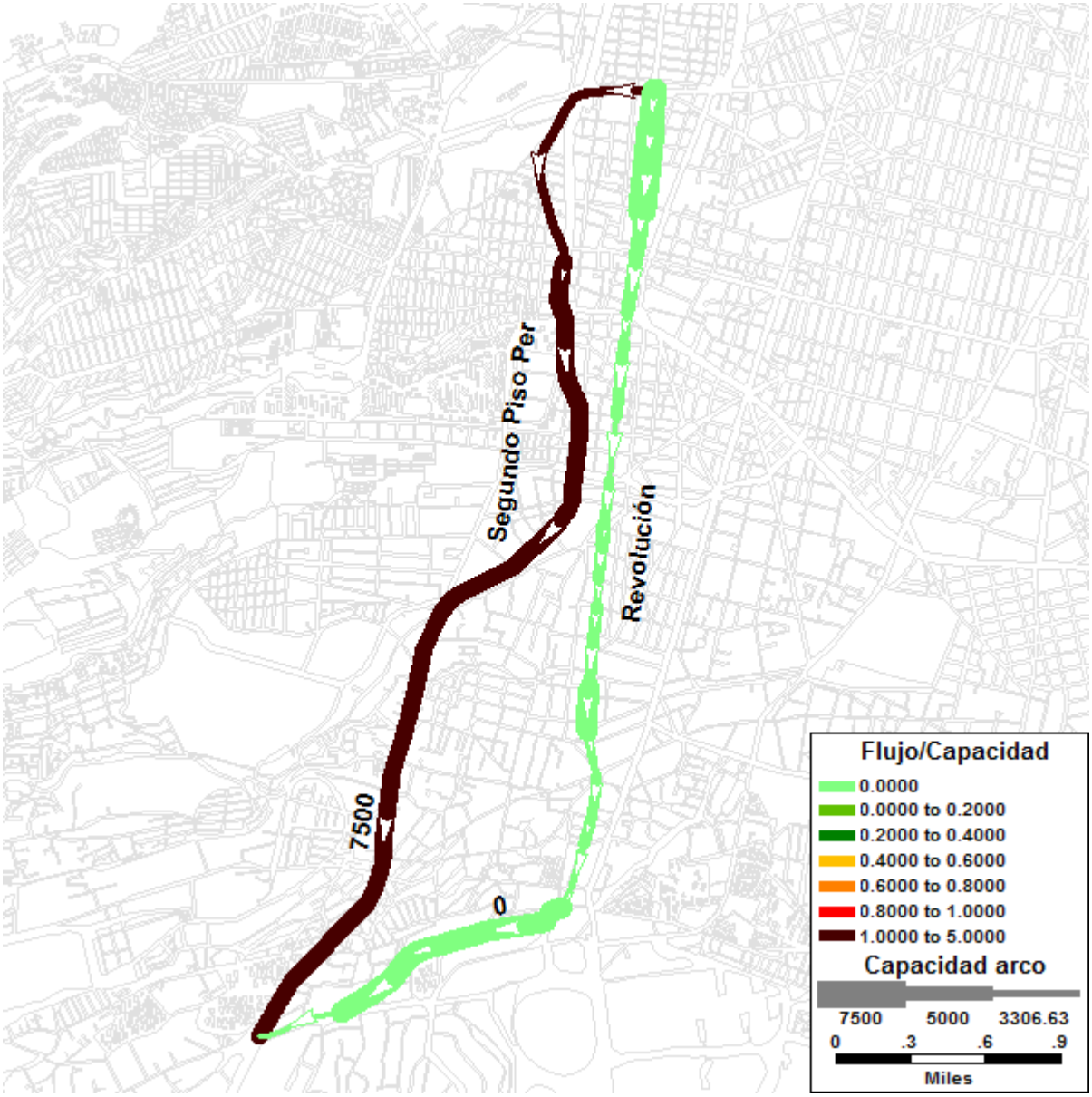



**Figura 6.9** *Comportamiento de la red vial: demanda de 7500 [veh/h] - función BPR-Webster* (Elaboración propia*)*

Para obtener una interpretación más cercana a la realidad, es necesario emplear una función de tiempo de viaje que considere los tiempos de congestión para los arcos de acceso controlado. Razón por la cual, a continuación se presenta el análisis de resultados del comportamiento del algoritmo Frank Wolfe cuando se trabaja con la combinación de las funciones Akcelik-Webster.

### 6.2.3 Resultados para la combinación de las funciones Akcelik-Webster

Mediante la combinación de las funciones Akcelik y Webster, el algoritmo Frank Wolfe genera el patrón de flujo vehicular que mejor representa las condiciones reales de operación. Esto se debe a que dicha combinación toma en cuenta la congestión de los dos corredores de la red vial de prueba. Para ello, es incorporado un nuevo valor conocido como demora promedio por sobreflujo (h/veh). Este valor se aplica en las funciones de tiempo de viaje tanto de los arcos de acceso controlado del Segundo Piso de Periférico como en los arcos semaforizados de Revolución. Las tablas 6.7, 6.8 y 6.9 contienen los resultados para la asignación de 7500 [veh/h].

**Tabla 6.7** *Resultados para la asignación de 7500 [veh/h] para la función Akcelik-Webster*
(Elaboración propia)

| Función de tiempo de viaje | Demanda vehicular [veh/h] | Flujo ruta Periférico [veh/h] | Tiempo ruta Periférico [h] | Flujo ruta Revolución [veh/h] | Tiempo ruta Revolución [h] | Costo del sistema [h] |
|---|---|---|---|---|---|---|
| Akcelik Webster | 7500 | 5652 | 0.225722 | 1848 | 0.22587 | 1166.833 |

**Tabla 6.8** *Segundo piso Periférico: demanda vehicular 7500 [veh/h] función Akcelik-Webster*
(Elaboración propia)

| Arco ID | Nombre | Longitud [km] | Capacidad arco [veh] | Flujo [veh/h] | Tiempo parcial [h] | Tiempo acumulado [h] | Flujo / Capacidad |
|---|---|---|---|---|---|---|---|
| 1 | Av. San Antonio | 0.4366 | 4000 | 5652 | 0.059002 | 0.059002 | 1.413116 |
| 74 | Segundo Piso Per | 1.188127 | 4000 | 5652 | 0.071527 | 0.130529 | 1.413116 |
| 73 | Segundo Piso Per | 0.201274 | 6000 | 5652 | 0.003616 | 0.134145 | 0.942077 |
| 75 | Segundo Piso Per | 0.878909 | 6000 | 5652 | 0.01491 | 0.149055 | 0.942077 |
| 76 | Segundo Piso Per | 1.464018 | 6000 | 5652 | 0.024662 | 0.173717 | 0.942077 |
| 77 | Segundo Piso Per | 3.104629 | 6000 | 5652 | 0.052005 | 0.225722 | 0.942077 |

**Tabla 6.9** *Revolución: demanda vehicular 7500 [veh/h] empleando la función Akcelik-Webster*
(Elaboración propia)

| Arco ID | Nombre | Longitud [km] | Capacidad arco [veh] | Flujo [veh/h] | Tiempo parcial [h] | Tiempo acumulado [h] | Flujo / Capacidad |
|---|---|---|---|---|---|---|---|
| 29 | Revolucion | 0.227672 | 7611.36 | 1848 | 0.008558 | 0.008558 | 0.242733 |
| 57 | Revolucion | 0.226836 | 8213.07 | 1848 | 0.007804 | 0.016362 | 0.224950 |
| 31 | Revolucion | 0.306455 | 8213.07 | 1848 | 0.009131 | 0.025493 | 0.224950 |
| 58 | Revolucion | 0.222364 | 5523.46 | 1848 | 0.007368 | 0.032861 | 0.334488 |
| 60 | Revolucion | 0.058622 | 5523.46 | 1848 | 0.004639 | 0.0375 | 0.334488 |
| 59 | Revolucion | 0.361574 | 5236.4 | 1848 | 0.011882 | 0.049382 | 0.352825 |
| 62 | Revolucion | 0.19977 | 4053.33 | 1848 | 0.008207 | 0.057589 | 0.455806 |
| 61 | Revolucion | 0.463248 | 5036.27 | 1848 | 0.010272 | 0.067861 | 0.366845 |
| 40 | Revolucion | 0.402258 | 3461.8 | 1848 | 0.013365 | 0.081226 | 0.533691 |
| 41 | Revolucion | 0.28311 | 4446.63 | 1848 | 0.008573 | 0.089799 | 0.415490 |
| 63 | Revolucion | 0.131752 | 5381.43 | 1848 | 0.004111 | 0.09391 | 0.343316 |
| 42 | Revolucion | 0.21205 | 4905.8 | 1848 | 0.006351 | 0.100261 | 0.376602 |
| 64 | Revolucion | 0.180144 | 3930.47 | 1848 | 0.008226 | 0.108487 | 0.470054 |
| 65 | Revolucion | 0.196042 | 4849.43 | 1848 | 0.006202 | 0.114689 | 0.380979 |
| 66 | Revolucion | 0.323824 | 4825.37 | 1848 | 0.008383 | 0.123072 | 0.382879 |
| 67 | Revolucion | 0.269949 | 6627.83 | 1848 | 0.004883 | 0.127955 | 0.278753 |
| 45 | Revolucion | 0.194886 | 4051.43 | 1848 | 0.008131 | 0.136086 | 0.456019 |
| 78 | Revolucion | 0.084141 | 3867.13 | 1848 | 0.006809 | 0.142895 | 0.477752 |
| 46 | Revolucion | 0.281403 | 4578.37 | 1848 | 0.00823 | 0.151125 | 0.403535 |
| 68 | Revolucion | 0.168874 | 4731 | 1848 | 0.006007 | 0.157132 | 0.390516 |
| 48 | Revolucion | 0.283899 | 3780.37 | 1848 | 0.010395 | 0.167527 | 0.488718 |
| 49 | Revolucion | 0.142425 | 3306.63 | 1848 | 0.009554 | 0.177081 | 0.558735 |
| 50 | Eje 10 Sur | 0.832107 | 6887.5 | 1848 | 0.015454 | 0.192535 | 0.268244 |
| 69 | Rio Magdal | 0.235444 | 6887.5 | 1848 | 0.00551 | 0.198045 | 0.268244 |
| 70 | Canoa | 0.544655 | 6887.5 | 1848 | 0.010664 | 0.208709 | 0.268244 |
| 71 | Eje 10 Sur | 0.533865 | 3541.13 | 1848 | 0.017161 | 0.22587 | 0.521736 |

Como se aprecia en las tablas 6.7 y 6.9, el flujo vehicular se distribuye sobre los dos corredores. Al considerar la demora por sobreflujo, se obtiene un costo total del sistema con un valor de 1166.833 [h] (ver tabla 6.7), el cual es mayor al obtenido con las funciones BPR y BPR-Webster.

De igual forma ocurre con el tiempo de viaje para recorrer la red, ya sea por Revolución o el Segundo piso de Periférico, con un valor 0.225 [h].

A diferencia de la función BPR- Webster, ahora se puede apreciar una mejor distribución del flujo vehicular sobre la red vial (ver figura 6.10). La mayor carga se asigna al Segundo piso de Periférico, presentando saturación en los primeros arcos que conforman el corredor, situación que mejora cuando aumenta la capacidad del corredor. Por su parte, Revolución distribuye un número considerable de vehículos, sin afectar las condiciones de operación de dicho corredor.

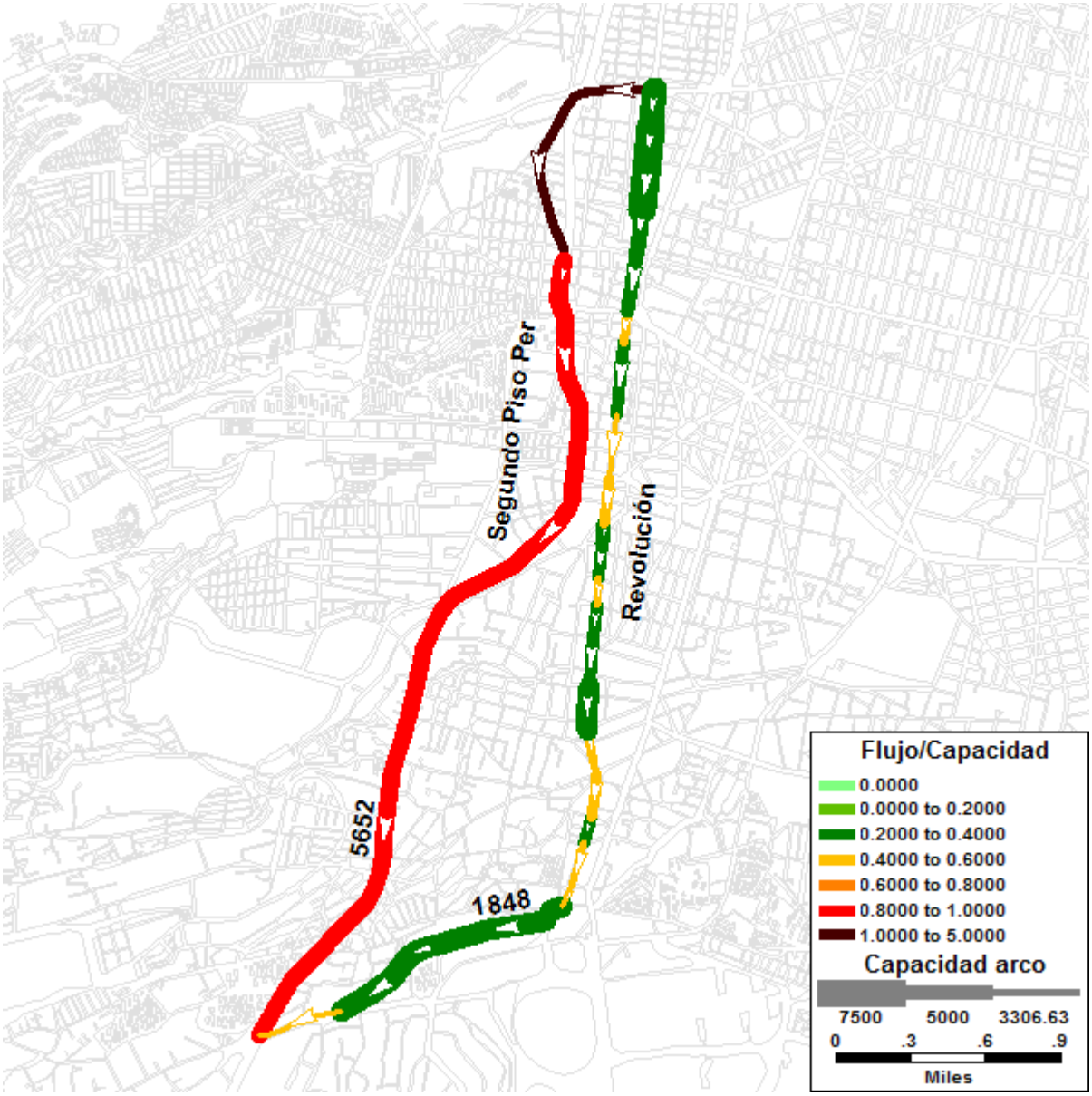



**Figura 6.10** *Comportamiento de la red vial: demanda de 7500 [veh/h] - función Akcelik-Webster* (Elaboración propia*)*

La gran ventaja de la combinación de las funciones Akcelik y Webster reside en distribuir el flujo vehicular haciendo uso de una función apropiada para trabajar con los arcos semaforizados y de acceso controlado respectivamente, así como la consideración de demoras por congestión originada por el sobreflujo, lo cual se ha implementado con éxito.

Es importante mencionar que la aplicación permite estimar el flujo vehicular para diferentes valores de demanda (matriz O-D) seleccionando alguna de las funciones implementadas: BPR, BPR-Webster y Akcelik-Webster. Dicha estimación puede ser desplegada en cualquier red vial construida previamente (para ello solo es necesario cambiar los archivos CSV de arcos y nodos).

Con el objeto de ilustrar el comportamiento del algoritmo Frank Wolfe, en el anexo 1 de la presente tesis se provee una serie de figuras generadas con TransCAD para los diversos escenarios de carga (demanda).

Para terminar, en el siguiente capítulo se brindan las conclusiones y recomendaciones finales.

# Conclusiones y trabajo futuro

A continuación se enumeran las conclusiones obtenidas para el presente trabajo, de igual forma, se provee una serie de recomendaciones que deben considerarse para continuar con el mismo.

## Conclusiones

En esta tesis se llevó a cabo la implementación de un algoritmo que permite resolver el problema de equilibrio del usuario (PEU) en una red vial congestionada que posee diferentes tipos de arcos.

Su formulación matemática, así como la de las funciones de tiempo de viaje fueron efectuadas en el Laboratorio de Transporte y Sistemas Territoriales (LTST) del Instituto de Ingeniería de la UNAM (Londoño y Lozano, 2012).

El Proceso Unificado ha sido el marco de desarrollo de software empleado en esta tesis con el objetivo de brindar estabilidad, control y organización durante la implementación realizada. Dicha metodología se adaptó de gran forma al carácter de investigación que presentó el proyecto, capturando los requisitos de mayor riesgo y modificando aquellos que surgieron al momento de evaluar y verificar las distintas posibilidades que existían para llevar a cabo la implementación. A lo largo de las iteraciones realizadas fue posible incrementar el sistema, añadiendo nuevas funcionalidades hasta que se generó la versión de la aplicación con la capacidad operativa final.

La elección de la tecnología de Microsoft .NET fue fundamental para realizar la implementación. En primer lugar, dicha tecnología facilitó la creación de la aplicación bajo Windows con suma rapidez y estabilidad, utilizando para ello el lenguaje C#. En segundo lugar, el conjunto de clases que conforma ADO.NET proporciona el objeto que permite llevar a cabo la construcción y almacenamiento en memoria de la red vial de prueba: el “DataTable”. Esta estructura cuenta con diversas operaciones de filtrado de datos que facilita a las funciones que forman parte del

algoritmo Frank Wolfe manipular la información de la red vial sin tener que acceder a una base de datos.

Ya que el framework de .NET basa su funcionalidad en clases y objetos, el diseño de la aplicación se desarrolló considerando el paradigma de programación orientada a objetos (OOP, en ingles), implementando con éxito sus características base: encapsulamiento, herencia y polimorfismo. Esto facilita enormemente la reutilización y extensión del código en futuros proyectos del LTST. Tal es el caso de la clase Dijkstra, la cual puede utilizarse para resolver problemas de ruta mínima. También, permitió crear la documentación del proyecto con una mayor rapidez y claridad, aprovechando los elementos y diagramas que provee el Lenguaje Unificado de Modelado (UML).

La conexión de la aplicación con el SIG TransCAD se ha realizado gracias a las clases .NET que se incluyen en el archivo CALIPERFORM.DLL. La clase CaliperForm.Connection permitió llevar a cabo la instanciación del objeto responsable de la conexión con el entorno GISDK$^{TM}$; el cual fue utilizado para ejecutar la macro que automatiza las operaciones que generan la representación geográfica del patrón de flujo vehicular. Dichas operaciones invocan funciones GISDK a través del lenguaje de programación Caliper Script. La documentación de dicho lenguaje se detalla dentro de la ayuda que posee el SIG TransCAD.

La aplicación construida ha resuelto con éxito el objetivo propuesto de esta tesis "*Implementar un algoritmo para resolver el problema de equilibrio del usuario (PEU) en una red vial congestionada*". La implementación es capaz de ejecutar el algoritmo Frank Wolfe con diferentes funciones de tiempo de viaje; situación que permitió verificar el desempeño de cada una de ellas. El análisis detallado de los resultados obtenidos y de las funciones de tiempo de viaje descritas queda fuera de los objetivos de la presente tesis, pero pueden ser consultados dentro del artículo próximo a publicarse llamado "*Suitable cost functions for signalized arterials and freeways, in the user equilibrium assignment problem*" escrito por Gloria Londoño y Angélica Lozano (Londoño et al., 2012) del Laboratorio de Transporte y Sistemas Territoriales (LTST) del Instituto de Ingeniería de la UNAM.

Cabe destacar que la implementación brinda al usuario la posibilidad de introducir cualquier valor para la demanda (Matriz O-D), seleccionar las funciones de tiempo de viaje, y finalmente, ejecutar el algoritmo sobre la red (subred de la red vial de la Zona Metropolitana del Valle). El algoritmo soluciona el problema de equilibrio del usuario y distribuye el flujo vehicular sobre la red vial. Las funciones de tiempo de viaje han sido programadas para trabajar con cada tipo de arco y se combinan en un solo proceso de asignación, lo que, a diferencia del software comercial, permite generar un análisis que se apega aún más al comportamiento real que existe en la red vial, ya que para cada tipo de arco es posible utilizar una función de tiempo de viaje que

considere adecuadamente las características que definen el comportamiento del flujo vehicular en dicho arco.

## Trabajo futuro y recomendaciones

En el LTST se ha iniciado una a investigación para desarrollar un modelo de asignación dinámica de tráfico, dentro del cual el algoritmo Frank Wolfe constituye un subproceso fundamental.

Por lo cual, a continuación se dan una serie de recomendaciones que deben ser tomadas en cuenta para continuar con dicho trabajo:

- La mínima diferencia que existe entre los tiempos de ruta para los corredores de Revolución y Segundo piso de Periférico en la red vial, depende directamente del método de bisección y no del algoritmo Frank Wolfe, por lo que se sugiere experimentar con otros métodos numéricos para reducir dicha diferencia.

- Con el propósito de optimizar el tiempo que se requiere para llevar a cabo la representación para visualizar el resultado en TransCAD, se aconseja exportar el patrón de flujo estimado sin utilizar archivos CSV, lo cual es posible con las funciones disponibles en el entorno GISDK™. Para llevar a cabo dicha tarea, la información necesaria se encuentra en la documentación de GISDK™ para TransCAD 5.0.

- Para continuar con el desarrollo del proyecto resulta favorable emplear el Proceso Unificado. Aunque si se cuenta con un equipo con experiencia, se puede utilizar programación extrema (XP). Esta metodología de desarrollo pone más énfasis en la adaptabilidad que en la previsibilidad, situación que se presenta varios de los requerimientos a lo largo del proyecto.

- El framework .NET es fundamental para realizar la conexión de la aplicación con TransCAD, por lo que siempre se debe tener en cuenta si se llega a emplear una tecnología diferente. En caso de cambiar de tecnología, se debe investigar cuáles son las opciones que existen para realizar la representación geográfica del patrón del flujo vehicular.

# Referencias bibliográficas


Handwerk, Brian. "Half of humanity will live in cities by year's end." National Geographic News, Marzo 2008. http://news.nationalgeographic.com/news/2008/03/080313-cities.html [Fecha de Consulta: 13 de Marzo 2010].

Houghton, Jamie; Reiners, John y Colin, Lim. "*Intelligent transport. How cities can improve mobility*". IBM Corporation, 2009.
http://www.electricdrive.org/index.php?ht=a/GetDocumentAction/id/27920
[Fecha de Consulta: 30 de junio 2010].

"*Caminos hacia un planeta más inteligente*". IBM Corporation, 2010.
http://www.ibm.com/smarterplanet/mx/es/traffic_congestion/visions/
[Fecha de Consulta: 30 de junio 2010].

Lozano, Angélica y Antún, Juan, P. "*Tráfico vehicular en zonas urbanas*". Instituto de Ingeniería de la UNAM. Ciencias núm. 70. 2003.

"*Traffic assignment manual for application with a large, high speed computer*". Washington, U.S. Department of Commerce Bureau of Public Roads Office of Planning Urban Planning Division, 1964.

Londoño, Gloria. "*Métodos de asignación dinámica de tráfico*". Tesis para obtener el grado de maestría de Transporte. Posgrado de Ingeniería de la UNAM. México, 2003.

*"Diccionario de la lengua española".* Vigésima segunda edición (versión en línea)
http://drae.rae.es [Fecha de Consulta: 15 de Agosto 2012].

*"Traffic engineering handbook".* Institute of Transportation Engineers. Quinta edición. Washington, D.C., 1999. pp. 1-3. Cal y Mayor Cárdenas. *"Ingeniería de tránsito".* Octava edición. México, 2007.

Bull, Alberto y Thomson, Ian. *"La congestión del tránsito urbano: causas y consecuencias económicas y sociales".* CEPAL, Chile, 2001.

*"Inventario de emisiones de contaminantes de la ZMVM"* Secretaria del Medio Ambiente del Gobierno del Distrito Federal. Primera edición. México, 2010.

*"Highway capacity manual".* Transportation Research Board. National Research Council, Washington, D.C., 2000.

Larraga, María, E. *"Modelación del tráfico vehicular, mediante autómatas celulares".* Gaceta del Instituto de Ingeniería, núm. 41, México, Mayo 2008. pp. 4.

Rosen, Kenneth. *"Discrete mathematics and its applications".* Cuarta edición, 1998.

Hayet, J. *"Caminos más cortos en grafos".* Centro de Investigación en Matemáticas, 2008.
http://www.cimat.mx/~jbhayet/CLASES/PROGRAMACIONII/clase10.pdf
[Fecha de Consulta: 15 de Agosto 2011].

Chapra, Steven, C. y Canale, Raymond, P. *"Numerical Methods for Engineers".* Quinta Edición, 2006.

Londoño, Gloria y Lozano, Angélica. *"Suitable cost functions for signalized arterials and freeways, in the user equilibrium assignment problem"*. Procedia Social and Behavioral Sciences. ISSN 1877-0428. Science Publications, Elsevier. In Press, 2012.

Grinderud, Knut. "*GIS: The geographic language of our age*". Akademika Publishing, 2009.

Bracken, Ian y Webster, Christopher. "*Information technology in geography and planning*". Routledge, 1990.

Aronoff, Stan. *"Geographic information systems: a management perspective"*. Wdl Pubns, 1989.

*"Core curriculum in GIS"*. National Center for Geographic Information and Analysis, 1990.

Burrough, P. y McDonnell, R. *"Principals of geographical information systems for land resource assessmen"*. Oxford University Press, 1998.

Longley, Paul; Goodchild, Michael y Maguire, David. *"Geographical information systems and Science"*. Wiley, 2005.

Harmon, John y Anderson, Steven. *"The design and implementation of geographic information systems"*. John Wiley & Sons, New Yorke, 2003.

Gutierrez, Javier. "*Sistemas de información geográfica: funcionalidades, aplicaciones y perspectivas en Mato Grosso do Sul*". Universidad Complutense de Madrid. Revista Internacional de Desenvolvimento Local. Num. 1, p. 41-48, 2000. http://www3.ucdb.br/mestrados/RevistaInteracoes/n1_puebla.pdf [Fecha de Consulta: 25 de Agosto 2011].

"*Spatial information clearinghouse for humanitarian mine action*". Spatial Information Clearinghouse, 2004. http://maic.jmu.edu/sic/home.htm [Fecha de Consulta: 29 de Octubre 2011].

Sanchez, Alejandra. "*Normas sobre metadatos (IS019115, ISO19115-2, ISO19139, ISO 15836)*". Instituto Geografico Nacional. España, 2008. http://iaaa.cps.unizar.es/curriculum/08-Publicaciones-Articulos/art_2008_Mapping_Normas.pdf [Fecha de Consulta: 30 de Agosto 2011].

"GPS Review". 2011. www.geekzone.com.ar/tags/navteq [Fecha de Consulta: 5 de Octubre 2011].

Domech, Annando, y Mosqueras, Carmen. *"Análisis crítico del estado actual de la modelación cartográfica",* 1995. http://www.mappinginteractivo.com/plantilla-ante.asp?id_articulo=1319 [Fecha de Consulta: 25 de Agosto 2011].

"*Datos geográficos y métodos de almacenamiento*". CIAF. Bogotá, Colombia, 2010. http://corponarino.gov.co/pmapper-4.1.1/sig/interfase/documentos/datos_geograficos.pdf [Fecha de Consulta: 2 de Diciembre 2011].

Steiniger, Stefan y Weibel, Robert. "*GIS Software - A description in 1000 words*". University of Calgary, University of Zurich, 2009.

"TransCAD Overview". Caliper, 2010. http://www.caliper.com/tcovu.htm [Fecha de Consulta: 22 de Diciembre 2011].

Larman, Craig. *"UML y Patrones".* Segunda edición. Madrid, España, 2003.

# Anexo 1

A continuación se presentan las figuras más representativas de la implementación del algoritmo Frank Wolfe para las funciones de tiempo de viaje utilizadas: BPR, BPR-Webster y Akcelik-Webster.

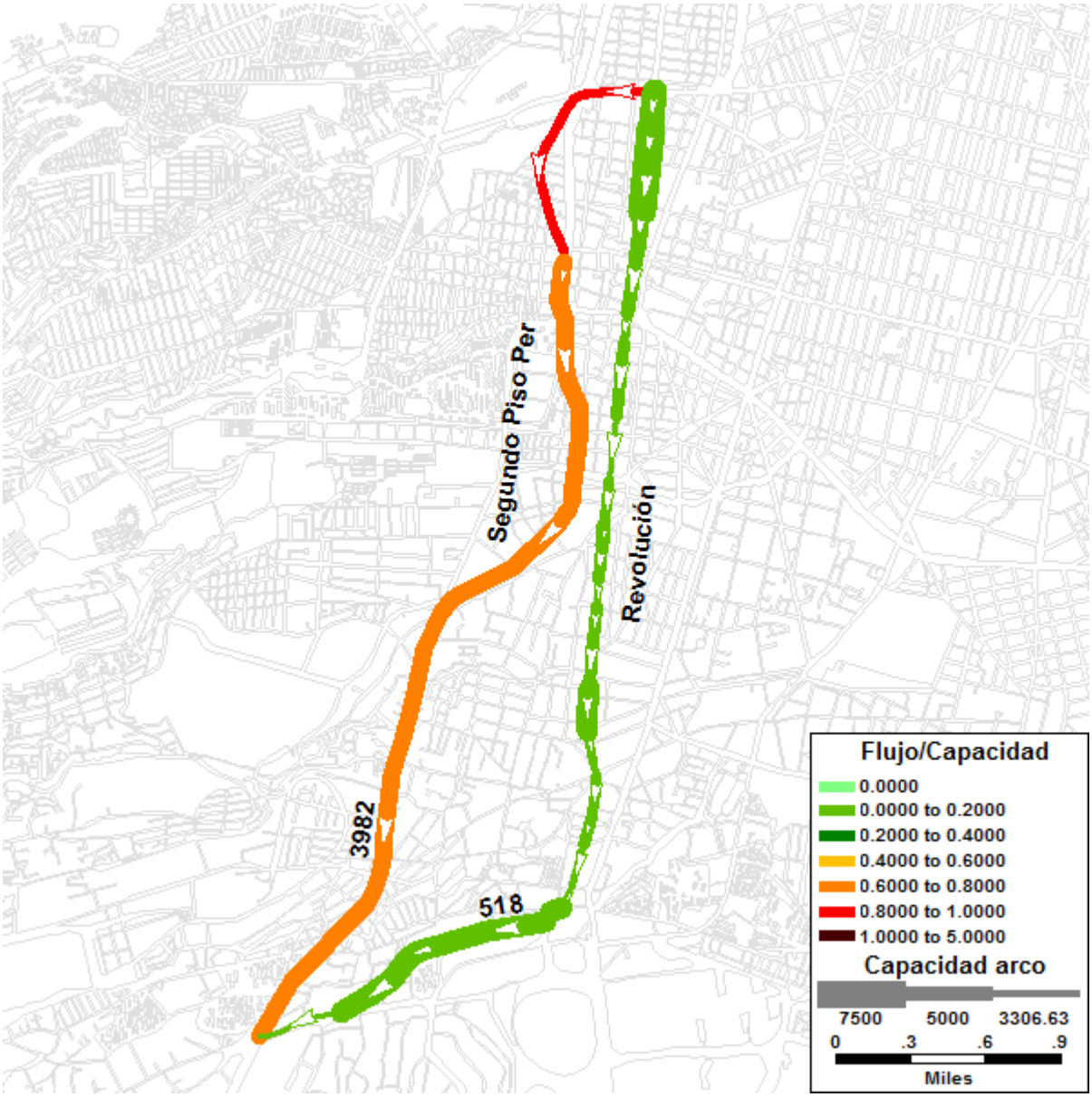



**Figura 7.1** *Comportamiento de la red vial: demanda de 4500 [veh/h] - función BPR*
(Elaboración propia)

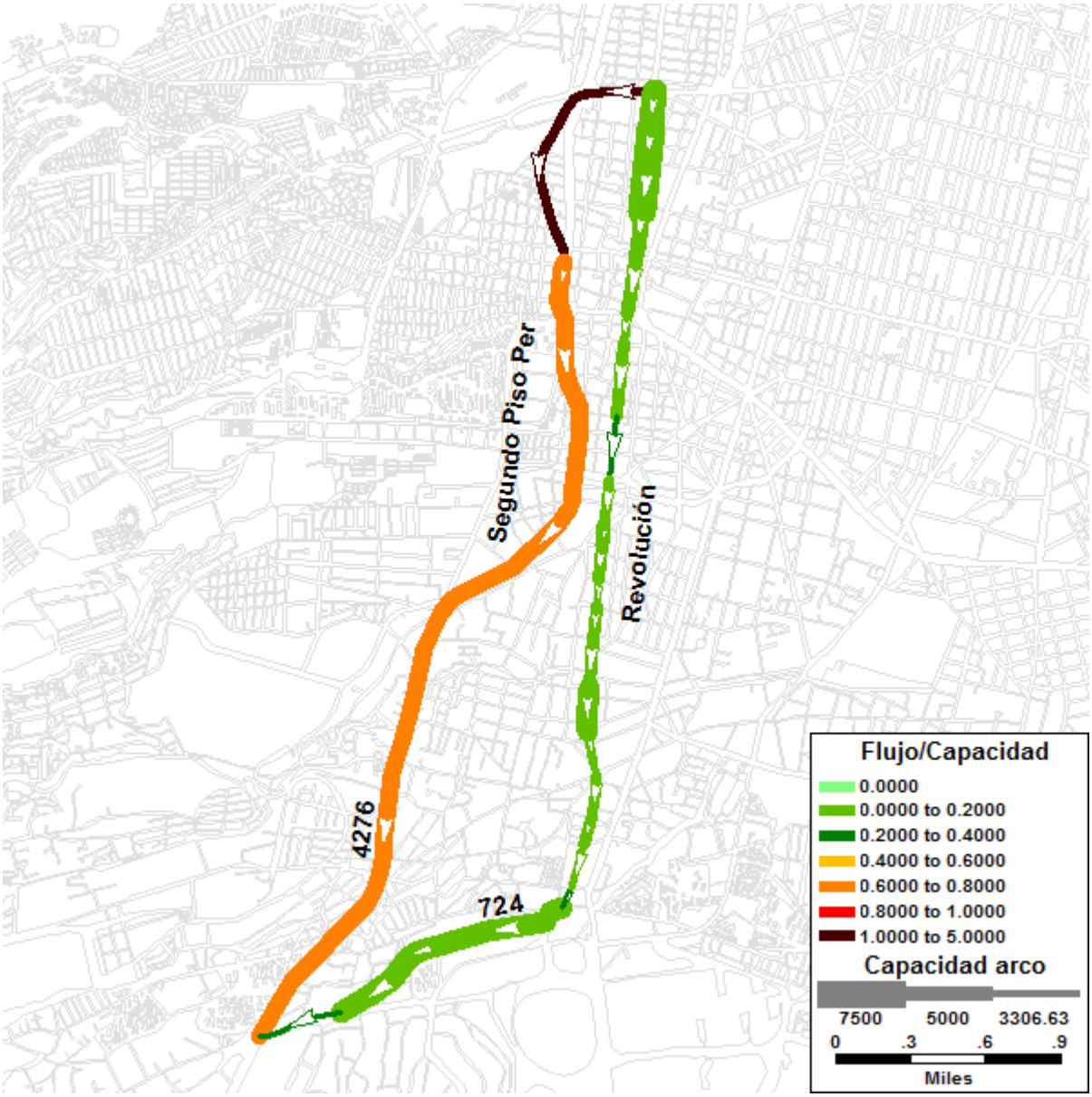



**Figura 7.2** *Comportamiento de la red vial: demanda de 5000 [veh/h] - función BPR*
(Elaboración propia)

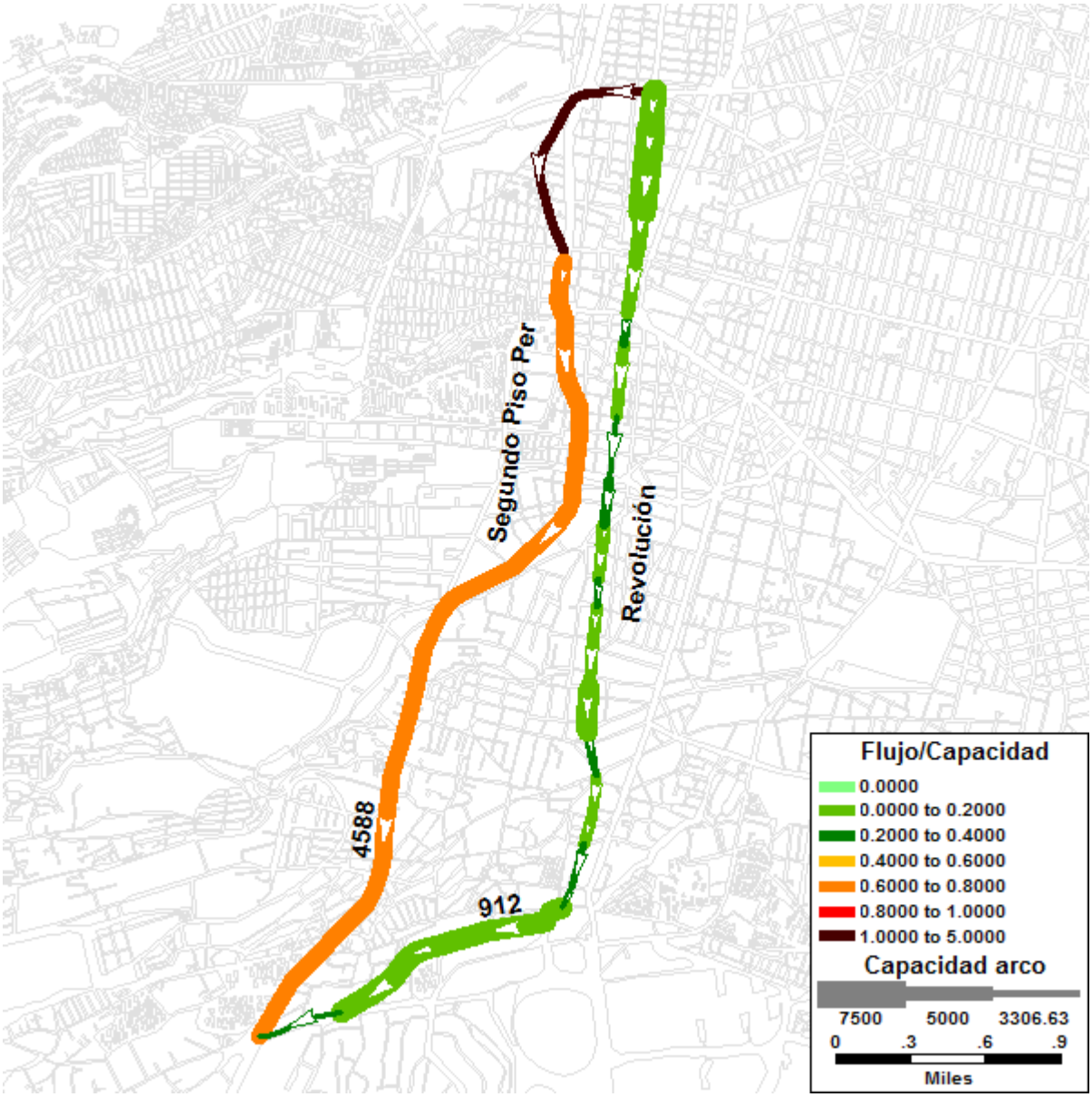



**Figura 7.3** *Comportamiento de la red vial: demanda de 5500 [veh/h] - función BPR*
(Elaboración propia)

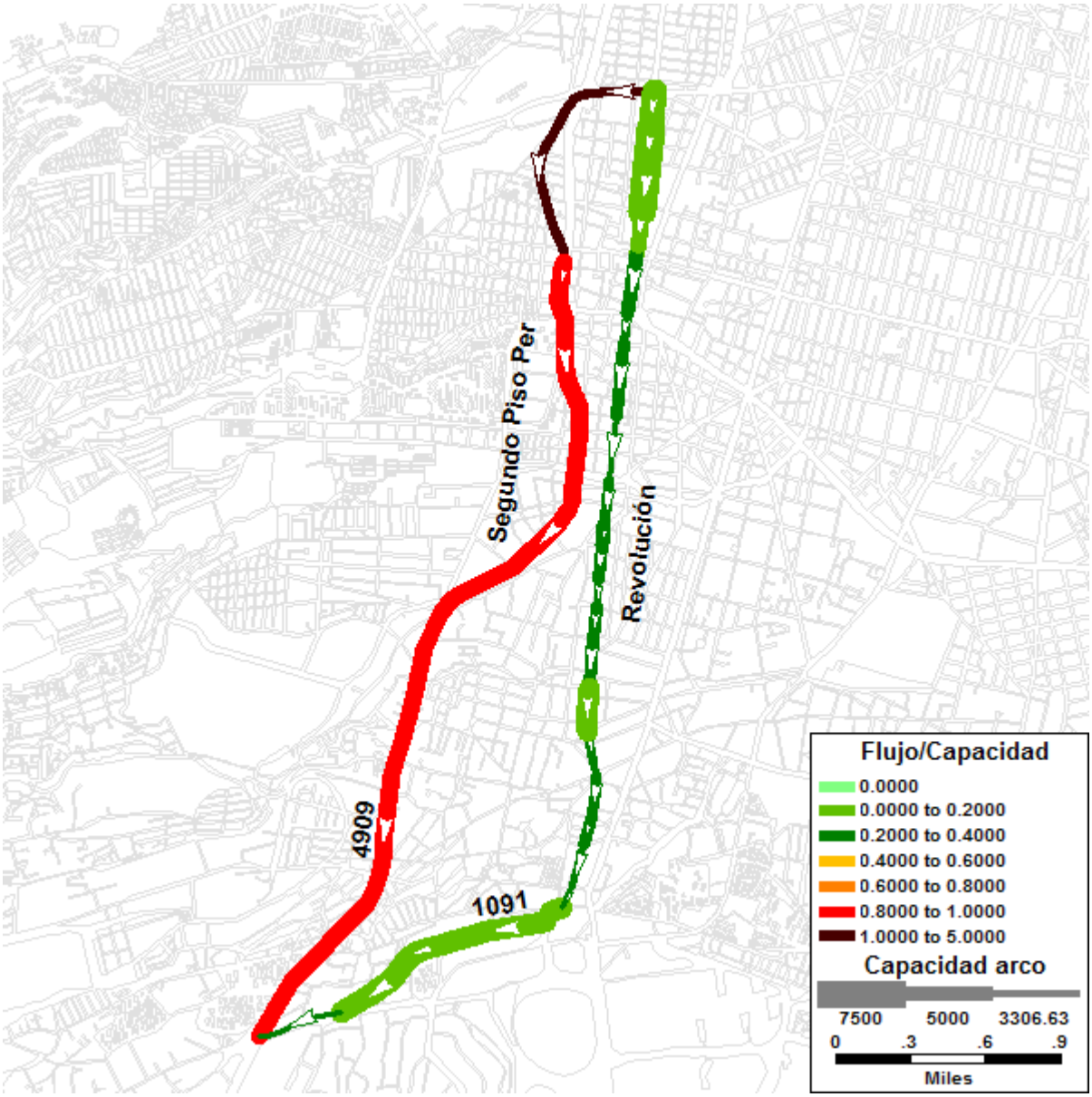



**Figura 7.4** *Comportamiento de la red vial: demanda de 6000 [veh/h] - función BPR*
(Elaboración propia)

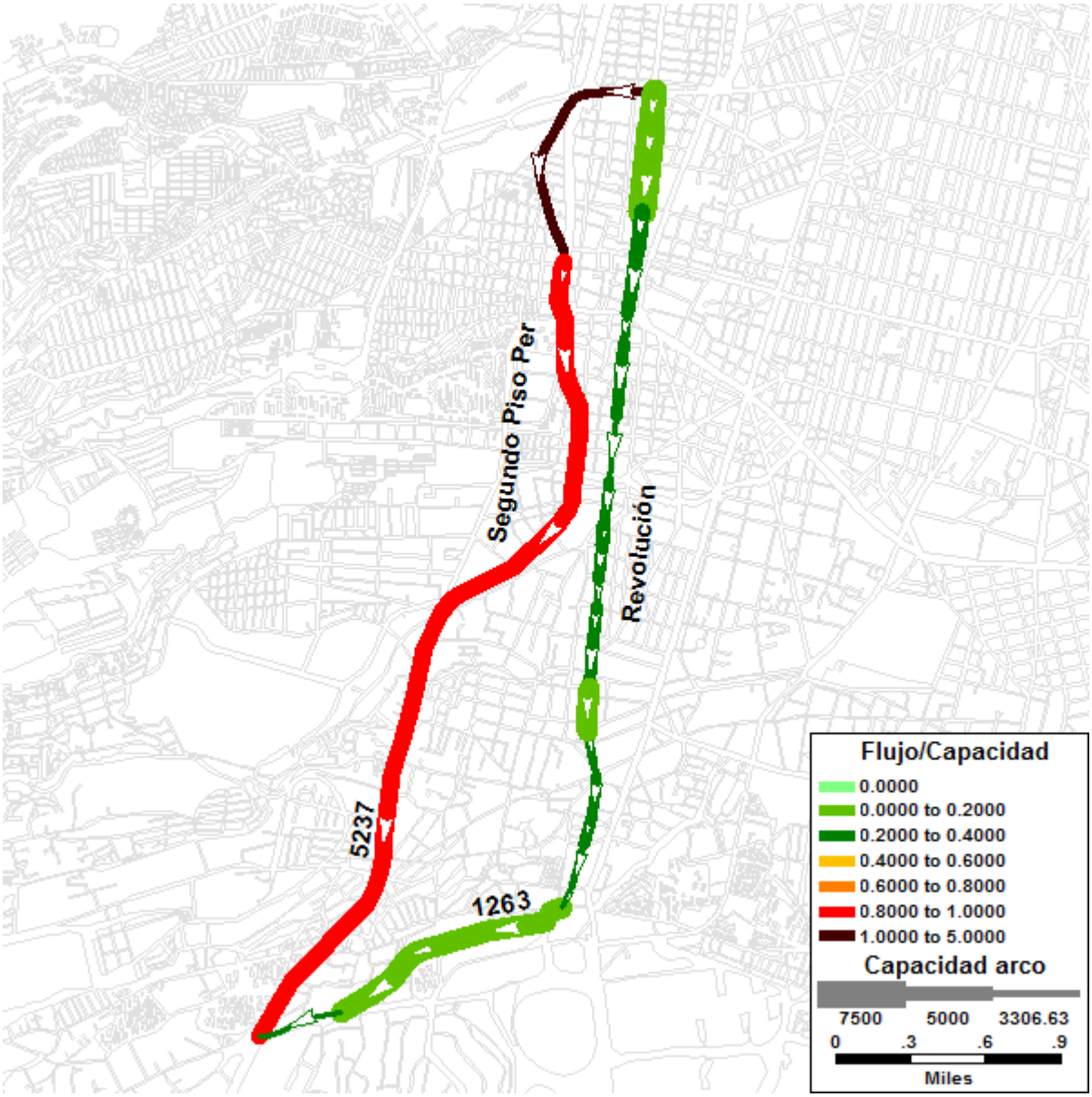



**Figura 7.5** *Comportamiento de la red vial: demanda de 6500 [veh/h] - función BPR*
(Elaboración propia)

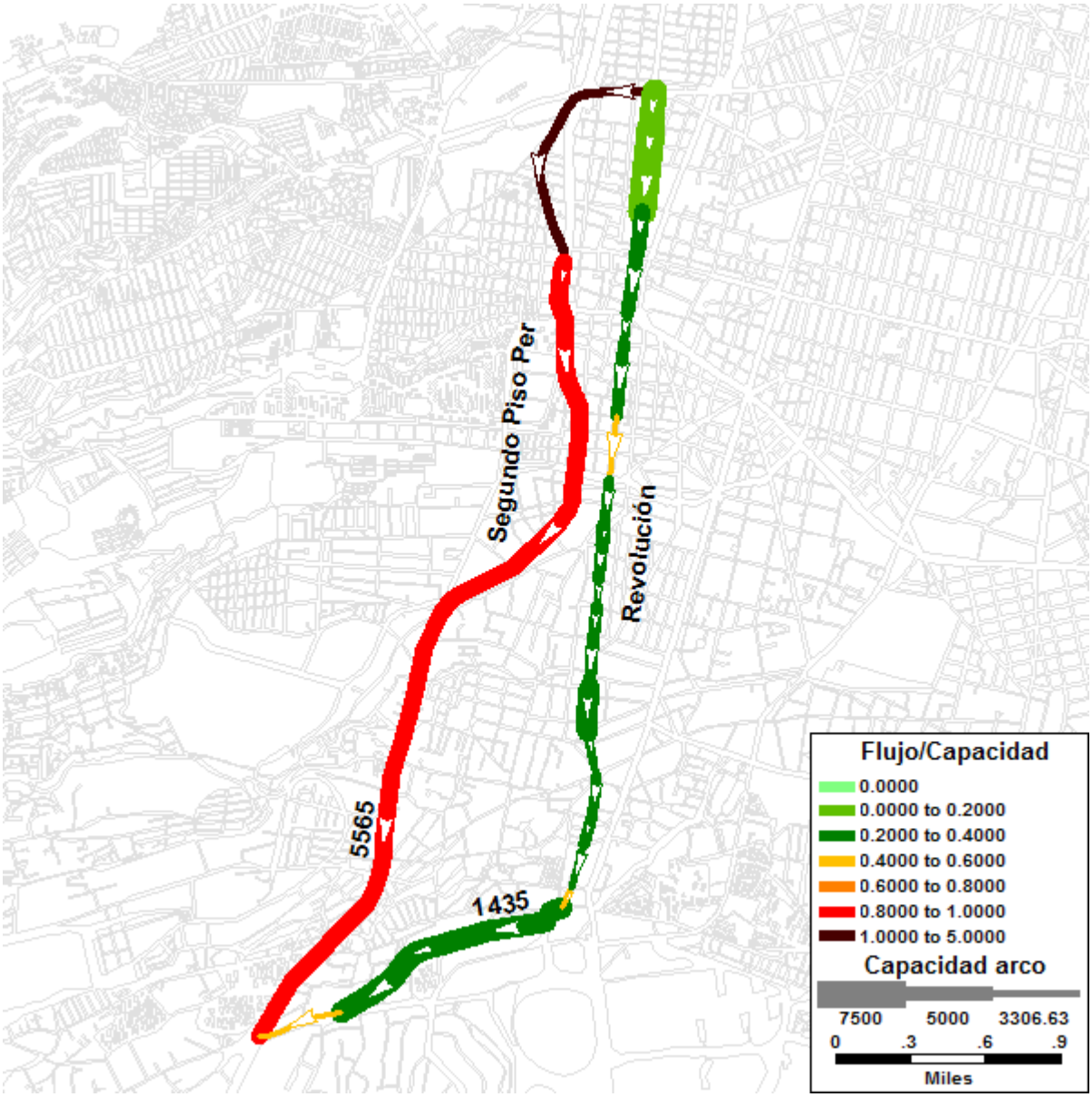



**Figura 7.6** *Comportamiento de la red vial: demanda de 7000 [veh/h] - función BPR*
(Elaboración propia)

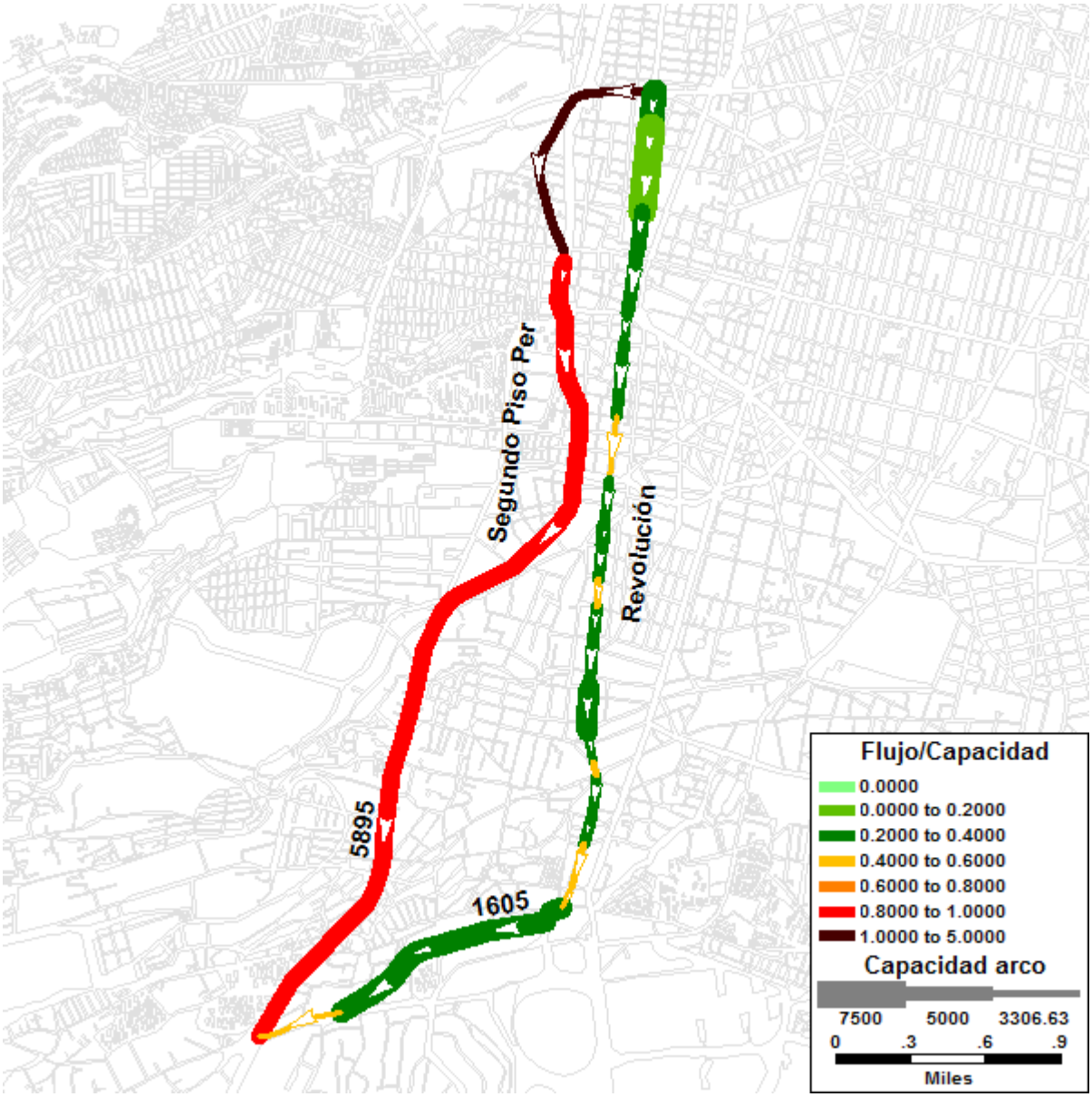



**Figura 7.7** *Comportamiento de la red vial: demanda de 7500 [veh/h] - función BPR*
(Elaboración propia)

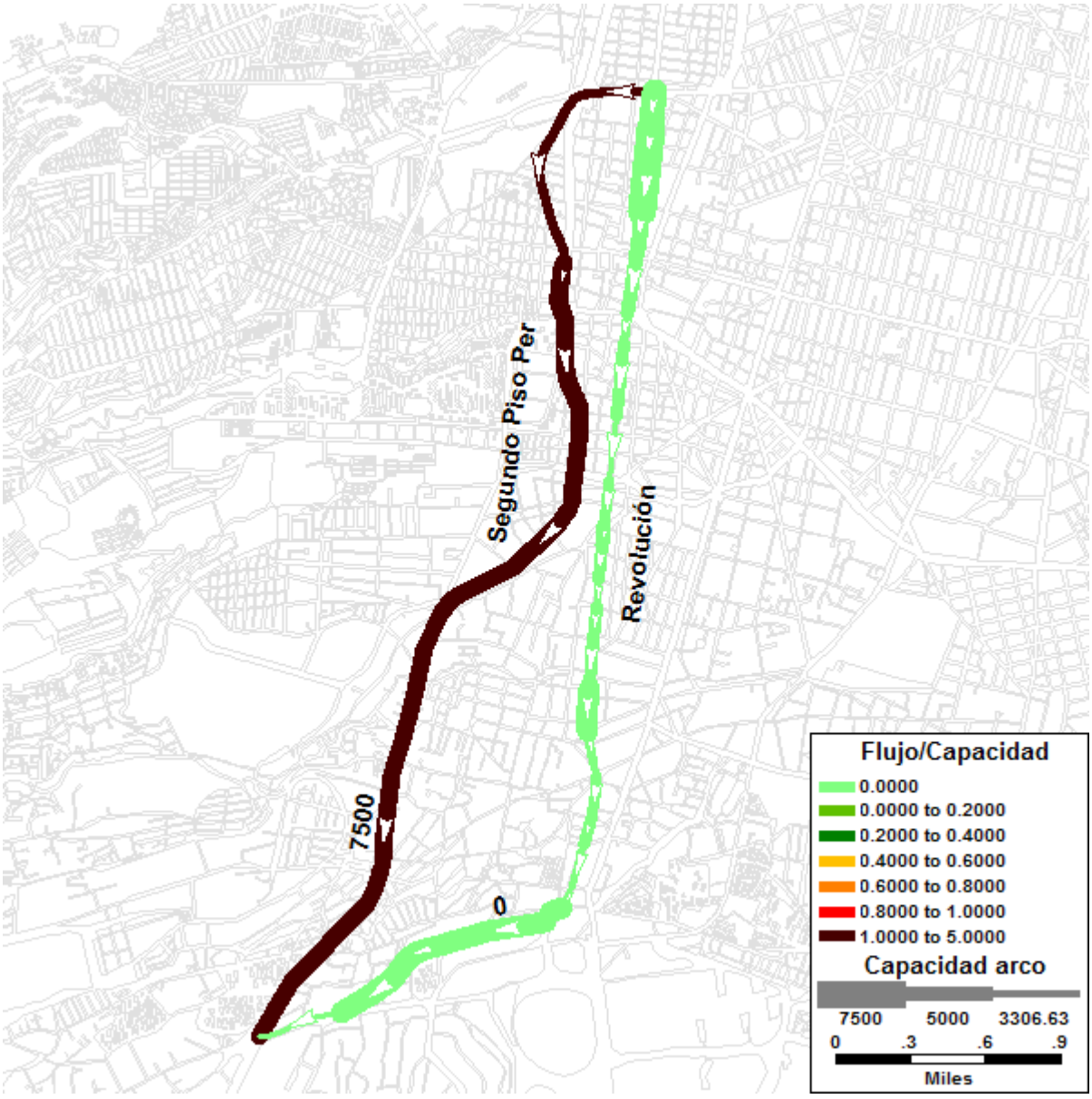



**Figura 7.8** *Comportamiento de la red vial: demanda de 7500 [veh/h] - función BPR-Webster*
(Elaboración propia)

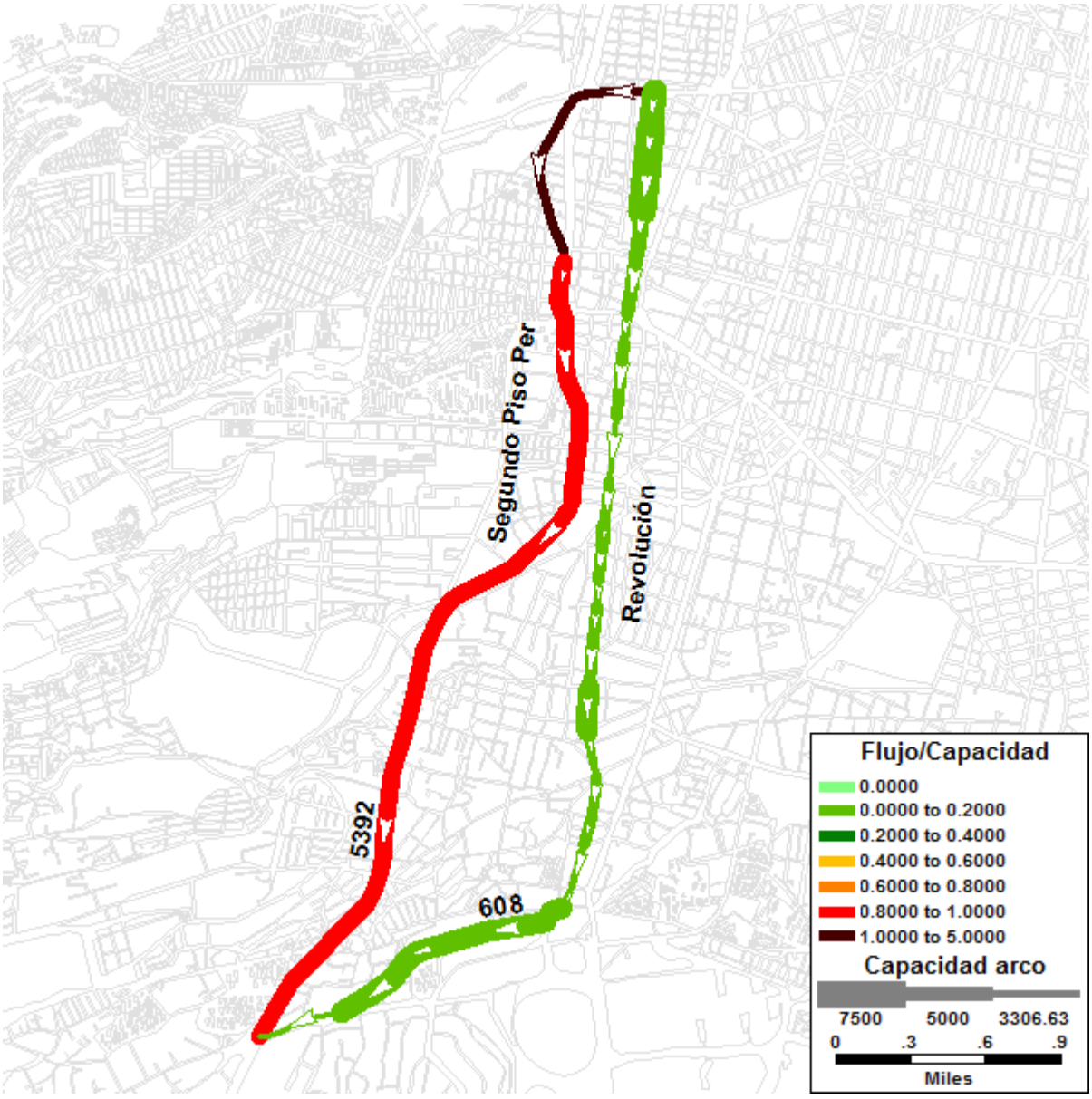



**Figura 7.9** *Comportamiento de la red vial: demanda de 6000 [veh/h] - función Akcelik-Webster*
(Elaboración propia)

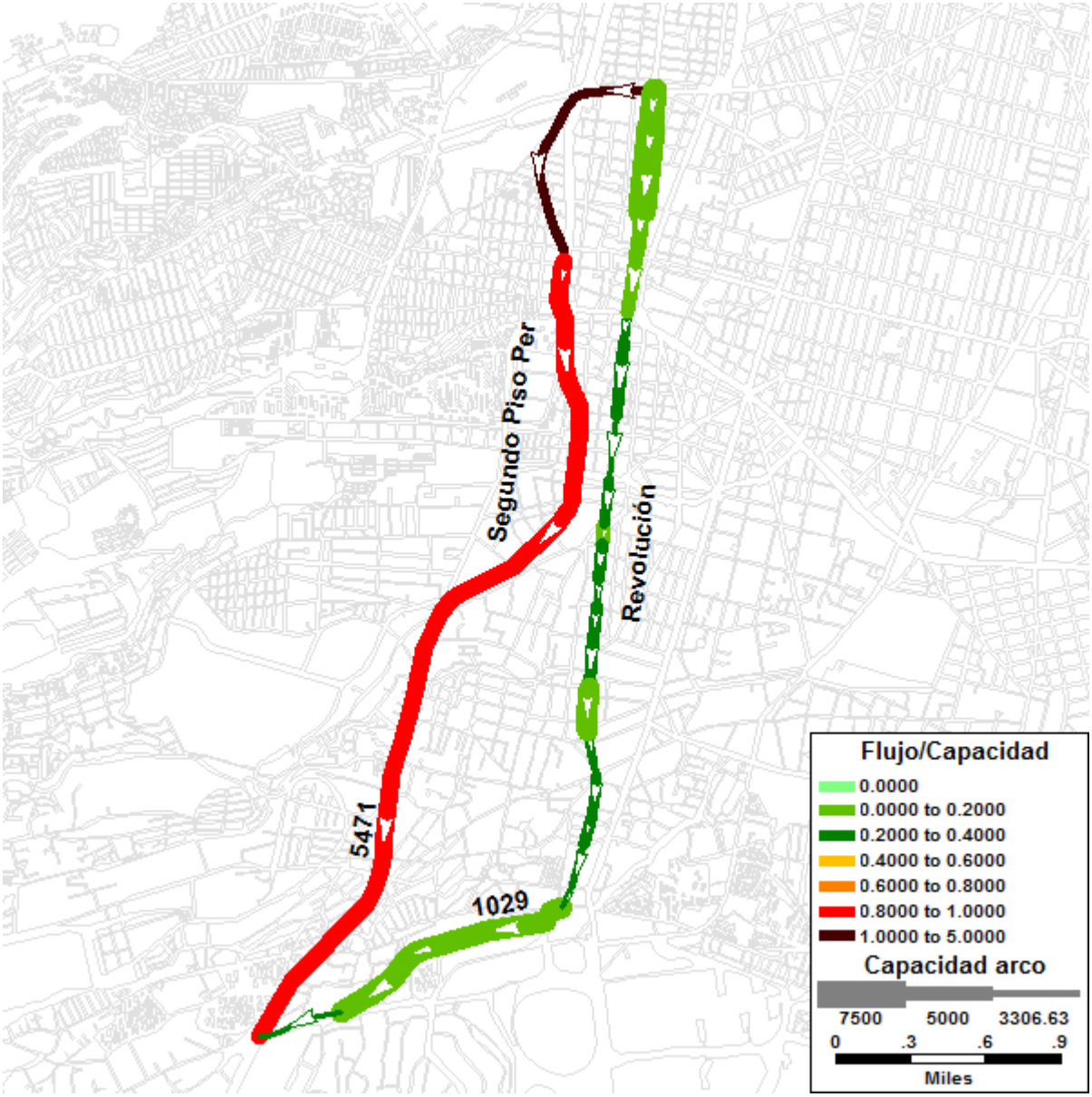



**Figura 7.10** *Comportamiento de la red vial: demanda de 6500 [veh/h] - función Akcelik-Webster*
(Elaboración propia)

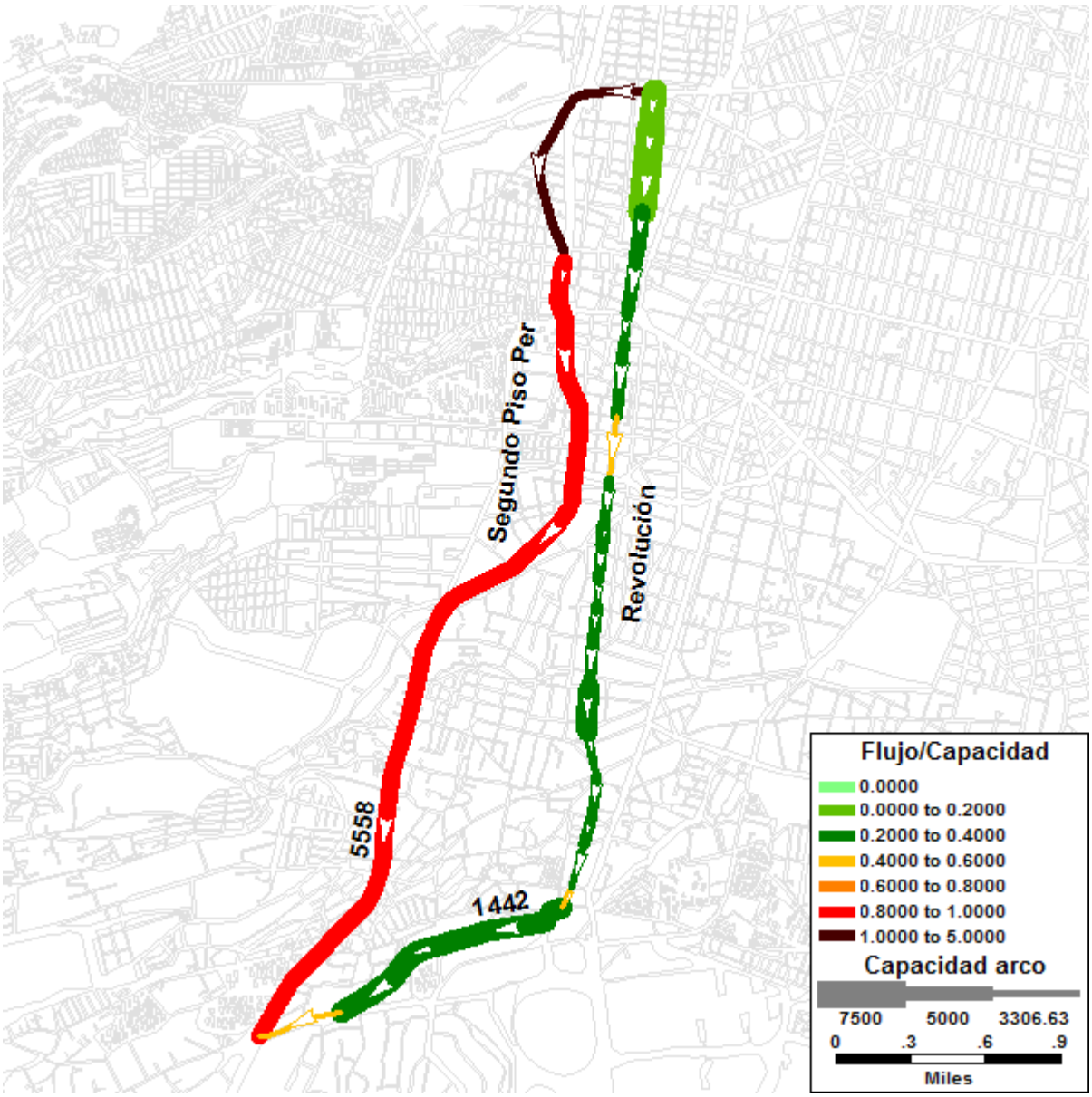



**Figura 7.11** *Comportamiento de la red vial: demanda de 7000 [veh/h] - función Akcelik-Webster*
(Elaboración propia)

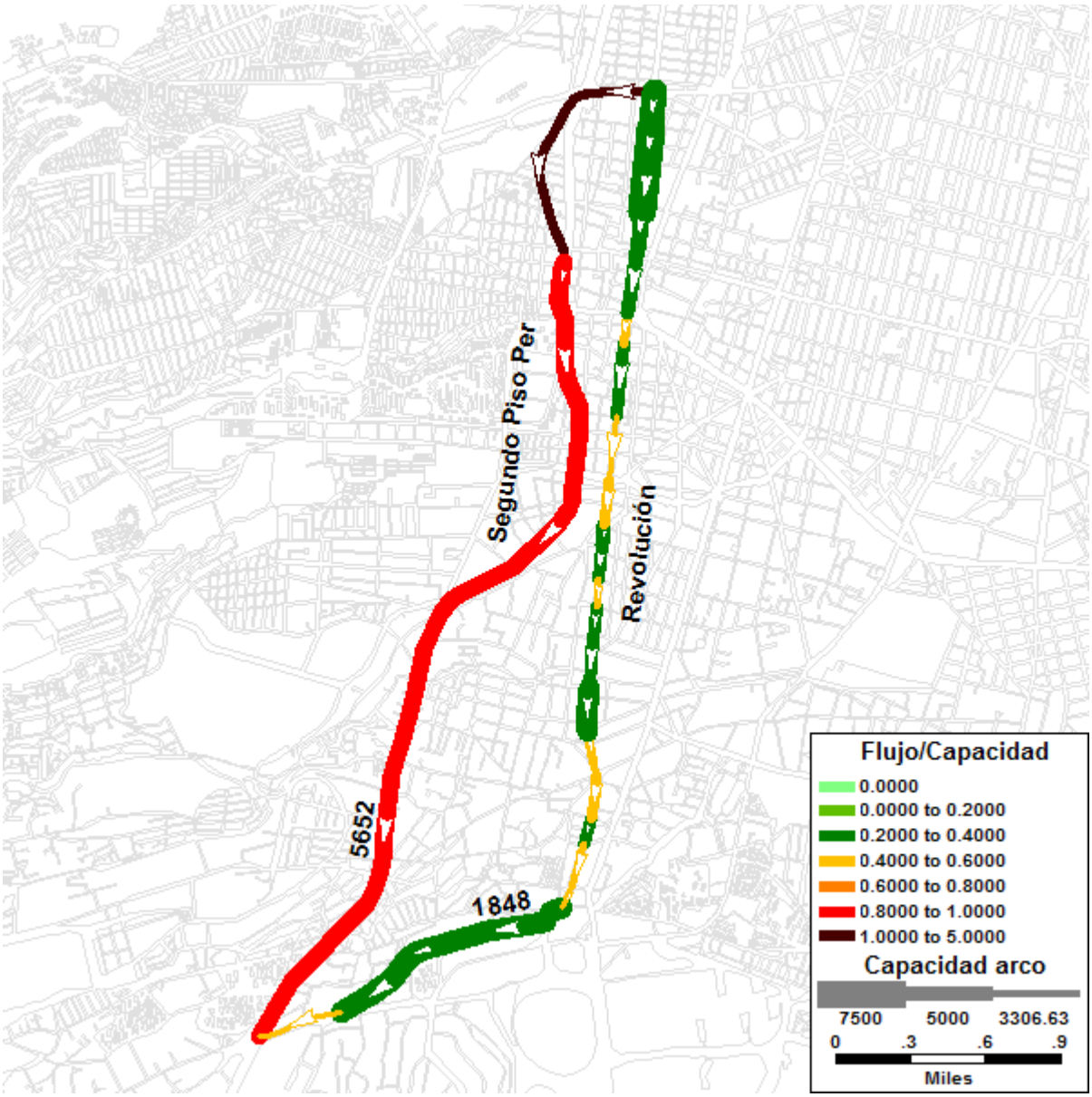



**Figura 7.12** *Comportamiento de la red vial: demanda de 7500 [veh/h] - función Akcelik-Webster*
(Elaboración propia)

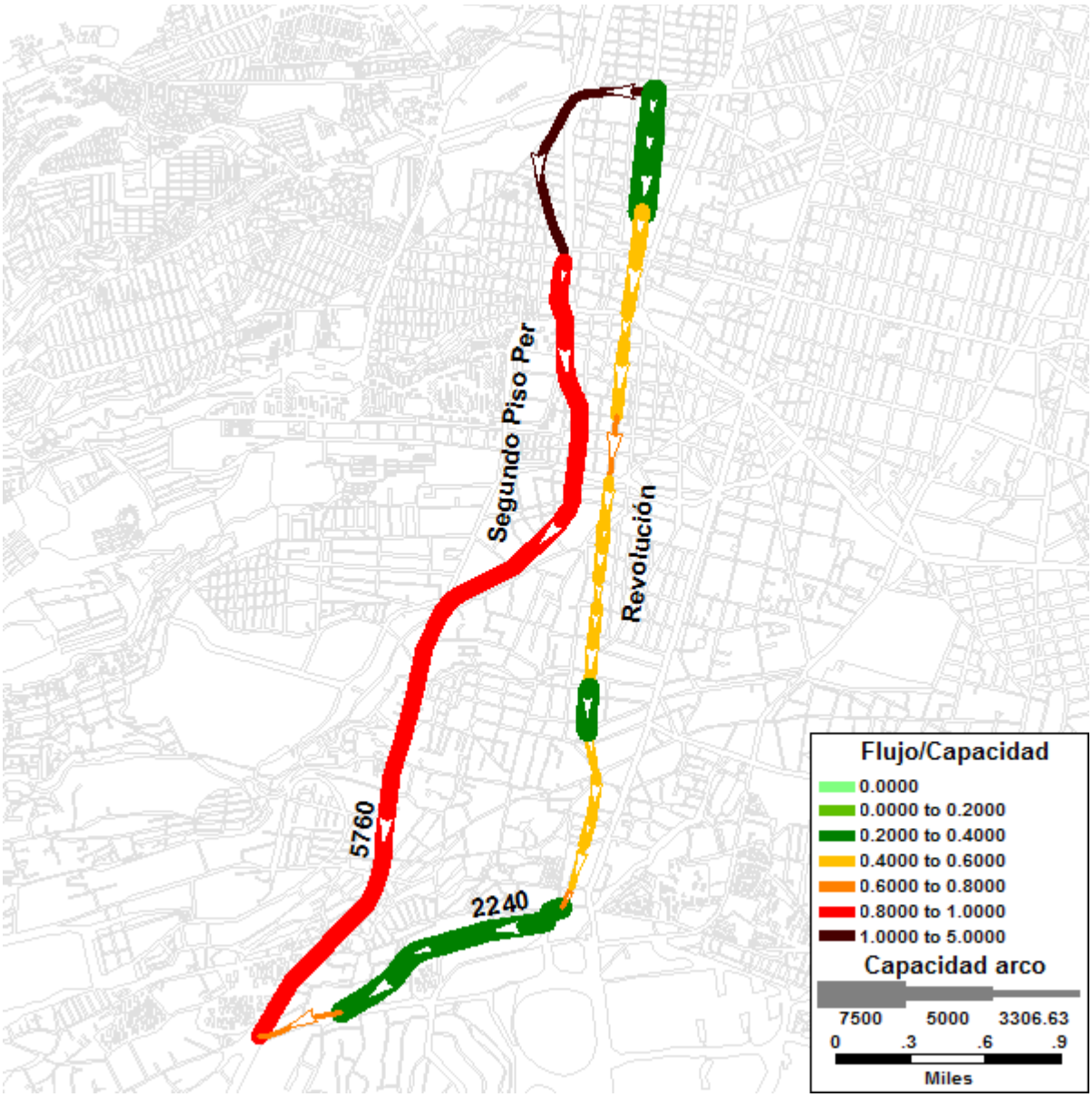



**Figura 7.13** *Comportamiento de la red vial: demanda de 8000 [veh/h] - función Akcelik-Webster*
(Elaboración propia)

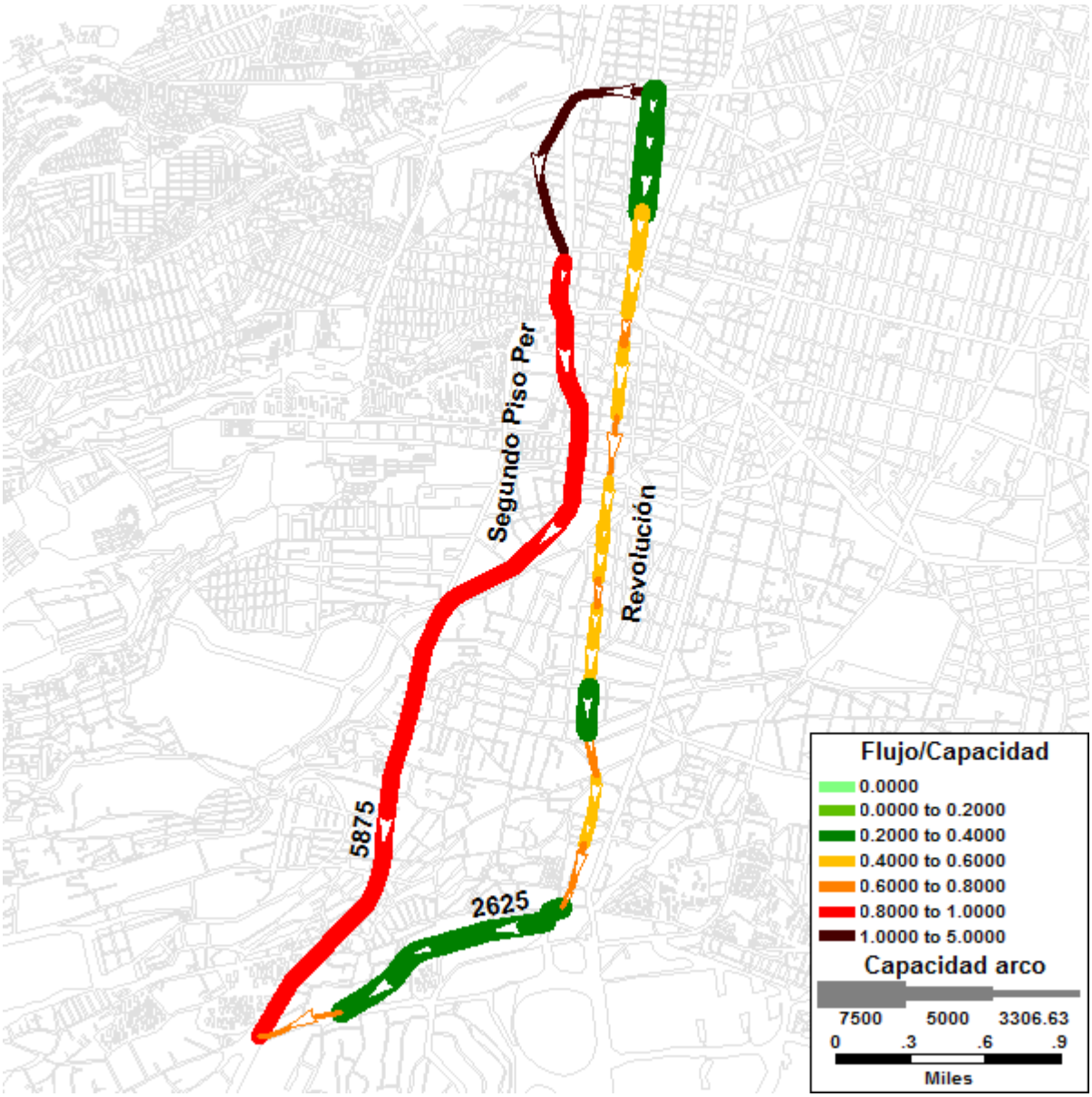



**Figura 7.14** *Comportamiento de la red vial: demanda de 8500 [veh/h] - función Akcelik-Webster*
(Elaboración propia)

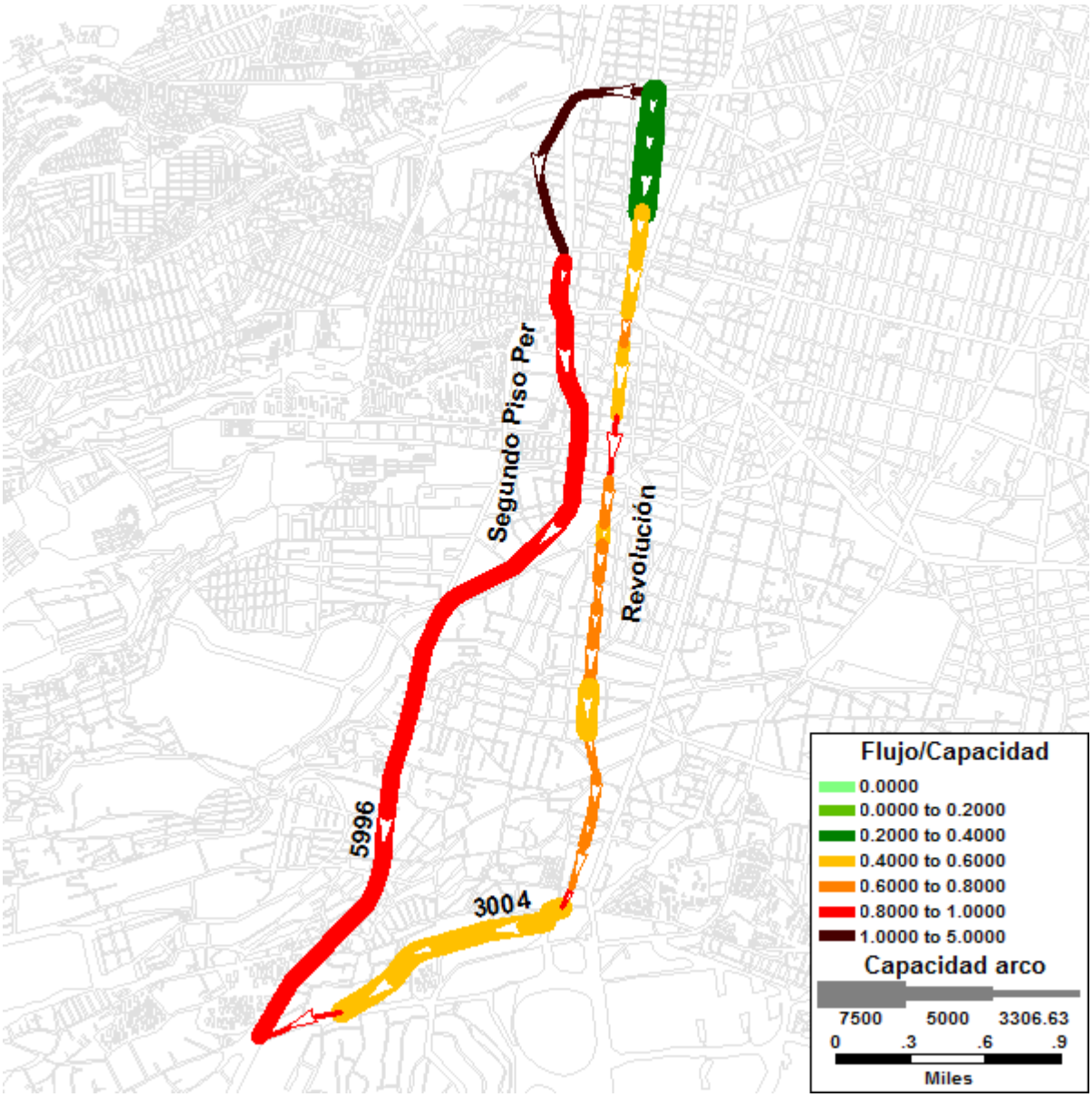
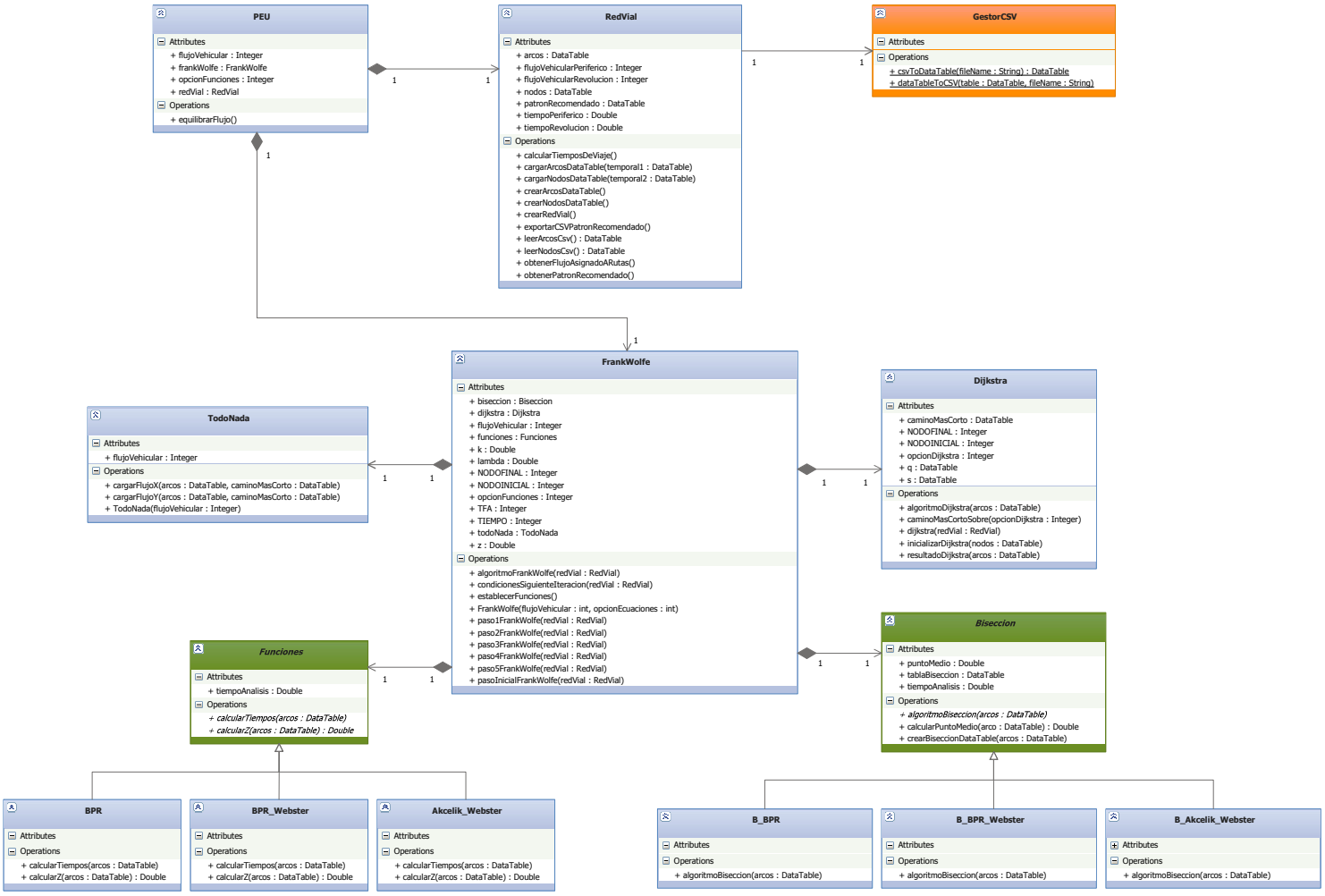



**Figura 7.15** *Comportamiento de la red vial: demanda de 9000 [veh/h] - función Akcelik-Webster*
(Elaboración propia)

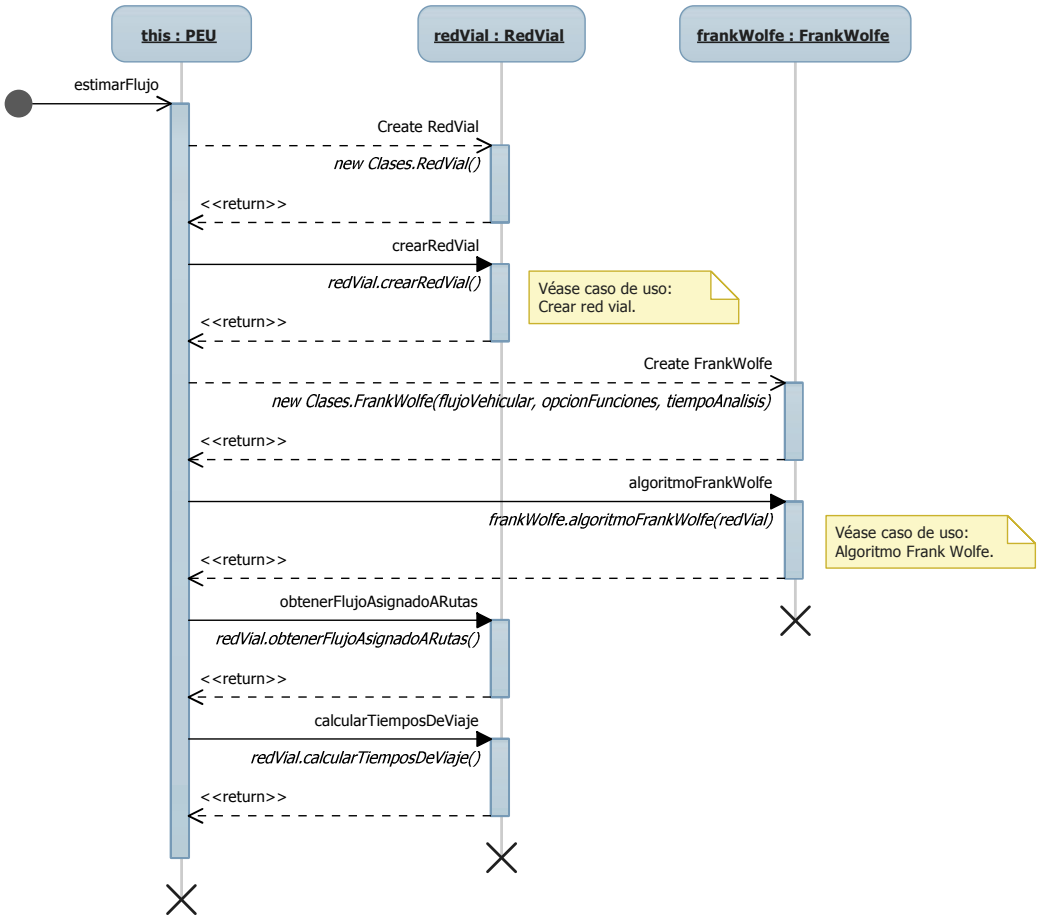
# Anexo 2

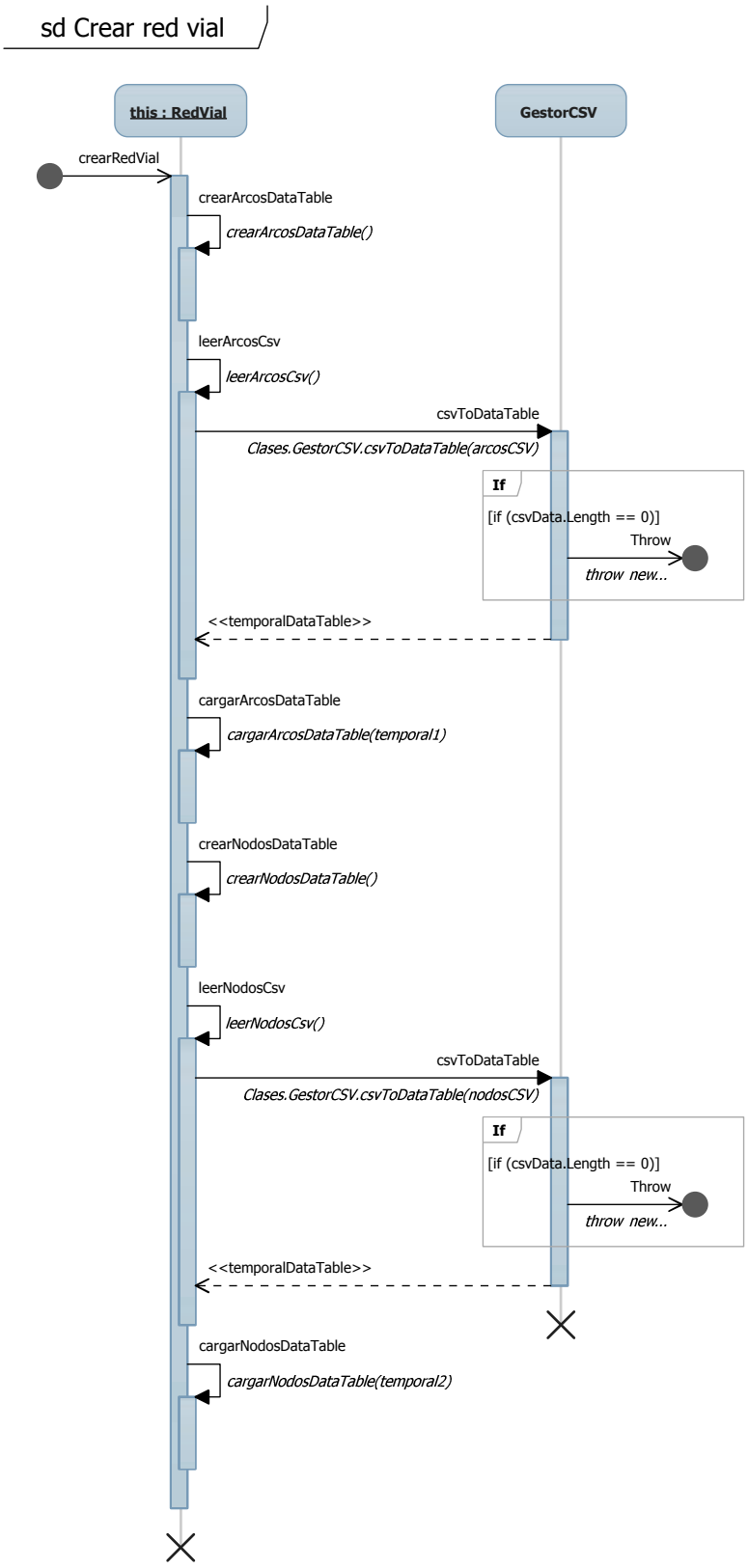
En esta sección se provee el diagrama de clases y de secuencia que conforman el diseño de la implementación que detalla el algoritmo que da solución al problema de equilibrio del usuario.

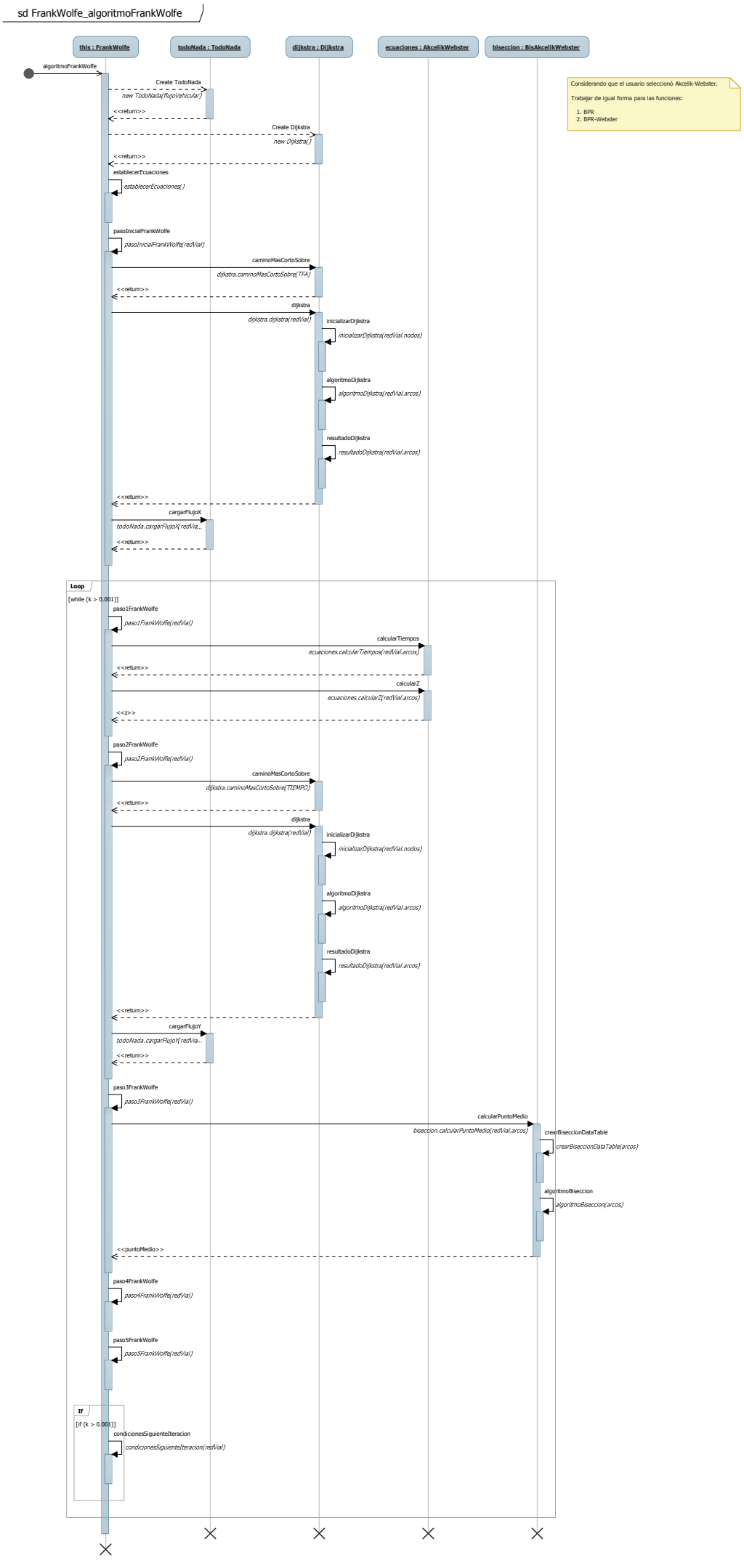
cd Algoritmo PEU



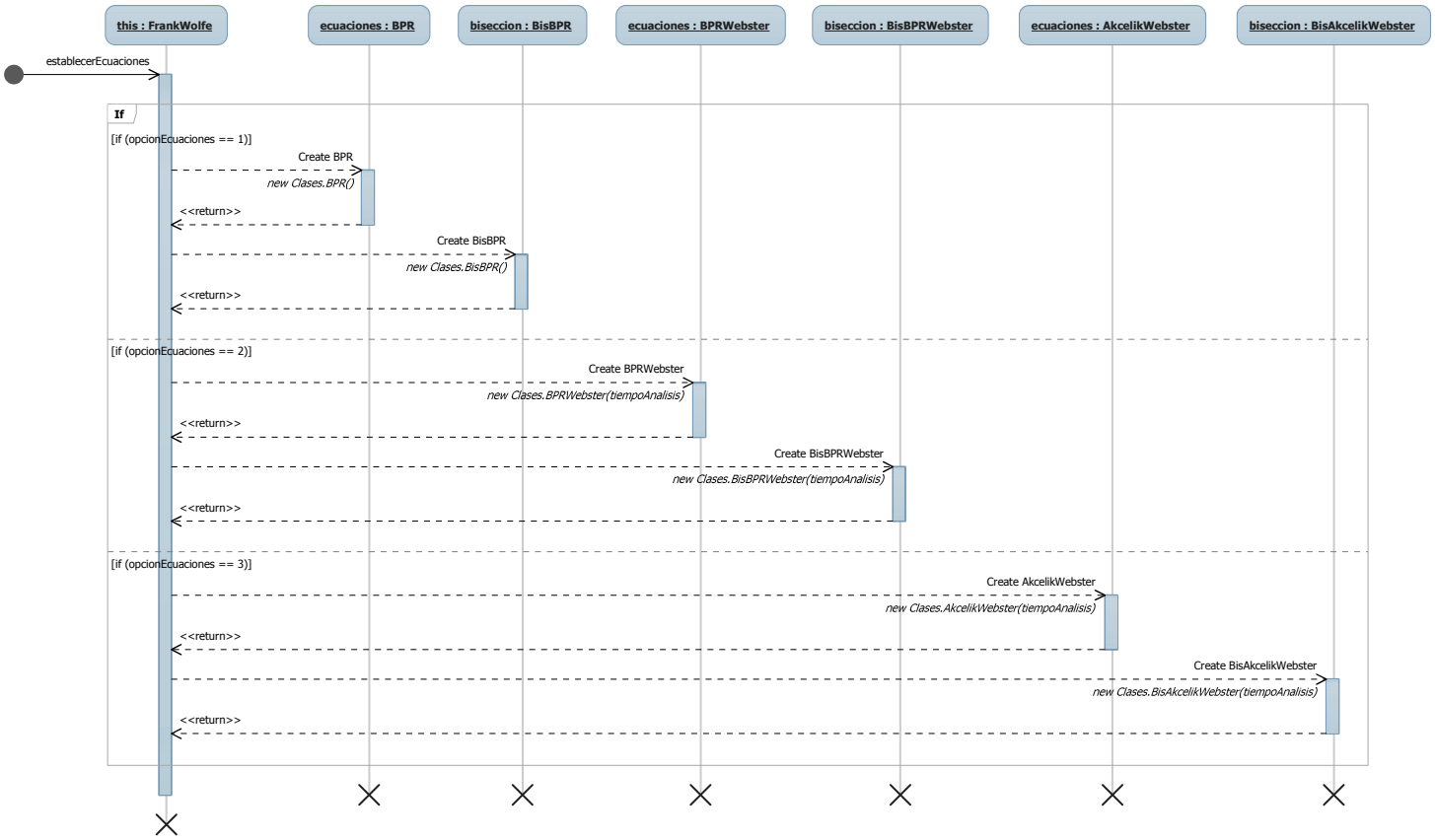
sd Estimar flujo vehicular



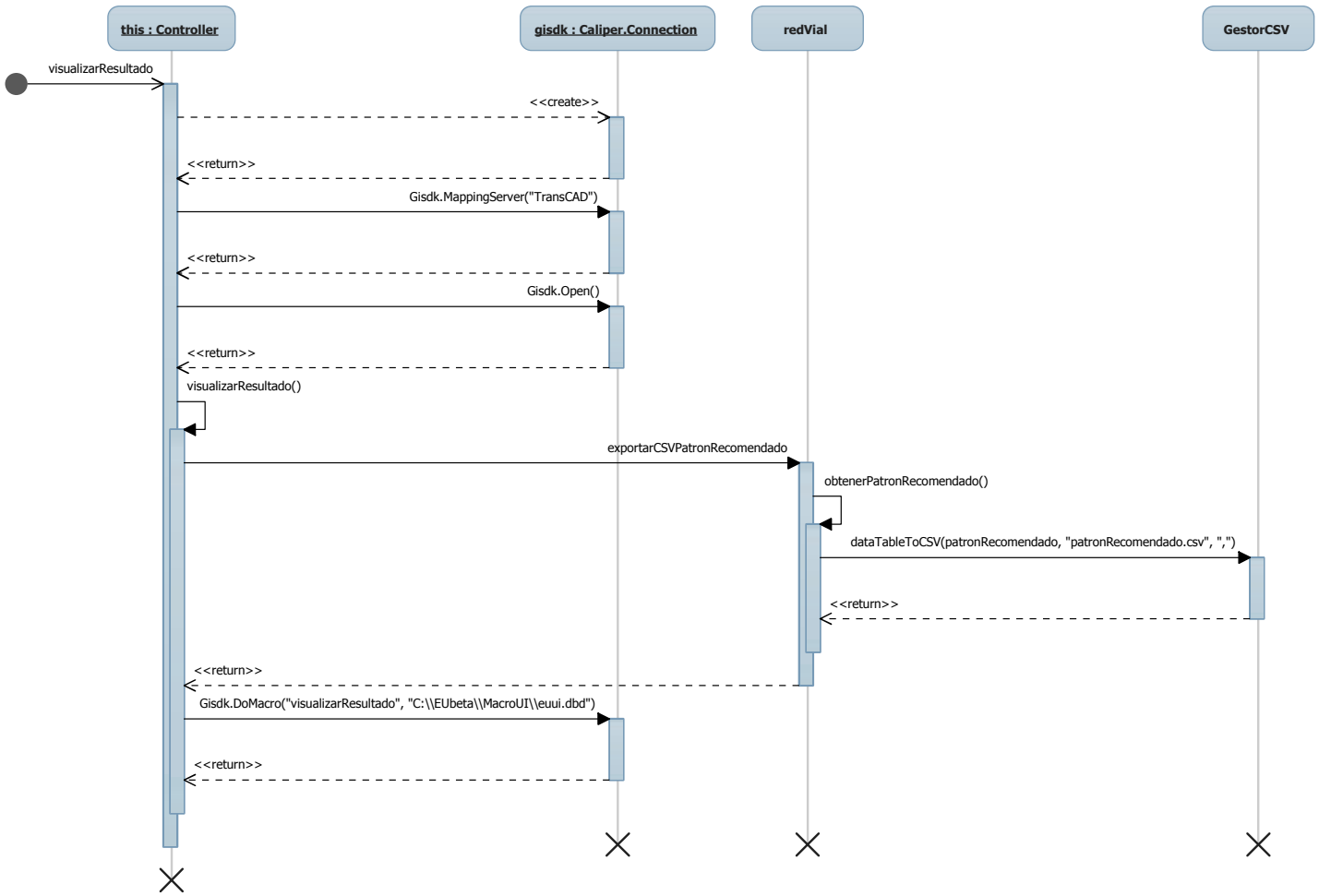

sd Crear red vial
this : RedVial
GestorCSV
crearRedVial
crearArcosDataTable
crearArcosDataTable()
leerArcosCsv
leerArcosCsv()
csvToDataTable
Clases.GestorCSV.csvToDataTable(arcosCSV)
If
[if (csvData.Length == 0)]
Throw
throw new..
<<temporalDataTable>>
cargarArcosDataTable
cargarArcosDataTable(temporal1)
crearNodosDataTable
crearNodosDataTable()
leerNodosCsv
leerNodosCsv()
csvToDataTable
Clases.GestorCSV.csvToDataTable(nodosCSV)
If
[if (csvData.Length == 0)]
Throw
throw new..
<<temporalDataTable>>
cargarNodosDataTable
cargarNodosDataTable(temporal2)


sd FrankWolfe_algoritmoFrankWolfe
this : FrankWolfe
todoNada : TodoNada
dijkstra : Dijkstra
ecuaciones : AkcelikWebster
biseccion : BisAkcelikWebster
algoritmoFrankWolfe
Create TodoNada
new TodoNada(flujoVehicular)
<<return>>
Create Dijkstra
new Dijkstra()
<<return>>
establecerEcuaciones
establecerEcuaciones()
pasoInicialFrankWolfe
pasoInicialFrankWolfe(redVial)
caminoMasCortoSobre
dijkstra.caminoMasCortoSobre(TPA)
<<return>>
dijkstra
dijkstra.dijkstra(redVial)
inicializarDijkstra
inicializarDijkstra(redVial.nodos)
algoritmoDijkstra
algoritmoDijkstra(redVial.arcos)
resultadoDijkstra
resultadoDijkstra(redVial.arcos)
<<return>>
cargarFlujoX
todoNada.cargarFlujoX(redVia...
<<return>>
Considerando que el usuario seleccionó Akcelik-Webster.
Trabajar de igual forma para las funciones:
1. BPR
2. BPR-Webster
Loop
[while (k > 0.001)]
paso1FrankWolfe
paso1FrankWolfe(redVial)
calcularTiempos
ecuaciones.calcularTiempos(redVial.arcos)
<<return>>
calcularZ
ecuaciones.calcularZ(redVial.arcos)
<<z>>
paso2FrankWolfe
paso2FrankWolfe(redVial)
caminoMasCortoSobre
dijkstra.caminoMasCortoSobre(TIEMPO)
<<return>>
dijkstra
dijkstra.dijkstra(redVial)
inicializarDijkstra
inicializarDijkstra(redVial.nodos)
algoritmoDijkstra
algoritmoDijkstra(redVial.arcos)
resultadoDijkstra
resultadoDijkstra(redVial.arcos)
<<return>>
cargarFlujoY
todoNada.cargarFlujoY(redVia...
<<return>>
paso3FrankWolfe
paso3FrankWolfe(redVial)
calcularPuntoMedio
biseccion.calcularPuntoMedio(redVial.arcos)
crearBiseccionDataTable
crearBiseccionDataTable(arcos)
algoritmoBiseccion
algoritmoBiseccion(arcos)
<<puntoMedio>>
paso4FrankWolfe
paso4FrankWolfe(redVial)
paso5FrankWolfe
paso5FrankWolfe(redVial)
If
[if (k > 0.001)]
condicionesSiguienteIteracion
condicionesSiguienteIteracion(redVial)

sd Establecer ecuaciones



sd Visualizar resultado



# Anexo 3

En el presente anexo se proporcionan las clases que conforman el núcleo central de la arquitectura que permitió implementar los diversos requerimientos a través de la serie de iteraciones efectuadas.

## PEU.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
   public class PEU
   {
      public int flujoVehicular;
      public String funcionesTiempoViaje;
      public RedVial redVial;
      public FrankWolfe frankWolfe;

      public ModeloEU()
      {
      }

      public void asignarFlujoEU()
      {
         // Red vial
         redVial = new RedVial();
         redVial.crearRedVial();

         // Frank Wolfe
         frankWolfe = new FrankWolfe(flujoVehicular, funcionesTiempoViaje);
         frankWolfe.algoritmoFrankWolfe(redVial);

         // Operaciones para presentar resultados
         redVial.obtenerFlujoAsignadoARutas();
         redVial.calcularTiemposDeViaje();
         redVial.calcularFlujoEntreCapacidadArco();
      }
    }
}
```

## FrankWolfe.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
   public class FrankWolfe
   {
      public int flujoVehicular;
      public String funcionesTiempoViaje;
      public double z;
      public double k;
      public double lambda;
      public TodoNada todoNada;
      public Dijkstra dijkstra;
      public Funciones funciones;
      public Biseccion biseccion;

      public FrankWolfe() {
      }

      public FrankWolfe(int flujoVehicular, String funcionesTiempoViaje)
      {
         this.flujoVehicular = flujoVehicular;
         this.funcionesTiempoViaje = funcionesTiempoViaje;
      }

      public void establecerFunciones()
      {
         if (funcionesTiempoViaje.Equals("BPR"))
         {
            funciones = new BPR();
            biseccion = new B_BPR();
         }
         else if (funcionesTiempoViaje.Equals("BPR-Webster"))
         {
            funciones = new BPR_Webster();
            biseccion = new B_BPR_Webster();
         }
         else if (funcionesTiempoViaje.Equals("Akcelik-Webster"))
         {
            funciones = new Akcelik_Webster();
            biseccion = new B_Akcelik_Webster();
         }
      }

      public void algoritmoFrankWolfe(RedVial redVial)
      {
           // Creacion de los objetos FrankWolfe
           todoNada = new TodoNada(flujoVehicular);
           dijkstra = new Dijkstra();
           establecerFunciones();
```

```
    // ALGORITMO FRANK WOLFE

    // Paso inicial
    pasoInicialFrankWolfe(redVial);

    k = 1;
    while (k > 0.001)
    {
        //  Paso 1
        paso1FrankWolfe(redVial);

        //  Paso 2
        paso2FrankWolfe(redVial);

        //  Paso 3
        paso3FrankWolfe(redVial);

        //  Paso 4
        paso4FrankWolfe(redVial);

        // Paso 5
        paso5FrankWolfe(redVial);

        if (k > 0.001)
        {
            condicionesSiguienteIteracion(redVial);
        }
    }
}


// Pasos del algoritmo Frank Wolfe
public void pasoInicialFrankWolfe(RedVial redVial)
{
    // Dijkstra
    dijkstra.caminoMasCortoSobre(TFA);
    dijkstra.dijkstra(redVial);

    // TodoNada
    todoNada.cargarFlujoX(redVial.arcos, dijkstra.caminoMasCorto);
}

public void paso1FrankWolfe(RedVial redVial)
{
    // Actualizar tiempos
    funciones.calcularTiempos(redVial.arcos);

    // Encontrar z
    z = funciones.calcularZ(redVial.arcos);
}

public void paso2FrankWolfe(RedVial redVial)
{
    // Dijkstra
    dijkstra.caminoMasCortoSobre(TIEMPO);
    dijkstra.dijkstra(redVial);
```

```
        // Todo Nada
        todoNada.cargarFlujoY(redVial.arcos, dijkstra.caminoMasCorto);
    }

    public void paso3FrankWolfe(RedVial redVial)
    {
        // Metodo de biseccion
        lambda = biseccion.calcularPuntoMedio(redVial.arcos);
        System.Console.WriteLine("lambda: " + lambda);
    }

    public void paso4FrankWolfe(RedVial redVial)
    {
       double x;
       double xNew;
       double y;
       double numRows;

       // Reasigna el flujo dentro de arcosDataTable
       numRows = redVial.arcos.Rows.Count;

       for (int i = 0; i < numRows; i++)
       {
          x = Convert.ToDouble(redVial.arcos.Rows[i]["x"]);
          y = Convert.ToDouble(redVial.arcos.Rows[i]["y"]);
          xNew = x + lambda * (y - x);
          redVial.arcos.Rows[i]["x_n+1"] = xNew;
       }
    }

    public void paso5FrankWolfe(RedVial redVial)
    {
       double x;
       double xNew;
       double difFlujos;
       double difFlujosCuadrado;
       double sumatoriaDiferencias;
       double numerador;
       double denominador;
       double numRows;

       // CRITERIO DE CONVERGENCIA
       numRows = redVial.arcos.Rows.Count;
       sumatoriaDiferencias = 0;

       for (int i = 0; i < numRows; i++)
       {
          x = Convert.ToDouble(redVial.arcos.Rows[i]["x"]);
          xNew = Convert.ToDouble(redVial.arcos.Rows[i]["x_n+1"]);
          difFlujos = xNew - x;
          difFlujosCuadrado = difFlujos * difFlujos;
          sumatoriaDiferencias = sumatoriaDiferencias + difFlujosCuadrado;
       }

       numerador = System.Math.Sqrt(sumatoriaDiferencias);
       denominador = 0;
       denominador = Convert.ToDouble(redVial.arcos.Compute("Sum(x)", ""));
       k = (numerador / denominador);
```

```
            Console.WriteLine("k: " + k);
        }

        public void condicionesSiguienteIteracion(RedVial redVial)
        {
            double x;
            double numRows;

            numRows = redVial.arcos.Rows.Count;
            for (int i = 0; i < numRows; i++)
            {
                x = Convert.ToDouble(redVial.arcos.Rows[i]["x_n+1"]);
                redVial.arcos.Rows[i]["x"] = x;
                redVial.arcos.Rows[i]["y"] = 0;
                redVial.arcos.Rows[i]["x_n+1"] = 0;
            }
        }
    }
}
```

## Dijkstra.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
    public class Dijkstra
    {
       public int opcionDijkstra;
       public DataTable q;
       public DataTable s;
       public DataTable caminoMasCorto;

       public void caminoMasCortoSobre(int opcionDijkstra)
       {
           this.opcionDijkstra = opcionDijkstra;
       }

       public void dijkstra(RedVial redVial)
       {
          // Dijkstra
          inicializarDijkstra(redVial.nodos);
          algoritmoDijkstra(redVial.arcos);

          // Resultado
          resultadoDijkstra(redVial.arcos);
       }

       public void inicializarDijkstra(DataTable nodos)
       {
          int lenghtNodos;
          double infinity;
          DataRow agregarFila;
          DataRow[] costoRow;

          q = new DataTable();
          s = new DataTable();

          infinity = 9999999999999;

          // Estructura e inicilializacion metodo dijkstra para qDataTable
          // Columna Nodo
          DataColumn nodoColumn = new DataColumn();
          nodoColumn.DataType = typeof(int);
          nodoColumn.DefaultValue = null;
          nodoColumn.ColumnName = "Nodo";
          q.Columns.Add(nodoColumn);

          // Agregar la Columna Nodo_Anterior
          // previo[v]:= indefinido
          DataColumn nodoAnteriorColumn = new DataColumn();
          nodoAnteriorColumn.DataType = typeof(int);
          nodoAnteriorColumn.DefaultValue = null;
          nodoAnteriorColumn.ColumnName = "Nodo_Anterior";
```

```
    q.Columns.Add(nodoAnteriorColumn);

    // Agregar la columna Cost
    // d[v]:= infinito
    DataColumn costoColumn = new DataColumn();
    costoColumn.DataType = typeof(double);
    costoColumn.DefaultValue = infinity;
    costoColumn.ColumnName = "Costo";
    q.Columns.Add(costoColumn);

    // Copiar Columna nodos (Nodos >>> qDataTable)
    lenghtNodos = nodos.Rows.Count;
    for (int i = 1; i <= lenghtNodos; i++)
    {
       agregarFila = q.NewRow();
       agregarFila["Nodo"] = nodos.Rows[i - 1]["Nodo"];
       q.Rows.Add(agregarFila);
    }

    // d[s]:=0
    String condicion;
    condicion = "Nodo = " + NODOINICIAL;
    costoRow = q.Select(condicion);
    costoRow[0]["Costo"] = 0.0;

    // Estructura e inicilializacion metodo dijkstra para sDataTable
    s = q.Clone();
}

public void algoritmoDijkstra(DataTable arcos)
{
    int numArcosSalientes;
    int lenght_vRow;
    int u;
    int v;
    int previous;
    double valorCostoMinimo;
    double dv;
    double du;
    double duv = 0;
    DataRow[] costoMinimoRow;
    DataRow[] arcosSalientesRow;
    DataRow[] vRow;
    DataRow[] uRow;
    DataRow[] coRow;
    string condicion1;
    string condicion2;
    string condicion3;
    string condicion4;
    string condicion5;

    // Inicializacion de la variable u para poder iniciar el ciclo while
    u = 0;

    // Dijkstra
    while ((q.Rows.Count > 0) & (u != NODOFINAL))
    {
       // u := Extraer min(Q)
```

```
// S := S union {u}

  // Calcula el costo minimo de la tabla Q
  valorCostoMinimo = (double)q.Compute("Min(Costo)", "");
  valorCostoMinimo = Math.Round(valorCostoMinimo, 6);

  // Condicion con el valor del costo minimo de la tabla Q
  condicion1 = "Costo = " + valorCostoMinimo;

  // Selecciona los renglones de la tabla Q con el costo minimo
  costoMinimoRow = q.Select(condicion1);

  // Selecciona u (Nodo con el costo minimo)
  u = (int)costoMinimoRow[0]["Nodo"];

  // S := S union {u}
  s.ImportRow(costoMinimoRow[0]);

  // Elimina la fila con el renglon que posee el costo minimo.
  costoMinimoRow[0].Delete();

  // Arcos Salientes = "Seleccionar To_Nodo de Arcos donde From_Nodo = u "
  condicion2 = "From_Nodo = " + u;
  arcosSalientesRow = arcos.Select(condicion2);
  numArcosSalientes = arcosSalientesRow.GetLength(0);

  // Para cada (u,v) saliente de u
  for (int i = 1; i <= numArcosSalientes; i++)
  {
      // Seleccion v
      v = (int)arcosSalientesRow[i - 1]["To_Nodo"];

      // dv (Seleccionar Costo de Q donde Nodo = v )
      condicion3 = "Nodo = " + v;
      vRow = q.Select(condicion3);
      lenght_vRow = vRow.GetLength(0);

      if (lenght_vRow > 0)
      {
         // dv (Seleccionar Costo de Q donde Nodo = v )
         dv = (double)vRow[0]["Costo"];

         // du (Seleccionar Costo de S donde Nodo_Actual = u )
         condicion4 = "Nodo = " + u;
         uRow = s.Select(condicion4);
         du = (double)uRow[0]["Costo"];

         // duv (Seleccionar Longitud de ARCOS donde from = u y to = v)
         condicion5 = "From_Nodo = " + u + " AND To_Nodo = " + v;
         coRow = arcos.Select(condicion5);

         if (opcionDijkstra == 1)
         {
            // Define sobre quien corre Dijkstra
            duv = (double)coRow[0]["Tiempo"];
         }

         if (opcionDijkstra == 0)
```

```
                {
                    // Define sobre quien corre Dijkstra
                    duv = (double)coRow[0]["Tfa"];
                }

                //  Tecnica de relajacion (u,v)
                if (dv > (du + duv))
                {
                    dv = du + duv;
                    previous = u;
                    dv = Math.Round(dv, 6);
                    vRow[0]["Costo"] = dv;
                    vRow[0]["Nodo_Anterior"] = u;
                }

            }
        }
    }
}

public void resultadoDijkstra(DataTable arcos)
{
    int from;
    int to;
    int varNodo;
    int lenghtnodosRutaDataTable;
    string condicion;
    DataRow[] resRow;
    DataRow[] rutaRow;
    DataRow[] existRouteRow;
    DataTable rutaNodosDataTable;

    rutaNodosDataTable = new DataTable();

    // Copia la estructura de la qDataTable
    rutaNodosDataTable = q.Clone();

    // Asigna nodoFinal a varNodo
    varNodo = NODOFINAL;

    // Condiciona para query
    condicion = "Nodo = " + NODOFINAL;

    // Selecciona DataRow de sDataTable con el nodoFinal
    existRouteRow = s.Select(condicion);

    // Si el nodoFinal posee nodo anterior
    if (existRouteRow[0]["Nodo_Anterior"] != DBNull.Value)
    {
        // Llena nodosRutaDataTable con relacion(Nodo/NodoAnterior/Costo)
        while (varNodo != NODOINICIAL)
        {
            condicion = "Nodo = " + varNodo;
            resRow = s.Select(condicion);
            rutaNodosDataTable.ImportRow(resRow[0]);
            varNodo = (int)resRow[0]["Nodo_Anterior"];
        }
```

```
                // Inserta dataRow del nodoInicial al nodosRutaDataTable
                condicion = "Nodo = " + NODOINICIAL;
                resRow = s.Select(condicion);
                rutaNodosDataTable.ImportRow(resRow[0]);
             }

            caminoMasCorto = new DataTable();
            caminoMasCorto = arcos.Clone();

            lenghtnodosRutaDataTable = rutaNodosDataTable.Rows.Count;
            lenghtnodosRutaDataTable = lenghtnodosRutaDataTable - 1;

            for (int k = 0; k < lenghtnodosRutaDataTable; k++)
            {
               from = Convert.ToInt16(rutaNodosDataTable.Rows[k]["Nodo_Anterior"]);
               to = Convert.ToInt16(rutaNodosDataTable.Rows[k]["Nodo"]);
               condicion = "From_Nodo = " + from + " AND To_Nodo = " + to;
               rutaRow = arcos.Select(condicion);
               caminoMasCorto.ImportRow(rutaRow[0]);
            }
         }
      }
   }
```

## TodoNada.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
    public class TodoNada
    {
       int flujoVehicular;

       public TodoNada(int flujoVehicular)
       {
          this.flujoVehicular = flujoVehicular;
       }

       public void cargarFlujoX(DataTable arcos, DataTable caminoMasCorto)
       {
          int arcoID;
          double numRows;
          string condicion;
          DataRow[] idRow;

          // Obtiene el numero de filas de la ruta
          numRows = caminoMasCorto.Rows.Count;

          // Actualiza el flujoV en la tabla arcosDataTable
          for (int i = 0; i < numRows; i++)
          {
             arcoID = Convert.ToInt16(caminoMasCorto.Rows[i]["ArcoID"]);
             condicion = "ArcoID = " + arcoID;
             idRow = arcos.Select(condicion);
             idRow[0]["x"] = flujoVehicular;
          }
       }

       public void cargarFlujoY(DataTable arcos, DataTable caminoMasCorto)
       {
          int arcoID;
          double numRows;
          string condicion;
          DataRow[] idRow;

          // Obtiene el numero de filas de la ruta
          numRows = caminoMasCorto.Rows.Count;

          // Actualiza el flujoV en la tabla arcosDataTable
          for (int i = 0; i < numRows; i++)
          {
             arcoID = Convert.ToInt32(caminoMasCorto.Rows[i]["ArcoID"]);
             condicion = "ArcoID = " + arcoID;
             idRow = arcos.Select(condicion);
```

```
            idRow[0]["y"] = flujoVehicular;
           }
        }
    }
}
```

## RedVial.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
    public class RedVial
    {
       public int flujoVehicularPeriferico;
       public int flujoVehicularRevolucion;
       public double tiempoPeriferico;
       public double tiempoRevolucion;
       public DataTable arcos;
       public DataTable nodos;
       public DataTable patronRecomendado;

       public void crearRedVial()
       {
          DataTable temporal1 = new DataTable();
          DataTable temporal2 = new DataTable();

          // Creacion de la tabla de arcos
          crearArcosDataTable();

          // Leer CSV de arcos
          temporal1 = leerArcosCsv();

          // Copiar valores CSV arcos a la tabla de arcos
          cargarArcosDataTable(temporal1);

          // Creacion de la tabla de nodos
          crearNodosDataTable();

          // Leer CSV de nodos
          temporal2 = leerNodosCsv();

          // Cargar valores a la tabla de nodos
          cargarNodosDataTable(temporal2);
       }

       public void crearArcosDataTable()
       {
          // Creacion de la tabla de arcos
          arcos = new DataTable();

          // Creacion de la estructura de la tabla de arcos

          // ArcoID
          DataColumn arcoIDColumn = new DataColumn();
          arcoIDColumn.ColumnName = "ArcoID";
          arcoIDColumn.DataType = typeof(int);
          arcoIDColumn.ReadOnly = true;
          arcos.Columns.Add(arcoIDColumn);
```

```
// Nombre
DataColumn arcoColumn = new DataColumn();
arcoColumn.ColumnName = "Arco_Nombre";
arcoColumn.DataType = typeof(string);
arcos.Columns.Add(arcoColumn);

// From
DataColumn fromColumn = new DataColumn();
fromColumn.ColumnName = "From_Nodo";
fromColumn.DataType = typeof(int);
arcos.Columns.Add(fromColumn);

// To
DataColumn toColumn = new DataColumn();
toColumn.ColumnName = "To_Nodo";
toColumn.DataType = typeof(int);
arcos.Columns.Add(toColumn);

// Longitud
DataColumn longitudColumn = new DataColumn();
longitudColumn.ColumnName = "Longitud";
longitudColumn.DataType = typeof(double);
arcos.Columns.Add(longitudColumn);

// Dir
DataColumn dirColumn = new DataColumn();
dirColumn.ColumnName = "Dir";
dirColumn.DataType = typeof(int);
arcos.Columns.Add(dirColumn);

// LAN Practicos
DataColumn lanPracticosColumn = new DataColumn();
lanPracticosColumn.ColumnName = "LAN_practicos";
lanPracticosColumn.DataType = typeof(int);
arcos.Columns.Add(lanPracticosColumn);

// Capacidad Teorica
DataColumn capTeoricaPorCarrilColumn = new DataColumn();
capTeoricaPorCarrilColumn.ColumnName = "Capacidad_teorica_por_carril";
capTeoricaPorCarrilColumn.DataType = typeof(double);
arcos.Columns.Add(capTeoricaPorCarrilColumn);

// VEL_FF
DataColumn velFFColumn = new DataColumn();
velFFColumn.ColumnName = "Vel_FF";
velFFColumn.DataType = typeof(double);
arcos.Columns.Add(velFFColumn);

// Ancho
DataColumn anchoColumn = new DataColumn();
anchoColumn.ColumnName = "Ancho";
anchoColumn.DataType = typeof(double);
arcos.Columns.Add(anchoColumn);

// Link Type
DataColumn linkTypeColumn = new DataColumn();
linkTypeColumn.ColumnName = "Link_Type";
linkTypeColumn.DataType = typeof(string);
```

```
arcos.Columns.Add(linkTypeColumn);

// Tipo
DataColumn tipoColumn = new DataColumn();
tipoColumn.ColumnName = "Tipo";
tipoColumn.DataType = typeof(int);
arcos.Columns.Add(tipoColumn);

// Tfa
DataColumn tfaColumn = new DataColumn();
tfaColumn.ColumnName = "Tfa";
tfaColumn.DataType = typeof(double);
arcos.Columns.Add(tfaColumn);

// Daf
DataColumn dafColumn = new DataColumn();
dafColumn.ColumnName = "Daf";
dafColumn.DataType = typeof(double);
arcos.Columns.Add(dafColumn);

// Columna x
DataColumn xColumn = new DataColumn();
xColumn.ColumnName = "x";
xColumn.DataType = typeof(double);
xColumn.DefaultValue = 0;
arcos.Columns.Add(xColumn);

// Columna y
DataColumn yColumn = new DataColumn();
yColumn.ColumnName = "y";
yColumn.DataType = typeof(double);
yColumn.DefaultValue = 0;
arcos.Columns.Add(yColumn);

// Capacidad arco
DataColumn capArcoColumn = new DataColumn();
capArcoColumn.ColumnName = "Capacidad_arco";
capArcoColumn.DataType = typeof(double);
arcos.Columns.Add(capArcoColumn);

// Capacidad cruce
DataColumn capCruceColumn = new DataColumn();
capCruceColumn.ColumnName = "Capacidad_cruce";
capCruceColumn.DataType = typeof(int);
arcos.Columns.Add(capCruceColumn);

// Capacidad aforo hora pico
DataColumn aforoHoraPicoColumn = new DataColumn();
aforoHoraPicoColumn.ColumnName = "Aforo_hora_pico";
aforoHoraPicoColumn.DataType = typeof(int);
arcos.Columns.Add(aforoHoraPicoColumn);

// Tiempo
DataColumn tiempoColumn = new DataColumn();
tiempoColumn.ColumnName = "Tiempo";
tiempoColumn.DataType = typeof(double);
arcos.Columns.Add(tiempoColumn);
```

```
// Ciclo
DataColumn cicloColumn = new DataColumn();
cicloColumn.ColumnName = "Ciclo";
cicloColumn.DataType = typeof(double);
cicloColumn.DefaultValue = 0;
arcos.Columns.Add(cicloColumn);

// Tiempo ciclo verde
DataColumn verdeColumn = new DataColumn();
verdeColumn.ColumnName = "Verde";
verdeColumn.DataType = typeof(double);
verdeColumn.DefaultValue = 0;
arcos.Columns.Add(verdeColumn);

// LAN Reales
DataColumn lanRealesColumn = new DataColumn();
lanRealesColumn.ColumnName = "LAN_reales";
lanRealesColumn.DataType = typeof(int);
lanRealesColumn.DefaultValue = 0;
arcos.Columns.Add(lanRealesColumn);

// Alfa
DataColumn alfaColumn = new DataColumn();
alfaColumn.ColumnName = "Alfa";
alfaColumn.DataType = typeof(double);
arcos.Columns.Add(alfaColumn);

// Beta
DataColumn betaColumn = new DataColumn();
betaColumn.ColumnName = "Beta";
betaColumn.DataType = typeof(double);
arcos.Columns.Add(betaColumn);

// A
DataColumn aColumn = new DataColumn();
aColumn.ColumnName = "A";
aColumn.DataType = typeof(double);
arcos.Columns.Add(aColumn);

// Z
DataColumn zColumn = new DataColumn();
zColumn.ColumnName = "z";
zColumn.DataType = typeof(double);
arcos.Columns.Add(zColumn);

// x_n+1
DataColumn newFlujoXColumn = new DataColumn();
newFlujoXColumn.ColumnName = "x_n+1";
newFlujoXColumn.DataType = typeof(double);
newFlujoXColumn.DefaultValue = 0;
arcos.Columns.Add(newFlujoXColumn);

// Tiempo caracteristico
DataColumn newtiempoCaracteristicoColumn = new DataColumn();
newtiempoCaracteristicoColumn.ColumnName = "TiempoCaracteristico";
newtiempoCaracteristicoColumn.DataType = typeof(double);
newtiempoCaracteristicoColumn.DefaultValue = 0;
arcos.Columns.Add(newtiempoCaracteristicoColumn);
```

```
        // Flujo / Capacidad
        DataColumn flujoEntreCapacidadColumn = new DataColumn();
        flujoEntreCapacidadColumn.ColumnName = "Flujo/Capacidad";
        flujoEntreCapacidadColumn.DataType = typeof(double);
        flujoEntreCapacidadColumn.DefaultValue = 0;
        arcos.Columns.Add(flujoEntreCapacidadColumn);
    }

    public DataTable leerArcosCsv()
    {
        // Leer csv de arcos
        String arcosCSV = "C:/EUbeta/EUbeta/RedVialDePrueba/arcosCSV.csv";
        DataTable temporalDataTable = new DataTable();
        temporalDataTable = GestorCSV.csvToDataTable(arcosCSV);
        return temporalDataTable;
    }

    public void cargarArcosDataTable(DataTable temporal1)
    {
        int numRows;
        DataRow row;

        // Copiar a tabla de arcos
        numRows = 0;
        numRows = temporal1.Rows.Count;
        for (int i = 1; i <= numRows; i++)
        {
            // Creacion de una fila de la tabla de arcos
            row = arcos.NewRow();

            // Copiar valores a la fila
            row["ArcoID"] = temporal1.Rows[i - 1]["ArcoID"];
            row["Arco_Nombre"] = temporal1.Rows[i - 1]["Arco_Nombre"];
            row["From_Nodo"] = temporal1.Rows[i - 1]["From_Nodo"];
            row["To_Nodo"] = temporal1.Rows[i - 1]["To_Nodo"];
            row["Longitud"] = temporal1.Rows[i - 1]["Longitud"];
            row["Dir"] = temporal1.Rows[i - 1]["Dir"];
            row["LAN_practicos"] = temporal1.Rows[i - 1]["LAN_practicos"];
            row["Vel_FF"] = temporal1.Rows[i - 1]["Vel_FF"];
            row["Link_Type"] = temporal1.Rows[i - 1]["Link_Type"];

            // Si es Semaforizada = 1 - Si es Acceso Control = 2
            if ((string)temporal1.Rows[i - 1]["Link_Type"] == "Acceso Control")
                row["Tipo"] = 1;
            else if ((string)temporal1.Rows[i - 1]["Link_Type"] == "Semaforizada")
                row["Tipo"] = 2;

            row["Tfa"] = temporal1.Rows[i - 1]["Tfa"];
            row["Capacidad_arco"] =
                    Convert.ToDouble(temporal1.Rows[i - 1]["Capacidad_arco"]);

            // Lee campos nulos y asigna en caso de no ser campo nulo
            if (Convert.ToString(temporal1.Rows[i - 1]["Ciclo"]) == "")
            { }
            else
            {
                row["Ciclo"] = temporal1.Rows[i - 1]["Ciclo"];
                row["Verde"] = temporal1.Rows[i - 1]["Verde"];
```

```
        }

        row["LAN_Reales"] = temporal1.Rows[i - 1]["LAN_Reales"];
        row["Alfa"] = temporal1.Rows[i - 1]["Alfa"];
        row["Beta"] = temporal1.Rows[i - 1]["Beta"];
        row["A"] = temporal1.Rows[i - 1]["A"];

        // Importar la fila a la tabla de arcos
        arcos.Rows.Add(row);
    }
}

public void crearNodosDataTable()
{
    nodos = new DataTable();

    // Tipo
    DataColumn nodoColumn = new DataColumn();
    nodoColumn.ColumnName = "Nodo";
    nodoColumn.DataType = typeof(int);
    nodos.Columns.Add(nodoColumn);

}

public DataTable leerNodosCsv()
{
     // Leer csv de nodos
     String nodosCSV;
     DataTable temporalDataTable = new DataTable();
     nodosCSV = "C:/EUbeta/EUbeta/RedVialDePrueba/nodosCSV.csv";
     temporalDataTable = GestorCSV.csvToDataTable(nodosCSV);

     return temporalDataTable;
}

public void cargarNodosDataTable(DataTable temporal2)
{
    int numRows;
    DataRow row;

    // Copiar a tabla de nodos
    numRows = 0;
    numRows = temporal2.Rows.Count;

    for (int i = 1; i <= numRows; i++)
    {
        // Creacion de una fila de la tabla de arcos
        row = nodos.NewRow();

        // Copiar los valores
        row["Nodo"] = temporal2.Rows[i - 1]["Nodo"];

        // Importar la fila a la tabla de nodos
        nodos.Rows.Add(row);
    }
}
```

```
public void obtenerPatronRecomendado()
{
   int numRows;
   DataRow row;

   // Tabla a exportar
   patronRecomendado = new DataTable();

   // ArcoID
   DataColumn arcoIDColumn = new DataColumn();
   arcoIDColumn.ColumnName = "ArcoID";
   arcoIDColumn.DataType = typeof(int);
   arcoIDColumn.ReadOnly = true;
   patronRecomendado.Columns.Add(arcoIDColumn);

   // Columna x
   DataColumn xColumn = new DataColumn();
   xColumn.ColumnName = "x";
   xColumn.DataType = typeof(int);
   xColumn.DefaultValue = 0;
   patronRecomendado.Columns.Add(xColumn);

   // Flujo/Capacidad
   DataColumn testColumn = new DataColumn();
   testColumn.ColumnName = "Flujo/Capacidad";
   testColumn.DataType = typeof(double);
   testColumn.DefaultValue = 0;
   patronRecomendado.Columns.Add(testColumn);

   numRows = 0;
   numRows = arcos.Rows.Count;

   for (int i = 0; i < numRows; i++)
   {
      row = patronRecomendado.NewRow();
      row["ArcoID"] = arcos.Rows[i]["ArcoID"];
      row["x"] = arcos.Rows[i]["x"];
      row["Flujo/Capacidad"] = arcos.Rows[i]["Flujo/Capacidad"];
      patronRecomendado.Rows.Add(row);

   }
}

public void obtenerFlujoAsignadoARutas()
{
   int i;
   int tipo;
   double numRows;

   // Flujo Periferico
   numRows = arcos.Rows.Count;

   for (i = 0; i < numRows; i++)
   {
      tipo = Convert.ToInt32(arcos.Rows[i]["Tipo"]);
      if (tipo == 1)
      {
         flujoVehicularPeriferico = Convert.ToInt32(arcos.Rows[i]["x"]);
```

```
            break;
         }
      }

      // Flujo Revolucion
      numRows = arcos.Rows.Count;

      for (i = 0; i < numRows; i++)
      {
         tipo = Convert.ToInt32(arcos.Rows[i]["Tipo"]);
         if (tipo == 2)
         {
            flujoVehicularRevolucion = Convert.ToInt32(arcos.Rows[i]["x"]);
            break;
         }
      }
   }

   public void calcularTiemposDeViaje()
   {
      int i;
      int tipo;
      double numRows;
      double tiempoParcial;

      // Tiempo de viaje Periferico
      tiempoPeriferico = 0.0;
      numRows = arcos.Rows.Count;

      for (i = 0; i < numRows; i++)
      {
         tipo = Convert.ToInt32(arcos.Rows[i]["Tipo"]);

         if (tipo == 1)
         {
            tiempoParcial = Convert.ToDouble(arcos.Rows[i]["tiempo"]);
            tiempoPeriferico = tiempoPeriferico + tiempoParcial;
         }
      }

      // Tiempo de viaje Revolucion
      i = 0;
      tiempoParcial = 0;
      tiempoRevolucion = 0;
      for (i = 0; i < numRows; i++)
      {
         tipo = Convert.ToInt32(arcos.Rows[i]["Tipo"]);

         if (tipo == 2)
         {
            tiempoParcial = Convert.ToDouble(arcos.Rows[i]["tiempo"]);
            tiempoRevolucion = tiempoRevolucion + tiempoParcial;
         }
      }
   }

   public void calcularFlujoEntreCapacidadArco()
   {
```

```
        int i;
        double flujo;
        double capacidad;
        double flujoEntreCapacidad;
        double numRows;

        numRows = arcos.Rows.Count;
        for (i = 0; i < numRows; i++)
        {
            flujo = Convert.ToDouble(arcos.Rows[i]["x"]);
            capacidad = Convert.ToDouble(arcos.Rows[i]["Capacidad_arco"]);
            flujoEntreCapacidad = flujo / capacidad;
            arcos.Rows[i]["Flujo/Capacidad"] = flujoEntreCapacidad;
        }
    }

    public void exportarCSVPatronRecomendado()
    {
       obtenerPatronRecomendado();
       GestorCSV.dataTableToCSV(patronRecomendado, "patronRecomendado.csv", ",");
    }

    public DataTable obtenerDatosRuta(int tipo) {

        int fromNodo = 0;
        DataRow arco;
        DataRow[] arcosRuta;
        DataTable ruta = new DataTable();
        DataTable temporal = new DataTable();

        // Construccion de la tabla ruta

        // ArcoID
        DataColumn arcoIDColumn = new DataColumn();
        arcoIDColumn.ColumnName = "ArcoID";
        arcoIDColumn.DataType = typeof(int);
        arcoIDColumn.ReadOnly = true;
        ruta.Columns.Add(arcoIDColumn);

        // Nombre
        DataColumn arcoColumn = new DataColumn();
        arcoColumn.ColumnName = "Nombre";
        arcoColumn.DataType = typeof(string);
        ruta.Columns.Add(arcoColumn);

        // Longitud
        DataColumn longitudColumn = new DataColumn();
        longitudColumn.ColumnName = "Longitud [km]";
        longitudColumn.DataType = typeof(double);
        ruta.Columns.Add(longitudColumn);

        // Capacidad arco
        DataColumn capArcoColumn = new DataColumn();
        capArcoColumn.ColumnName = "Capacidad_arco";
        capArcoColumn.DataType = typeof(double);
        ruta.Columns.Add(capArcoColumn);
```

```
// Flujo
DataColumn flujoColumn = new DataColumn();
flujoColumn.ColumnName = "Flujo [veh/h]";
flujoColumn.DataType = typeof(double);
ruta.Columns.Add(flujoColumn);

// Tiempo Parcial
DataColumn tiempoPColumn = new DataColumn();
tiempoPColumn.ColumnName = "Tiempo parcial [h]";
tiempoPColumn.DataType = typeof(double);
ruta.Columns.Add(tiempoPColumn);

// Tiempo Acumulado
DataColumn tiempoAColumn = new DataColumn();
tiempoAColumn.ColumnName = "Tiempo acumulado [h]";
tiempoAColumn.DataType = typeof(double);
ruta.Columns.Add(tiempoAColumn);

// Flujo/Capacidad
DataColumn flujoEntreCapacidadColumn = new DataColumn();
flujoEntreCapacidadColumn.ColumnName = "Flujo/Capacidad";
flujoEntreCapacidadColumn.DataType = typeof(double);
flujoEntreCapacidadColumn.DefaultValue = 0;
ruta.Columns.Add(flujoEntreCapacidadColumn);


// Select a la tabla de arcos [periferico=1, revolucion=2]
String condicion1 = "Tipo = " + tipo;
arcosRuta = arcos.Select(condicion1);
int lenghtArcos = arcosRuta.GetLength(0);

temporal = arcos.Clone();

for (int i=0; i<lenghtArcos; i++) {
   temporal.ImportRow(arcosRuta[i]);
}

fromNodo = 30;
String condicion2 = "";
double tiempoAcumulado = 0;

while (true) {

   condicion2 = "From_Nodo = " + fromNodo;
   arcosRuta = temporal.Select(condicion2);

   arco = ruta.NewRow();

   arco["ArcoID"] = arcosRuta[0]["ArcoID"];
   arco["Nombre"] = arcosRuta[0]["Arco_Nombre"];
   arco["Longitud [km]"] = arcosRuta[0]["Longitud"];
   arco["Capacidad_arco"] = arcosRuta[0]["Capacidad_arco"];
   arco["Flujo/Capacidad"] = arcosRuta[0]["Flujo/Capacidad"];


   if (tipo == 1) {
      arco["Flujo [veh/h]"] = Controller.eu.redVial.flujoVehicularPeriferico;
   }
```

```
                else if (tipo == 2) {
                    arco["Flujo [veh/h]"] = Controller.eu.redVial.flujoVehicularRevolucion;
                }

                arco["Tiempo parcial [h]"] = Math.Round((double)arcosRuta[0]["Tiempo"], 6);

                tiempoAcumulado += (double)arcosRuta[0]["Tiempo"];
                arco["Tiempo acumulado [h]"] = Math.Round(tiempoAcumulado, 6);

                ruta.Rows.Add(arco);

                fromNodo = (int) arcosRuta[0]["To_Nodo"];

                if (fromNodo == 57) { break;}

            }

            return ruta;
        }
    }
}
```

## Funciones.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
   public abstract class Funciones
   {
      public abstract void calcularTiempos(DataTable arcos);
      public abstract double calcularZ(DataTable arcos);

      // Tiempo en [min]
      public const double tiempoAnalisis = 0.25;
   }
}
```

## BPR.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
    public class BPR : Funciones
    {
       public override void calcularTiempos(DataTable arcos)
       {
          double x;
          double Q;
          double tfa;
          double alfa;
          double beta;
          double tiempo;
          double X;
          double numRows;

          numRows = arcos.Rows.Count;

          // Calcula el tiempo para cada arco
          for (int i = 0; i < numRows; i++)
          {
             alfa = Convert.ToDouble(arcos.Rows[i]["Alfa"]);
             beta = Convert.ToDouble(arcos.Rows[i]["Beta"]);
             tfa = Convert.ToDouble(arcos.Rows[i]["Tfa"]);
             x = Convert.ToDouble(arcos.Rows[i]["x"]);
             Q = Convert.ToDouble(arcos.Rows[i]["Capacidad_arco"]);

             X = (x / Q);
             tiempo = tfa * (1.0 + alfa * System.Math.Pow(X, beta));
             arcos.Rows[i]["Tiempo"] = Math.Round(tiempo, 6);
          }

       }

       public override double calcularZ(DataTable arcos)
       {
          double tfa;
          double x;
          double alfa;
          double beta;
          double Q;
          double zArco;
          double z;
          double numRows;

          numRows = arcos.Rows.Count;

          // Calcula z para cada arco
          for (int i = 0; i < numRows; i++)
          {
             tfa = Convert.ToDouble(arcos.Rows[i]["Tfa"]);
```

```
            x = Convert.ToDouble(arcos.Rows[i]["x"]);
            alfa = Convert.ToDouble(arcos.Rows[i]["Alfa"]);
            beta = Convert.ToDouble(arcos.Rows[i]["Beta"]);
            Q = Convert.ToDouble(arcos.Rows[i]["Capacidad_arco"]);
            zArco = (tfa * x) +
            ((tfa * alfa * Math.Pow(x, (beta + 1))) / ((beta + 1) * Math.Pow(Q, beta)));
            arcos.Rows[i]["z"] = zArco;
        }

        // Sumatoria de z
        z = 0;
        z = Convert.ToDouble(arcos.Compute("Sum(z)", ""));
        System.Console.WriteLine();
        System.Console.WriteLine("z(x) = " + z);
        z = Math.Round(z, 3);

        return z;

    }
  }
}
```

## Biseccion.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
    public abstract class Biseccion
    {
       public double puntoMedio;
       public DataTable tablaBiseccion;

       // Tiempo en [min]
       public const double tiempoAnalisis = 0.25;
       public abstract void algoritmoBiseccion(DataTable arcos);

       public void crearBiseccionDataTable(DataTable arcos)
       {
          double numRows;
          DataRow row;
          tablaBiseccion = new DataTable();

          // Estructura de la tabla
          // Agregar la columna F(a)
          DataColumn faColumn = tablaBiseccion.Columns.Add("Fa", typeof(double));
          faColumn.DefaultValue = 0;

          // Agregar la columna F(b)
          DataColumn fbColumn = tablaBiseccion.Columns.Add("Fb", typeof(double));
          fbColumn.DefaultValue = 0;

          // Agregar la columna F(a) * F(b)
          DataColumn faXfbColumn =
             tablaBiseccion.Columns.Add("FPuntoMedio", typeof(double));
          faXfbColumn.DefaultValue = 0;

          // Generar renglones de la tabla
          numRows = arcos.Rows.Count;

          // Agregar filas
          for (int i = 0; i < numRows; i++)
          {
             row = tablaBiseccion.NewRow();
             row["Fa"] = 0;
             tablaBiseccion.Rows.Add(row);
          }

       }

       public double calcularPuntoMedio(DataTable arcos)
       {
           // Creacion de la tabla de biseccion
           crearBiseccionDataTable(arcos);
```

```
            // Ejecutar algoritmo de biseccion
            algoritmoBiseccion(arcos);

            return puntoMedio;
        }

    }
}
```

## B_BPR.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace PEU
{
    public class B_BPR : Biseccion
    {
       public override void algoritmoBiseccion(DataTable arcos)
       {
          double x;
          double y;
          double tfa;
          double alfa;
          double beta;
          double Q;
          double numRows;
          double a;
          double b;
          double cn;
          double cn1;
          double fa;
          double fb;
          double fcn;
          double sumatoriaFa = 0;
          double sumatoriaFb = 0;
          double sumatoriaCn = 0;

          // Numero de renglones de arcosDataTable
          numRows = arcos.Rows.Count;

          // Algoritmo de Biseccion BPR

          // 1. Definir intervalo [a,b] que posee la raiz
          a = 0.0;
          b = 1.0;

          // 2. Encontrar la primera aproximacion a la raiz
          cn = (a + b) / 2.0;

          // Repetir pasos 3,4 y 5 hasta obtener la raiz
          while (true)
          {
             // 3. Determinar en que subintervalo se encuentra la raíz:
             // Evaluar la funcion en los extremos
             for (int i = 0; i < numRows; i++)
             {
                x = Convert.ToDouble(arcos.Rows[i]["x"]);
                y = Convert.ToDouble(arcos.Rows[i]["y"]);
                tfa = Convert.ToDouble(arcos.Rows[i]["Tfa"]);
                alfa = Convert.ToDouble(arcos.Rows[i]["Alfa"]);
                beta = Convert.ToDouble(arcos.Rows[i]["Beta"]);
                Q = Convert.ToDouble(arcos.Rows[i]["Capacidad_arco"]);
```

```
                if ((y - x) == 0)
                {
                    cn = 0;
                    break;
                }

                fa = (y - x) * tfa *
                  (1 + (alfa / (Math.Pow(Q, beta))) * Math.Pow((x + a * (y - x)),beta));
                fb = (y - x) * tfa *
                  (1 + (alfa / (Math.Pow(Q, beta))) * Math.Pow((x + b * (y - x)),beta));
                fcn = (y - x) * tfa *
                  (1 + (alfa / (Math.Pow(Q, beta))) * Math.Pow((x + cn * (y - x)),beta));

                tablaBiseccion.Rows[i]["Fa"] = fa;
                tablaBiseccion.Rows[i]["Fb"] = fb;
                tablaBiseccion.Rows[i]["FPuntoMedio"] = fcn;
            }

            sumatoriaFa = Convert.ToDouble(tablaBiseccion.Compute("Sum(Fa)", ""));
            sumatoriaFb = Convert.ToDouble(tablaBiseccion.Compute("Sum(Fb)", ""));
            sumatoriaCn =
                 Convert.ToDouble(tablaBiseccion.Compute("Sum(FPuntoMedio)", ""));

            // Verificar en que intervalo se encuentra la raiz
            if (sumatoriaFa * sumatoriaCn == 0)
            {
                break;
            }
            else if ((sumatoriaFa * sumatoriaCn) < 0)
            {
                b = cn;
            }
            else if ((sumatoriaFa * sumatoriaCn) > 0)
            {
                a = cn;
            }

            // 4. Calcular una nueva aproximación a la raíz
            cn1 = (a + b) / 2.0;

            // 5. Evaluar la aproximación relativa
            if (((Math.Abs(cn1 - cn)) / cn1) < 0.001)
            {
                cn = cn1;
                break;
            }
            else
            {
                cn = cn1;
            }
        }

        puntoMedio = cn;

    }
  }
}
```