UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA DE SISTEMAS – TRANSPORTE

DESARROLLO DE UN MÓDULO DE SOFTWARE SOBRE LA ASIGNACIÓN DE VIAJES A LA RED VIAL BAJO CONDICIONES DE CONGESTIONAMIENTO

TESIS
QUE PARA OBTENER EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
ING. FELIPE TONATIUH MIGUEL BAUTISTA

DIRECTOR DE TESIS
DR. RICARDO ACEVES GARCÍA

MÉXICO, D. F. JUNIO 2013

JURADO ASIGNADO:

Presidente:	Dr. José de Jesús Acosta Flores
Secretario:	M. I. Héctor Daniel Reséndiz López
Vocal:	Dr. Ricardo Aceves García
1 er. Suplente:	M. I. José Antonio Rivera Colmenero
2 do. Suplente:	M. I. Luis Alejandro Guzmán Castro

Ciudad Universitaria, México D.F.

DIRECTOR DE TESIS:

DR. RICARDO ACEVES GARCÍA

_____

FIRMA

Con amor

a mi madre,

padre,

hermana

y sobrino.

AGRADECIMIENTOS

A Dios por la oportunidad de vida.

A la Universidad Nacional Autónoma de México y a los profesores que me compartieron de sus conocimientos.

Al Consejo Nacional de Ciencia y Tecnología por la beca que hizo posible la realización de mis estudios de posgrado.

Al Dr. Ricardo Aceves García por la propuesta de este tema de tesis y la tutoría para su desarrollo.

Al Dr. Laurent Yves G. Dartois Girald y al M. I. José Antonio Rivera Colmenero por sus sugerencias e interés mostrado en este trabajo, como parte del comité evaluador de avance de tesis, para Trabajo de Investigación I, II, y Seminario de Investigación.

A los sinodales, por la revisión crítica del contenido de esta tesis y sus sugerencias.

Al Dr. Benito Sánchez Lara por brindarme la oportunidad de colaborar con él, así como a los compañeros del Laboratorio de Planeación.

Al M. C. Oscar Alejandro González Bustamante por su disposición y generosa asesoría en Java. Y al M. I. Guillermo Alfonso Martínez Aguilar por su enseñanza en los cursos de Java.

A mis padrinos Yolanda Miguel Morales y Pedro de la Cruz Bautista por recibirme cálidamente en su hogar durante mis estudios de la maestría.

A la Sra. Carolina Gayosso Reyes y al Sr. Joel Trejo Hernández por su amistad.

A mi padre Narcizo Miguel Morales, hermana Tonahi Miguel Bautista y a la Ing. Yessica Trejo Domínguez, por estar siempre conmigo, a pesar de las distancias.

Finalmente, mi mayor agradecimiento lo dirijo a mi madre Juana Bautista Aquino, por el gran apoyo que siempre me ha brindado.

RESUMEN

En la primera parte de este trabajo se coloca en contexto el desarrollo de un módulo de software para resolver el problema determinístico de asignación de viajes a la red bajo condiciones de congestionamiento, como herramienta para la planeación del transporte, análisis y toma de decisiones. La segunda parte describe su desarrollo en Eclipse, el código fuente en Java que resuelve el algoritmo Frank-Wolfe y las características técnicas de programación que hacen al módulo parte de gvSIG Desktop. En la tercera parte se muestra la funcionalidad que tiene el módulo de software en el entorno de un Sistema de información geográfica, las características de los datos de entrada, el uso de la interfaz gráfica de usuario y la visualización del resultado. Finalmente, en la cuarta parte, se exponen las conclusiones y recomendaciones para futuros desarrollos.

Palabras clave: asignación de viajes, algoritmo Frank-Wolfe, gvSIG, Java, planeación del transporte, modelos de transporte, sistemas de información geográfica.

ABSTRACT

The first part of this work is placed in context the development of software plug-in to solve the traffic assignment problem under congested conditions, deterministic case, as a tool of transportation planning, analysis and decision making. The second part describes its development in Eclipse, the Java source code that solves Frank-Wolfe algorithm and programing technical features that make plug-in of gvSIG Desktop. The third part shows the functionality that has the plug-in for Geographic information system environment, the characteristics of input data, the use of graphical user interface and display the result. Finally, the fourth part, conclusions and recommendations for future developments are exposed.

Key words: traffic assignment, Frank-Wolfe algorithm, gvSIG, Java, transport planning, transport modelling, geographical information systems.

CONTENIDO

AGRADECIMIENTOS.....vii
RESUMEN.....ix
ABSTRACT.....ix
CONTENIDO.....xi
ÍNDICE DE TABLAS.....xiii
ÍNDICE DE FIGURAS.....xiii
INTRODUCCIÓN.....1
CAPÍTULO 1. MARCO REFERENCIAL.....3
1.1. Introducción.....3
1.2. Planeación de los sistemas de transporte.....3
1.3. Modelos de transporte.....3
1.3.1. Generación de viajes.....5
1.3.2. Distribución de viajes.....5
1.3.3. Elección modal.....6
1.3.4. Asignación de viajes.....7
1.4. Asignación de viajes a la red bajo condiciones de congestionamiento.....8
1.4.1. Asignación de viajes para automóvil: caso determinístico.....11
1.4.2. Algoritmo de Frank-Wolfe.....16
1.5. Sistemas de información geográfica como herramienta de planeación.....21
1.5.1. gvSIG.....23
1.5.2. Casos de uso de gvSIG.....24
1.6. Resumen.....25
CAPÍTULO 2. METODOLOGÍA DE DESARROLLO DE SOFTWARE.....27
2.1. Introducción.....27
2.2. gvSIG versión 1.X.....27
2.2.1. Subsistema *gvSIG*.....28
2.2.2. Subsistema *FMap*.....29
2.2.3. Subsistema *Subdriver*.....29
2.3. Extensión de redes.....29

2.4. Creación de extensiones para gvSIG versión 1.X 31
2.5. Complemento Asignación por equilibrio 33
2.5.1. Archivo de configuración *config.xml* 34
2.5.2. Archivos de traducción *text_xx.properties* 37
2.5.3. Extensión *Asignación por equilibrio* 38
2.6. Resumen 52
CAPÍTULO 3. RESULTADO 55
3.1. Introducción 55
3.2. Planteamiento del problema ejemplo 55
3.3. Datos de entrada 56
3.3.1. Red 57
3.3.2. Orígenes y destinos de viajes 60
3.3.3. Costo a flujo libre y Nivel de servicio de los arcos 61
3.3.4. Matriz de viajes 62
3.4. Ejecución del algoritmo Frank-Wolfe 63
3.5. Visualización de asignación de viajes a la red 65
3.6. Comparación de soluciones 67
3.7. Resumen 69
CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES 71
4.1. Introducción 71
4.2. Conclusiones 71
4.3. Recomendaciones 72
REFERENCIAS 73
APÉNDICE A. CÓDIGO FUENTE 76
A.1. Clase AssignmentEquilibriumExtension.java 76
A.2. Clase AssignmentEqControlPanel.java 80
A.3. Clase AssignmentEqTask.java 89

ÍNDICE DE TABLAS

Tabla 1. Factores que influyen en la demanda del transporte 4
Tabla 2. Simbología de relación de cases en el diagrama UML 39
Tabla 3. Características de los arcos 56

ÍNDICE DE FIGURAS

Figura 1. Proceso de generación de viajes (Valencia) 5
Figura 2. Proceso de distribución de viajes (Valencia) 6
Figura 3. Proceso de elección modal (Valencia) 6
Figura 4. Proceso de asignación de viajes (Valencia) 7
Figura 5. Función de congestionamiento del arco 10
Figura 6. Método de bisección 19
Figura 7. Estructura de un SIG 22
Figura 8. Diagrama de flujo del algoritmo Frank-Wolfe 26
Figura 9. Estructura del directorio de una extensión, vista en el Entorno de Desarrollo Integrado Eclipse (Peñarrubia, 2010). 28
Figura 10. Esquema general de los componentes de la extensión de redes de gvSIG (Acevedo, 2011) 30
Figura 11. Proyectos básicos para trabajar con redes 32
Figura 12. Comandos de Eclipse IDE para navegar en el código fuente 33
Figura 13. Estructura del módulo Modelos de transporte/Asignación por equilibrio 34
Figura 14. Opción de menú Modelos de transporte/Asignación por Equilibrio 36
Figura 15. Menú, submenú e Interfaz gráfica de usuario en inglés, de la extensión Asignación por equilibrio 37
Figura 16. Diagrama de clases UML de la extensión Asignación por equilibrio 38
Figura 17. Representación UML de las clases de la extensión Asignación por equilibrio 40
Figura 18. Interfaz Gráfica de Usuario (GUI) de la extensión Asignación por equilibrio 44
Figura 19. Diagrama de flujo para el desarrollo de extensiones para gvSIG 53
Figura 20. Red de ejemplo 55
Figura 21. Datos de entrada para el módulo Asignación por equilibrio en gvSIG 57
Figura 22. Capa shape tipo línea, entidad gráfica de la red 58
Figura 23. Adición de campos a la Tabla de atributos de la capa de la red 58
Figura 24. Asignación de Costo a flujo libre (t_a^0) como Costo principal y Costo secundario 59
Figura 25. Generar capa de puntos con el Gestor de paradas 60
Figura 26. Capas de puntos Origen y Destino 61
Figura 27. Ingreso de Orígenes y Destinos de viajes, y Tolerancia de puntos a la red 61
Figura 28. Ingreso de las características de los arcos y Precisión de convergencia a la solución 62

Figura 29. Opciones de Matriz de viajes 62
Figura 30. Ingreso de Matriz de viajes y Directorio donde se guardará el resultado. 63
Figura 31. Botones para Cancelar o Iniciar la ejecución del algoritmo Frank-Wolfe 63
Figura 32. Archivo generado, parte inicial 64
Figura 33. Archivo generado, parte final 64
Figura 34. Tabla de atributos con valores de Demanda de arco actualizada 65
Figura 35. Configuración de las propiedades de simbología de la capa 66
Figura 36. Visualización gráfica de asignación de viajes a la red, $\varepsilon = \pm 0.01$ 66
Figura 37. Solución de asignación de viajes a la red del libro de Norbert Oppenheim 67
Figura 38. Solución del módulo Asignación por equilibrio, $\varepsilon = \pm 0.01$ 67
Figura 39. Solución del módulo Asignación por equilibrio, $\varepsilon = \pm 0.0001$ 68
Figura 40. Visualización gráfica de asignación de viajes a la red, $\varepsilon = \pm 0.0001$ 68
Figura 41. Diagrama de flujo para el uso del módulo Asignación por equilibrio 69

INTRODUCCIÓN

El tema de congestión vehicular es abordado desde niveles políticos hasta conversaciones personales día con día en la Ciudad de México, ya sea para gestionar la movilidad en la ciudad o para intercambiar observaciones de tiempos de recorrido. Es queja común, puesto que tiende a ir en aumento el impacto de sus efectos negativos: pérdida de tiempo de pasajeros y automovilistas, aumento de emisiones contaminantes, disminución de productividad, decremento de calidad de vida, por mencionar algunos.

La congestión empeora a medida que la capacidad de la red nunca coincidirá aumentar al mismo ritmo que lo hace la demanda. Por ello la labor del planeador de transporte ha pasado a ser del proveedor de carreteras y su aumento de capacidad, a la exploración de medios por los cuales la capacidad existente puede ser mejor utilizada y asignar prioridad de usuarios (Banister, 2002).

Los Sistemas de información geográfica son útiles en la presentación de la complejidad de la planeación del transporte, facilitando la comprensión de los problemas y tendencias subyacentes, con el propósito de ayudar en el proceso de toma de decisiones de alto nivel.

Planteamiento del problema

La necesidad de una aplicación de Sistemas de información geográfica de escritorio libre que apoye el proceso de planeación, gestión y administración pública de vialidades, ha dado origen al presente trabajo.

Justificación

Colaborar en el avance del desarrollo de software libre con un módulo para la asignación de viajes a la red, y al mismo tiempo, brindar una opción que apoye la independencia tecnológica a instituciones gubernamentales y educativas.

Objetivos

- Desarrollar un módulo para la Asignación de viajes a la red, programado en Java e incorporarlo al software libre gvSIG.
- Contribuir con una herramienta que permita el manejo de información para la planeación, gestión y administración de redes viales, con las ventajas de un software multiplataforma con licencia GNU/GPL.

CAPÍTULO 1. MARCO REFERENCIAL

1.1. INTRODUCCIÓN

En este capítulo se abordan los temas de modelos de transporte y Sistemas de información geográfica, como contexto del módulo de software sobre la asignación de viajes. Particularmente se expone la parte teórica del algoritmo de Frank-Wolfe implementado, así como las generalidades del programa de software gvSIG Desktop.

1.2. PLANEACIÓN DE LOS SISTEMAS DE TRANSPORTE

La ingeniería de transporte y la planeación del transporte están encaminadas hacia el diseño eficiente de infraestructura y servicios que satisfagan la necesidad de accesibilidad y movilidad. Muchos sistemas de transporte bien diseñados que satisfacen estas necesidades, se basan en la buena comprensión de la conducta humana. Dado que los sistemas de transporte son la columna vertebral que conecta las partes vitales de una ciudad, el conocimiento profundo de la naturaleza humana es esencial para la planificación, diseño y análisis operacional de los sistemas de transporte (Goulias, 2003).

Los sistemas de transporte son planeados y diseñados para ofrecer a la gente la capacidad de realizar actividades en los lugares y horarios de su preferencia, cuando no se cumple esta condición, se considera que el sistema de transporte proporciona un nivel de servicio pobre. Los modelos de transporte tienen como objetivo modelar y predecir dónde y cuándo tendrá lugar la demanda de viajes, de tal manera que el sistema de transporte puede ser planificado y diseñado para satisfacer la demanda proyectada de viajes y garantizar una alta calidad de vida para los habitantes de una región geográfica (Pendyala, 2003).

1.3. MODELOS DE TRANSPORTE

La demanda de transporte se refiere a la cantidad y tipo de viaje que la gente elige en determinadas condiciones, teniendo en cuenta factores como la calidad de las opciones de transporte disponibles y sus precios, así como otros listados en la tabla 1. La comprensión de la demanda es importante para la planeación del transporte en general y particularmente importante para la gestión de la demanda de transporte, que incluye diversas estrategias que influyen en el comportamiento del viajero (Victoria Transport Policy Institute, 2012).

Tabla 1. Factores que influyen en la demanda del transporte

Demográficos	Económicos	Precio	Opciones de Transporte	Calidad de Servicio	Uso de Suelo
Número de habitantes (residentes, trabajadores y visitantes). Ingresos. Edad/Ciclo de vida. Estilo de vida. Preferencias.	Número de trabajos. Ingresos. Actividad empresarial. Transporte de mercancías. Actividad turística.	Precios de combustible e impuestos. Impuestos a vehículos. Peajes. Tarifas de estacionamiento. Seguro de vehículos. Tarifas de transporte público.	A pie. Bicicleta. Transporte público. Auto compartido. Automóvil. Servicio de taxi. Trabajo a distancia. Servicio de abastecimiento.	Rapidez o demora relativa. Confiabilidad. Comodidad. Protección y seguridad. Condiciones de espera. Condiciones de estacionamiento. Información de usuario. Posición social.	Densidad. Mixto. Peatonal. Conectividad. Proximidad de servicio de tránsito. Diseño de vialidades.

La demanda de transporte tiene lugar en relación al espacio. Es la distribución de las actividades en el espacio lo que provoca la demanda del transporte. El enfoque más usual para tratar el espacio consiste en dividir el área de estudio en zonas y codificarlas, junto a redes de transporte, de una forma adecuada para su procesamiento y tratamiento con la ayuda de paquetes computacionales especializados, software (Ortúzar & Willumsen, 2008).

Los métodos de análisis que nos ayudan a entender y predecir el comportamiento humano, en cuanto a viajes se refiere, han sido desarrollados en las últimas décadas. En esta área, la investigación básica y aplicada ha sido muy activa en los métodos cuantitativos desde 1970, con un notable desarrollo: los modelos de demanda desagregados. Sin embargo, ha sido recientemente que se han experimentado los beneficios de la práctica de estos métodos. Inevitablemente, la asimilación del resultado de las investigaciones en la toma de decisiones de las políticas públicas es muy lento (Goulias, 2003).

Los modelos son representaciones simplificadas de la realidad que pueden ser utilizados para explorar las consecuencias de políticas o estrategias particulares. Éstos se han simplificado deliberadamente con el fin de mantenerlas viables y para evitar detalles superfluos, encapsulando las características importantes del sistema de interés (Bonsall, 2006).

El enfoque tradicional para el análisis y proyección de la demanda de viajes se ha apoyado en modelos que se basan en el cálculo de cuatro aspectos principales del comportamiento del viajero (Pendyala, 2003):

1. ¿Cuántos viajes se hacen? (generación de viajes)

2. ¿Dónde se realizan los viajes? (distribución de viajes)

3. ¿Por qué medios de transporte son realizados los viajes? (elección modal)

4. ¿En qué ruta o trayectoria son hechos los viajes? (asignación a la red)

La secuencia generación-distribución-elección modal-asignación es la más común pero no es la única posible, sin embargo, el modelo secuencial de las cuatro etapas constituye un punto de referencia respecto a métodos alternativos (Ortúzar & Willumsen, 2008).

1.3.1. Generación de viajes

Con datos obtenidos típicamente de una encuesta origen-destino, se pueden estimar modelos que permitan predecir el total de viajes generados y atraídos por cada zona (generación de viajes). Para modelar la generación, la práctica contempla el uso de información a nivel de hogares y técnicas de clasificación, regresión lineal múltiple o mixta. Para modelar la atracción de viajes, aún se considera aceptable utilizar datos a nivel zonal (información agregada). Se deben estimar modelos diferentes para los distintos propósitos de viaje que se estén considerando en el estudio (Ortúzar S., 2000).

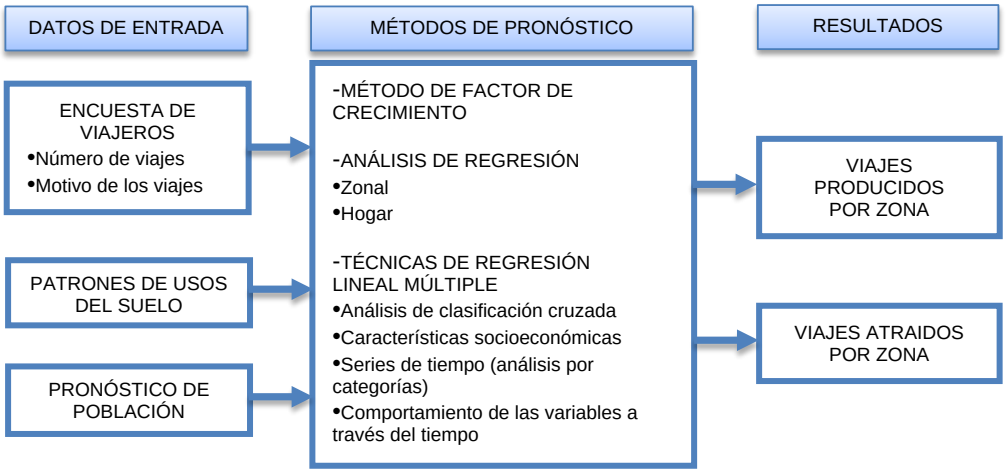


Figura 1. Proceso de generación de viajes (Valencia)

1.3.2. Distribución de viajes

La segunda etapa del modelo clásico consiste en distribuir los viajes generados por cada zona a sus posibles zonas de destino (distribución de viajes), a fin de producir matrices de viajes para los distintos propósitos modelados en los diversos periodos del día que se estén considerando (típicamente punta, fuera de punta y total diario). En esta tarea, los resultados del modelo de generación de viajes constituyen los totales a los que deben sumar, horizontal y verticalmente, las celas de la matriz. Además, se requiere información sobre los “costos generalizados” de viaje entre cada par de zonas para cada tipo de persona modelado (típicamente con y sin acceso al transporte privado) (Ortúzar S., 2000).

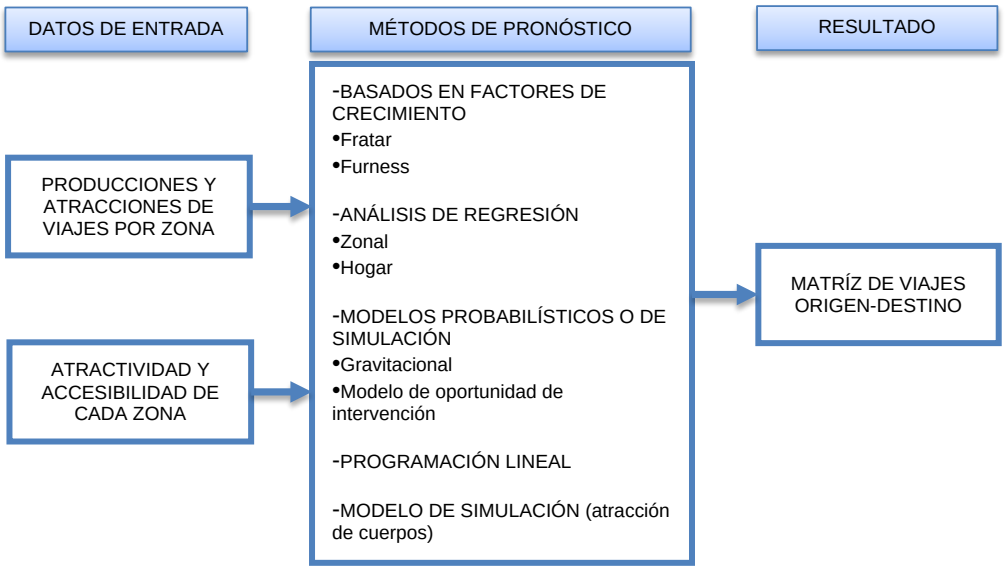


Figura 2. Proceso de distribución de viajes (Valencia)

1.3.3. Elección modal

La etapa que sigue normalmente, considera modelar cómo se reparten los viajes interzonales que predice el modelo anterior entre los distintos medios de transporte disponibles para cada tipo de persona (elección modal). Ésta es una de las áreas en las que se han producido mayores avances en los últimos años, particularmente a través del uso de las técnicas de elección discreta conducentes a los populares modelos desagregados de demanda (Ortúzar S., 2000).

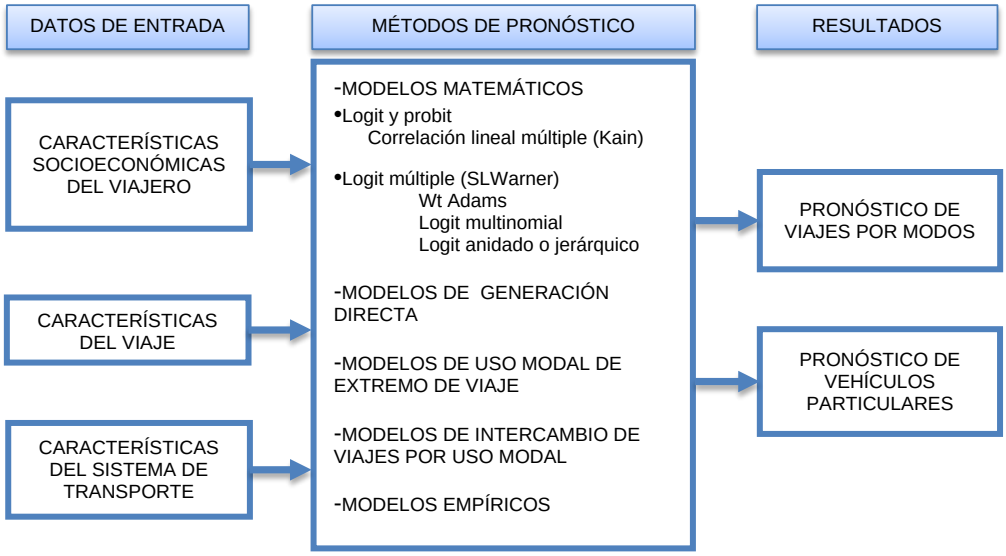


Figura 3. Proceso de elección modal (Valencia)

1.3.4. Asignación de viajes

La asignación es un proceso que permite la estimación del volumen de viajes en cada componente individual del sistema de transporte. Este proceso es comúnmente conocido como “carga” de la demanda a la red. En éste, al igual que en los procesos anteriores, pueden simularse volúmenes en el sistema existente o se pueden pronosticar volúmenes en sistemas alternativos futuros (Escobar).

La información necesaria para este proceso, se limita a los datos de la red vial y de transporte público, además de los pronósticos del número de viajes origen-destino, que se espera ocurrirán en automóviles particulares y transporte público, respectivamente. A partir de estos datos pueden obtenerse, tanto el volumen esperado de automóviles para cada tramo de la red vial, como el volumen de pasajeros en cada ruta y modalidad de transporte público (Escobar).

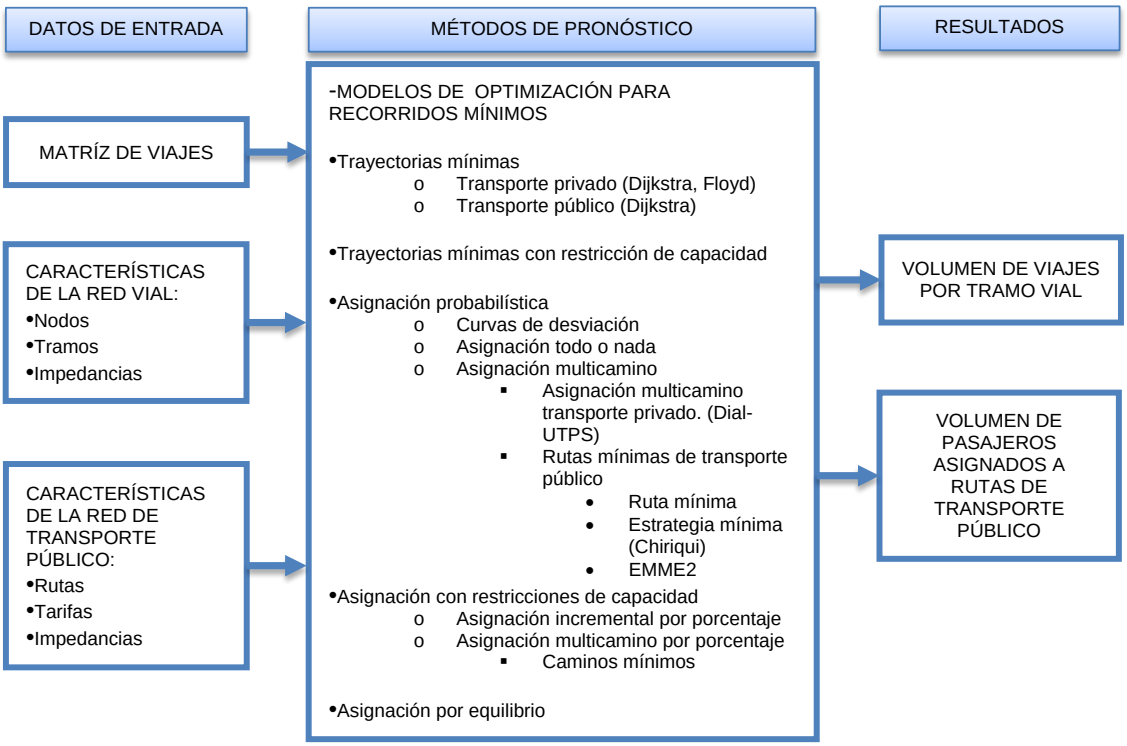


Figura 4. Proceso de asignación de viajes (Valencia)

El submodelo de asignación de viajes considera los aspectos de oferta y equilibrio oferta-demanda. En su versión original, sólo se encargaba de distribuir los viajes por cada medio de transporte entre las distintas rutas disponibles para cada par de zonas. Ahora bien, esta asignación se efectúa de modo de conseguir lo que se conoce como equilibrio de usuarios de acuerdo con el primer principio de Wardrop, que consiste en asegurar, que en el equilibrio, todas las rutas seleccionadas tengan un costo mínimo e igual entre sí. El

costo de cada ruta depende del flujo que circule por los arcos que la componen, ya que al existir congestión, cada nuevo usuario de un arco hará que la velocidad de circulación por el mismo disminuya (Ortúzar S., 2000).

El submodelo es, en la actualidad, de oferta y equilibrio, puesto que cumple una labor bastante más compleja que el mero equilibrio del tráfico anterior. El problema es que cuando se asigna la demanda a la red, sus arcos quedan cargados con flujos que no tienen por qué ser iguales a los supuestos previamente para obtener los costos iniciales. Por ende, los costos varían, lo que puede (y de hecho sucede cuando existe congestión) afectar el número de viajes por cada medio entre cada par de zonas (Ortúzar S., 2000).

La solución al problema de asignación de tráfico es relativamente difícil. La principal dificultad es que el costo del arco generalmente se incrementa en función no lineal del flujo en el arco. Resolver el problema de asignación de tráfico para redes a gran escala de interés práctico requiere de métodos de solución computacionalmente eficientes. Muchos métodos computacionales se encuentran disponibles para aproximar la solución de equilibrio. Todos estos métodos son iterativos, es decir, empiezan considerando alguna asignación inicial, se calculan los costos utilizando los flujos de la asignación considerada, se modifica la asignación y se actualizan los costos (Bar-Gera & Boyce, 2004).

La asignación de viajes a partir de matrices origen-destino ha sido una práctica útil en la planeación y administración de las redes viales. Específicamente el uso de modelos de asignación estática con el enfoque de equilibrio del usuario u óptimo para el sistema, permite obtener valiosos indicadores de desempeño, especialmente en un análisis macroscópico, muy apropiados para la toma de decisiones de movilidad y de gran aceptación en la comunidad científica (Londoño & Lozano, 2012).

Los resultados que se obtienen para cada arco de la red son: el flujo, la relación flujo/capacidad, las velocidades dadas las condiciones de tráfico; que pueden relacionarse, por ejemplo, con el nivel de servicio en el arco, con emisiones contaminantes por arco; asimismo, la asignación de viajes permite el análisis de impacto de nueva infraestructura vial y de equipamiento urbano (Lozano, 2012).

1.4. ASIGNACIÓN DE VIAJES A LA RED BAJO CONDICIONES DE CONGESTIONAMIENTO

El viajero individual es considerado un “consumidor de viajes”, el cual tiene varias alternativas que se interpretan como “versiones de un mismo producto”, por ejemplo viajar por una u otra ruta.

La utilidad de una ruta para automóviles particulares, es función de la demanda que tiene la ruta, ya que al aumentar los niveles de tráfico, se disminuye la velocidad del viaje, aumentan las dificultades en el traslado y se incrementa el tiempo de viaje. Lo mismo sucede con rutas de transporte público, cuando un gran número de pasajeros abordan y

bajan, pueden generar un tiempo de espera mayor, en la terminal o estación y en las paradas. De igual forma, la utilidad para un destino dado, es también afectada por el número de viajeros que se dirigen hacia él (Oppenheim, 1995).

En todos los casos, la ausencia o presencia de congestión, afecta la función de utilidad en la selección de viaje. Para el caso con congestión, la utilidad de los viajeros en auto particular, se puede ver afectada por las elecciones que realizan, ya que el comportamiento de un viajero es dependiente e influenciado por todos los demás viajeros. A este fenómeno se le conoce en microeconomía como externalidades de la demanda.

En presencia de congestionamiento, la determinación de los valores desconocidos de utilidad de las rutas, es consistente con el principio de maximizar la utilidad, que se usa en el caso sin congestionamiento de forma determinística o estocástica. Para reiterar, en equilibrio, ningún viajero puede unilateralmente incrementar la utilidad que recibe, por cambiar de ruta. Esto implica que las rutas usadas que conectan un origen y destino dados, ofrecen igual utilidad máxima y todas las rutas no usadas ofrecen menor utilidad.

La utilidad de las rutas se puede definir de igual forma que para el caso no congestionado, como

$$\tilde{V}_{ijr} = b_{ij} - c_{ijr} - \tau t^*_{ijr} + \epsilon_{ijr} = b_{ij} - g^*_{ijr} + \epsilon_{ijr}; \qquad \forall\ i,\ j,\ r$$

Donde los valores de t^*_{ijr} o g^*_{ijr} ahora son desconocidos y deben determinar sus valores de equilibrio.

Y esta utilidad, también se puede establecer en función de las utilidades de los arcos que forman la ruta:

$$\tilde{V}_{ijr} = b_{ij} - \sum_a \left(g_a(v^*_a) + \epsilon_{ijr}\right)\delta^a_{ijr}\,; \qquad \forall\ i,\ j,\ r$$

Donde

$$g_a(v_a) = \tau t_a(v_a) + c_a; \qquad \forall\ a$$

Y el tiempo de viaje en los arcos t_a es función de la demanda en los arcos v_a, como se especifica por la función de congestionamiento del arco, función del desempeño del arco o simplemente la función de costo del arco $t_a(\,)$.

En general, se tienen diversas funciones de tiempo de viaje que pueden ser usadas, la más común es la función para autos del U. S. Bureau of Public Road, 1964:

$$t_a(x_a) = t_a^0\left[1 + 0.15\left(\frac{v_a}{K_a}\right)^4\right]; \qquad \forall\ a$$

Donde

t_a^0: Es el tiempo de viaje a flujo libre, es el costo (mínimo) de viaje a volumen cero.

K_a: Es el nivel de servicio en el arco. Puede ser igual al flujo en el arco cuando, el tiempo de viaje es 15% mayor al tiempo de viaje a flujo cero.

La gráfica de la función de desempeño de un arco, se puede representar con la siguiente figura.

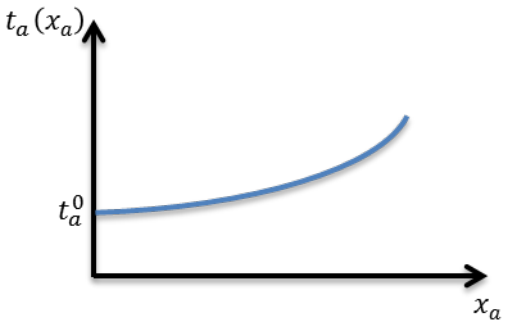


Figura 5. Función de congestionamiento del arco

Es importante notar que la función de costo generalizada del arco, debe estrictamente aumentar con la demanda en el arco, para garantizar la unicidad de la demanda en las rutas.

Específicamente se requiere que

$$\frac{\partial g_a(v_a)}{\partial\, v_a} > 0; \qquad \forall\ a, v_a$$

Cómo el término c_a, que representa el costo por usar el arco, es constante, y la función de costo del arco $t_a(\,)$, es estrictamente creciente con el volumen en el arco, entonces la condición anterior siempre debe ser establecida.

De forma más general, la función de desempeño de un arco puede ser también, función de la demanda en otros arcos. Por ejemplo, la velocidad de viaje en una dirección, puede ser afectada por el volumen de vehículos, en dirección opuesta.

Es más, cuando las redes modales interactúan o cuando el transporte público y los autos particulares usan la misma red vial, cada tiempo de viaje modal puede ser función de la demanda de los otros modos.

1.4.1. Asignación de viajes para automóvil: caso determinístico

Para el caso determinístico, la utilidad que recibe un viajero por viajar de i a j, usando la ruta r está dada por

$$\tilde{V}_{ijr} = b_{ij} - \tau t^*_{ijr} - c_{ijr}; \qquad \forall\ i,j,r$$

Donde ahora t^*_{ijr} es desconocido y es considerado el valor de equilibrio del tiempo de viaje de la ruta, determinado por la demanda de la ruta T_{ijr} en equilibrio.

Debido a la presencia de congestionamiento sobre la red, la demanda agregada de la ruta, no se puede obtener como simple adición de las demandas individuales.

Además de que, la selección de la ruta por los viajeros individuales es interdependiente, ya que la utilidad de la ruta es función de esta selección y del volumen de la misma.

Y la demanda agregada de la ruta se establece como

$$T^*_{ijr} = \begin{cases} T_{ij} & \text{si} \quad g_{ijr} = g^*_{ijr} = \min_k g_{ijk} \\ 0 & \text{si} \quad g_{ijr} > \min_k g_{ijk} \end{cases}; \qquad \forall\ i,j,r$$

Esto es, la ruta entre i y j con la máxima utilidad, captura toda la demanda origen-destino.

Ahora en términos de probabilidad se tiene

$$P_{ijr} = \begin{cases} 1 & \text{si} \quad g_{ijr} = g^*_{ijr} \\ 0 & \text{si} \quad g_{ijr} > g^*_{ijr} \end{cases}; \qquad \forall\ i,j,r$$

La diferencia con el caso no congestionado, es que la utilidad de la ruta $\tilde{V}_{ijr}$, ahora es desconocida ya que los t_{ijr} son una variable.

La utilidad directa por la elección de la ruta con presencia de externalidades en la red, está dada por

$$U_R = -\tau \sum_a \int_0^{v_a} t_a(v)\, dv + T_0$$

Donde

$$v_a = \sum_i \sum_j \sum_r \left(T_{ijr} + S_{ijr}\right)\delta^a_{ijr}\,; \qquad \forall\ a$$

S_{ijr}: Es la demanda de la ruta, correspondiente a todo propósito de viaje.

$t_a(\,)$: Es la función de tiempo de viaje en el arco.

La incorporación del término S_{ijr} es para considerar el hecho de que la congestión es creada por los viajeros de todos los propósitos. Sin embargo, desde el punto de vista del viajero representativo (RT, Representative Traveler), esos términos son dados y resultan de selecciones conocidas por otros RTs, que representan viajeros con otros propósitos. Si las utilidades de estos segundos RTs son las mismas que los primeros, entonces ellos harán la misma selección de ruta, y las demandas origen-destino S_{ijr} dadas se pueden simplemente aumentar a las T_{ijr}, en la determinación del equilibrio utilidades/tiempos de viaje. Si no, el problema de la ruta involucra a dos diferentes usuarios de la red y llega a ser más compleja.

En consecuencia, la maximización de la utilidad del problema de selección de ruta, del viajero representativo es

$$\underset{(T_{ijr},T_0)}{\text{Max}}\ U_R = -\tau \sum_a \int_0^{\sum_i \sum_j \sum_r (T_{ijr}+S_{ijr})\delta_{ijr}^a} t_a(v)dv + T_0$$

s. a.

$$\sum_i \sum_j \sum_r T_{ijr} c_{ijr} + T_0 = B;$$

$$\sum_r T_{ijr} = T_{ij}; \qquad \forall\ i,j$$

$$T_{ijr} \geq 0; \quad T_0 \geq 0; \qquad \forall\ i,j,r$$

Cambiando el signo de la función objetivo, también se puede escribir como un problema de minimización, después de eliminar el término constante B.

$$\min_{(T_{ijr})} U'_R = \tau \sum_a \int_0^{\sum_i \sum_j \sum_r (T_{ijr}+S_{ijr})\delta_{ijr}^a} t_a(v)dv + \sum_i \sum_j \sum_r T_{ijr} c_{ijr}$$

O también, como

$$\min_{(T_{ijr})} U'_R = \sum_a \int_0^{v_a} g_a(v)dv$$

s. a.

$$\sum_r T_{ijr} = T_{ij}; \qquad \forall\ i,j$$

$$T_{ijr} \geq 0; \qquad \forall\ i,j,r$$

Donde

U'_R: Representa la desutilidad del viajero representativo.

$\sum_r T_{ijr} = T_{ij}$: Indica el balance de la demanda requerida.

$T_{ijr} \geq 0$: Indica que todas las demandas deben ser no negativas.

Para probar que la solución del problema anterior produce la demanda agregada, las condiciones de Karush-Kuhn-Tucker (KKT) son establecidas, con respecto a la demanda T_{ijr} de la ruta.

Estas condiciones son

$$\frac{\partial U'_R}{\partial T_{ijr}} - \sum_i \sum_j \mu_{ij} \frac{\partial}{\partial T_{ijr}} \left\{ \sum_r T_{ijr} - T_{ij} \right\} \geq 0 \, ; \qquad \forall \; i,j,r$$

$$T_{ijr} \left[\frac{\partial U'_R}{\partial T_{ijr}} - \sum_i \sum_j \mu_{ij} \frac{\partial}{\partial T_{ijr}} \left\{ \sum_r T_{ijr} - T_{ij} \right\} \right] = 0; \qquad \forall \; i,j,r$$

Donde

μ_{ij}: Son las variables duales para las restricciones $\sum_r T_{ijr} = T_{ij}$

T_{ij}: Son las constantes dadas.

Calculando las derivadas parciales de U'_R con respecto a las demandas de la ruta T_{ijr}, usando la regla de la cadena, tenemos

$$\frac{\partial U'_R}{\partial T_{ijr}} = \sum_a \frac{\partial U'_R}{\partial v_a} \cdot \frac{\partial v_a}{\partial T_{ijr}} = \sum_a g_a(v_a) \frac{\partial v_a}{\partial T_{ijr}} = \sum_a g_a(v_a) \delta^a_{ijr} \, ; \qquad \forall \; i,j,r$$

Recordando que $t_a()$ es la función de tiempo de viaje del arco a, y dada la definición de δ^a_{ijr}, el término $\sum_a g_a(v_a)\delta^a_{ijr}$ representa la suma de los costos de viaje sobre los arcos que componen la ruta r entre el origen i y el destino j, y por lo tanto, es igual al costo de viaje g^*_{ijr} sobre esa la ruta.

Entonces las condiciones KKT se puede escribir como

$$g^*_{ijr} \geq \mu_{ij}; \qquad \forall\ i,j,r$$

$$T^*_{ijr}\left(g^*_{ijr} - \mu_{ij}\right) = 0; \qquad \forall\ i,j,r$$

La restricción $g^*_{ijr} \geq \mu_{ij}$ implica que μ_{ij} es más pequeño que el costo generalizado de cualquier ruta r entre i y j.

La restricción $T^*_{ijr}\left(g^*_{ijr} - \mu_{ij}\right) = 0$ implica que T_{ijr} es positivo, si la ruta r entre i y j es seleccionada, entonces $g^*_{ijr} - \mu_{ij}$ será cero, lo cual se debe a que μ_{ij} representa el costo generalizado de la ruta, que es entonces el costo mínimo entre i y j.

De otra forma, si la ruta no es seleccionada, si $T_{ijr} = 0$, entonces la restricción $g^*_{ijr} \geq \mu_{ij}$ es operativa, y en consecuencia el costo generalizado de la ruta, es mayor o igual que, el costo mínimo generalizado.

Esto establece que

$$T^*_{ijr} = 0; \quad \text{si} \quad g^*_{ijr} > \min_k\left(g_{ijk}\right); \qquad \forall\ i,j,r$$

$$T^*_{ijr} \geq 0; \quad \text{si} \quad g^*_{ijr} = \min_k\left(g_{ijk}\right); \qquad \forall\ i,j,r$$

Y estas relaciones indican la demanda agregada.

Entonces, la utilidad que recibe el viajero representativo por la selección una ruta con presencia de externalidades en la red, es igual al valor óptimo de la función de utilidad directa, dada por

$$\widetilde{W}_R = B - \tau \sum_a \int_0^{v^*_a} t_a(v)dv - \sum_i \sum_j \sum_r T^*_{ijr} c_{ijr} = B - \sum_a \int_0^{v^*_a} g_a(v)dv$$

Donde

v_a^*: Son los volúmenes de arco en equilibrio.

T_{ijr}^*: Son las demandas de ruta.

Y por las externalidades, la utilidad agregada recibida no es igual, a la simple suma de las utilidades individuales recibidas.

Además se puede observar que el "efecto negativo" de la congestión sobre la utilidad del viajero representativo, o sobre la función de bienestar comunitaria, se mide por el término

$$-\tau\left[\sum_a \int_0^{v_a^*} t_a(v)dv - \sum_i \sum_j \sum_r T_{ijr}^* t_{ijr}^*\right]$$

La ecuación del viajero superfluo para la selección de ruta determinística con presencia de externalidades en la red, se establece como

$$TS_R = -\tau \sum_a \int_0^{v_a^*} t_a(v)dv - \sum_i \sum_j \sum_r T_{ijr}^* c_{ijr} = -\sum_a \int_0^{v_a^*} g_a(v)dv$$

1.4.2. Algoritmo de Frank-Wolfe

En el caso del problema de maximización de la utilidad del viajero representativo, la función objetivo del problema es convexa y las restricciones son lineales, de modo que definen una región factible convexa, donde es posible usar varios algoritmos de solución.

Un algoritmo muy eficiente, es el llamado "Método de combinaciones convexas", también conocido como "Algoritmo de Frank-Wolfe", el cual se basa en hacer una aproximación lineal de la función objetivo, que puede considerarse como una generalización de la técnica de "descenso factible". El principio del método es el siguiente.

En la iteración k del algoritmo, el valor de la función objetivo $U'_R(v)$

$$\min_{(T_{ijr})} U'_R = \tau \sum_a \int_0^{\sum_i \sum_j \sum_r (T_{ijr}+S_{ijr})\delta_{ijr}^a} t_a(v)dv + \sum_i \sum_j \sum_r T_{ijr} c_{ijr}$$

Es aproximada por una función lineal en las variables auxiliares y_a.

$$U'_R(y^k) = U'_R(v^k) + \sum_a \left[\frac{U'_R(v)}{\partial v_a}\right]_k (y_a^k - v_a^k)$$

El problema original es remplazando entonces con un auxiliar o subproblema, esto es

$$\min U'_R(y^k) = \min \left\{ U'_R(v^k) + \sum_a \left[\frac{\partial U'_R(v)}{\partial v_a}\right]_k (y_a^k - v_a^k) \right\}$$

Donde las demandas de arco v_a^k son conocidas y sujetas a las restricciones $\sum_r T_{ijr} = T_{ijr}$ y $T_{ijr} \geq 0$, en las variables auxiliares Z.

En esta función objetivo las variables v_a^k son conocidas, se pueden eliminar y sólo quedar las variables y_j.

Por lo tanto, el primer término $U'_M(v^k)$ puede ser eliminado, así como todos los términos de la forma

$$\left[\frac{\partial U'_R(v)}{\partial v_a}\right]_k v_a^k$$

En consecuencia, el problema auxiliar puede ser escrito como

$$\min_{y} f(y) = \left\{ \sum_{a} \left[\frac{\partial U_R'(v)}{\partial v_a} \right]_k y_a^k \right\}$$

s. a.

$$\sum_{r} Z_{ijr} = T_{ij}; \qquad \forall\ i,j$$

$$Z_{ijr} \geq 0; \qquad \forall\ i,j,r$$

Dado que el valor del término $[\partial\, U_R'(v)/\partial v_a]$ es conocido en la k-ésima iteración, entonces la función objetivo $f(y)$ queda como una combinación lineal de los valores desconocidos y_j.

Entonces, el problema no lineal es aproximado mediante un problema lineal. Su solución y^k puede ser mostrada para definir el vector de la solución actual v^k, para el punto auxiliar de solución y^k, a través del cual existe la mayor razón de decremento de la función objetivo $U_R'(v^k)$, desde su valor actual.

Una vez que se ha identificado esta dirección para resolver del problema anterior, la mejor estimación para la solución del problema original, se encuentra entre los puntos v^k y y^k. Esto debido a que y^k es la solución de un problema lineal, por lo que automáticamente se encuentra en la frontera de la región factible, en la intersección de dos o más restricciones. La posición de la solución del problema no lineal original, es decir, el tamaño del movimiento a través de la dirección de la solución v^k, puede ser determinada, al resolver el problema

$$\min_{\theta} U_R' \left(v^k + \theta\left(y^k - v^k\right) \right)$$

s. a.

$$0 \leq \theta \leq 1$$

Que es un problema de optimización uno-dimensional en θ, cuya región factible es un rango de valores, en este caso de 0 a 1.

El problema se puede resolver usando el "Algoritmo de Bisección", ya que $U_R'(\theta)$ es una función continua de un solo valor, cuyo máximo se encuentra en el intervalo 0 y 1.

El método se ilustra en la siguiente figura.

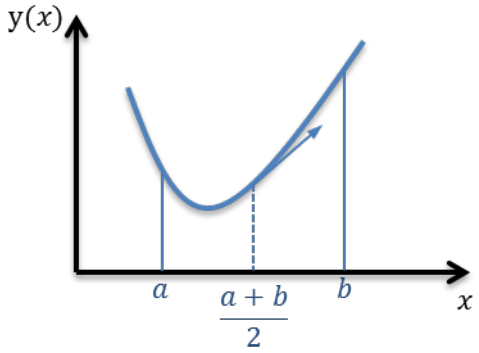


Figura 6. Método de bisección

Su principio es muy simple. Si para un punto dado θ, la derivada de $U_R'(\theta)$ es negativa, implica que el valor mínimo de $U_R'(\theta)$ se encuentra a la derecha del punto, tal que, el intervalo de la izquierda del punto puede ser eliminado.

De la misma forma, si la derivada de $U_R'(\theta)$ es positiva, implica que el valor mínimo de $U_R'(\theta)$ se encuentra a la izquierda del punto, tal que, el intervalo de la derecha se puede eliminar.

El algoritmo de Bisección para minimización, se puede describir como sigue:

Actualizar el contador de iteraciones en uno.

Hacer $\theta = (a + b)/2$, (punto medio del intervalo actual $[a, b]$)

Calcular el valor de la derivada $F(\theta) = (dU_R'/d\theta)$

Si $F(\theta) \leq 0$ hacer $\theta = a$, cota inferior actualizada del intervalo.

Si $F(\theta) \geq 0$ hacer $\theta = b$, cota superior actualizada del intervalo.

Si $b - a \leq \varepsilon$, terminar. Una solución estimada como $\theta = (a + b)/2$, se ha encontrado, el punto medio del intervalo actual $[a, b]$.

Si $b - a > \varepsilon$, iterar; esto es, ir al inicio de la etapa iterativa.

Después de n iteraciones, la amplitud del intervalo actual en donde se estima que se encuentra la solución es igual a $(b_0 - a_0)/2^n$. Esta relación puede ser usada para determinar el número de iteraciones requeridas.

La expresión

$$F(\theta) = \frac{dU'_R}{d\theta} = \sum_a (y_a - v_a) g_a\big(v_a - \theta(y_a - v_a)\big)$$

Puede ser usada directamente en la aplicación del algoritmo del método de bisección, así se evita la estimación numérica del valor de $U'_R(\theta)$ y la derivada de θ en cada iteración.

Una vez encontrado el valor de θ, la nueva solución actualizada al problema no linear se estima como

$$v_a^{k+1} = v_a^k + \theta\big(y_a^k - v_a^k\big); \qquad \forall\ a$$

Donde v_a^{k+1} es la demanda de arco actualizada.

La gran ventaja de usar el método de linealización para resolver problemas de equilibrio en la red, es que el problema lineal auxiliar se convierte en un simple problema de Ruta mínima. Ya que las demandas de ruta T_{ijr} pueden ser usadas como variables alternativas en lugar de las demandas de arco v_a, se puede expresar la función objetivo del problema linealizado en términos de las demandas de las rutas auxiliares Z_{ijr} en vez de las demandas de los arcos auxiliares y_a.

Como se mencionó anteriormente, las derivadas de la función objetivo en términos de las demandas de ruta son igual a

$$\frac{U'_R}{\partial\, T_{ijr}} = \sum_a (\tau t_a(v_a) + c_a)\delta_{ijr}^a = \sum_a g_a(v_a)\delta_{ijr}^a = g_{ijr}; \qquad \forall\ i,j,r$$

Puede reconocerse que esta cantidad es igual al costo generalizado de viaje entre i y j en la ruta r. Por consiguiente, la función objetivo linealizada, para este problema, en la iteración k, se puede escribir como

$$\min_{Z}\left\{\sum_{i}\sum_{j}\sum_{r}\left[\frac{\partial U'_{R}(T)}{\partial T_{ijr}}\right]_{k} Z_{ijr}^{k}\right\} = \sum_{i}\sum_{j}\sum_{r} g_{ijr}^{k} Z_{ijr}^{k}$$

Donde Z_{ijr} representa la demanda auxiliar en la ruta r entre i y j. Es ésta la función objetivo que representa el costo total generalizado incurrido por todos los viajeros entre cada punto i y j. Este costo total puede ser minimizado cuando todos los viajeros entre cualquier punto dado i y j, sean asignados a la Ruta mínima generalizada entre estos dos puntos. De esta manera, la solución al problema lineal se puede obtener mediante la identificación de la Ruta mínima entre dos puntos cualesquiera de la red (Oppenheim, 1995).

1.5. SISTEMAS DE INFORMACIÓN GEOGRÁFICA COMO HERRAMIENTA DE PLANEACIÓN

Durante las últimas décadas, en la planeación urbana, los productos software fueron en general utilizados para el mapeo, el almacenamiento de la información y, en menor medida, para el análisis. Ahora, frente a la evolución de las técnicas informáticas, la planeación urbana convencional puede ser totalmente reformulada (Laurini, 2001).

Los últimos avances en tecnologías de información han aumentado la producción, recopilación y difusión de los datos geográficos, lo que favorece el diseño y desarrollo de Sistemas de Información Geográfica (SIG). Hoy en día, los SIG se están convirtiendo en una infraestructura de información común, que penetran cada vez más en aspectos de nuestra sociedad (Bauzer & Zimányi, 2009). En la siguiente figura se muestra la estructura de un SIG.

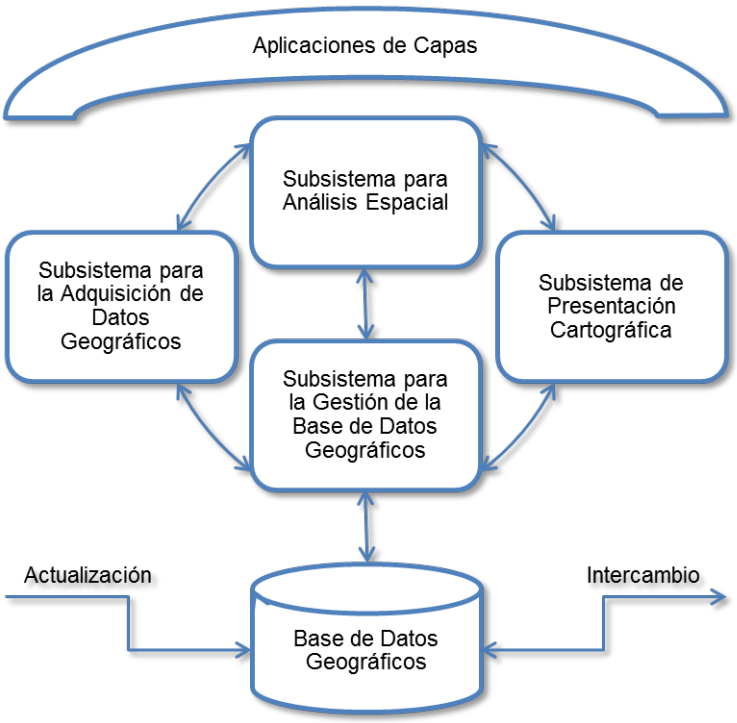


Figura 7. Estructura de un SIG

El objetivo de cualquier aplicación de un Sistema de información geográfica es proporcionar información para apoyar la planificación y la gestión. Como la información pretende reducir la incertidumbre en la toma de decisiones, los errores e incertidumbres en las bases de datos espaciales y los datos de salida de los productos SIG pueden tener repercusiones prácticas, financieras e incluso legales, para el usuario. Por estas razones, las personas involucradas en la adquisición y el procesamiento de datos espaciales deben ser capaces de evaluar la calidad de los datos básicos y los productos de información derivados (de Bay, 2001).

Un SIG es una representación espacial, o modelo, de los datos usados para representar una porción de la superficie de la tierra. En el contexto del transporte, hay tres clases relevantes de modelos de SIG:

- Modelos de campo, o representación de la variación continua de un fenómeno sobre el espacio. La elevación del terreno usa este modelo.
- Modelos discretos, según el cual las entidades discretas (puntos, líneas o polígonos) llenan el espacio. Áreas de descanso en autopistas, barreras de peaje, áreas urbanizadas pueden usar este modelo.
- Modelos de red para representar topológicamente entidades lineales conectadas (como son carreteras, líneas ferroviarias, o rutas aéreas) que son ubicadas en la superficie de referencia continua.

Mientras que los tres modelos pueden ser útiles en el transporte, el modelo de red conformado por los conceptos de arco y nodo juega el papel más importante en

aplicaciones de este ámbito, debido a que la infraestructura de red mono y multimodal son vitales para permitir y apoyar el movimiento de pasajeros y mercancías. De hecho, muchas aplicaciones para transporte sólo requieren del modelo de red para representar los datos (Thill, 2000). Ejemplos de tales aplicaciones son:

- pavimento y otros sistemas de gestión de instalaciones;
- procedimientos de ruteo en tiempo real y fuera de línea, incluyendo el envío de vehículos de emergencia y la asignación de tráfico en el proceso de planeación de transporte urbano de los cuatro pasos;
- sistemas de información del tráfico basado en la web y la planeación de viajes motorizados;
- sistemas de navegación en vehículos particulares;
- gestión de la congestión en tiempo real y detección de accidentes.

El acrónimo SIG-T es a menudo empleado para referirse a la aplicación de los SIG en la investigación, la planeación y la gestión en el transporte.

1.5.1. gvSIG

gvSIG es un proyecto que nace en 2002 con la tarea de apoyar la migración a software libre de los sistemas informáticos de la Conselleria de Infraestructuras y Transporte de la Generalitat Valenciana. Sus principales aplicaciones son gvSIG Desktop y gvSIG Mobile.

gvSIG Desktop, abreviadamente gvSIG, fue el primer desarrollo del proyecto Generalitat Valenciana Sistema de Información Geográfica. La primera versión de gvSIG que se publicó fue la 0.2 en octubre de 2004; durante el desarrollo del proyecto han ido publicándose constantemente nuevas versiones con nuevas funcionalidades, hasta llegar a las actuales, la 1.12 y 2.0.

Es un software libre diseñado para el manejo de información geográfica (Asociación gvSIG). Entre sus características se tiene que es:

- portable: funciona en diferentes plataformas por estar programado en java;
- modular: puede ser ampliado con nuevas funcionalidades mediante el desarrollo de extensiones;
- de código abierto: se distribuye bajo Licencia Pública General de GNU o sistema operativo totalmente libre, que garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el software;
- interoperable: es compatible con formatos de software privado;
- internacionalizable: permite la incorporación de nuevos idiomas con facilidad;
- sujeto a estándares: sigue las directrices marcadas por el Open Geospatial Consortium.

En gvSIG Desktop encontramos las herramientas propias de un completo cliente SIG de escritorio (Carrera, 2013), entre otras:

- Formatos raster y vectoriales soportados.
- Navegación
- Consulta
- Selección
- Geoprocesos
- Edición gráfica
- Edición alfanumérica
- Servicio de catálogo y nomenclátor.
- Representación vectorial
- Representación raster
- Etiquetado
- Tablas
- Constructor de mapas
- Impresión
- Redes
- Raster y teledetección
- Publicación
- 3D y animación
- Topología
- Otros

La ventaja como software libre, es que puede ser incorporado como herramienta de trabajo en instituciones públicas, adaptarlo a necesidades específicas y tener independencia tecnológica.

1.5.2. Casos de uso de gvSIG

En México, gvSIG se ha empleado en la educación, administración pública, estudios ambientales, agricultura, ganadería y pesca, e industria y comercio (gvSIG Outreach, 2013). Específicamente algunos proyectos han sido:

- Visor de mapas cartográficos en el Instituto Nacional de Ecología y Cambio Climático. INECC, México.
- Utilización de gvSIG en el Módulos de SIG-GPS para la Formación de Técnicos Certificados en Agricultura de Conservación. CIMMYT, México.
- Experiencia de desarrollo de aplicaciones de edición cartográfica con plugins de gvSIG. Catastro del Estado de Sinaloa, México.
- gvSIG y el Programa Federal “Hábitat”. SEDESOL, México.

A nivel internacional, gvSIG también ha hecho posible proyectos relacionados con el transporte:

- Los SIG en la Optimización de la Red de Transporte. El caso de Servicios Municipales de Transporte Público de Barreiro. Universidad de Lisboa, Portugal.
- Determinación de factor de sinuosidad para la estimación de tiempos por arco, a partir de un modelo de velocidades según tipo de carpeta en una red vial. Subsecretaría de Desarrollo Regional y Administrativo, Chile.
- gvSIG para la gestión de información de carreteras. Conselleria de Infraestructuras y Transporte, Generalitat Valenciana, España.
- gvSIG BATOVI: un recurso educativo para el Plan Ceibal. Ministerio de Transporte y Obras Públicas, Uruguay.
- gvBUS: gvSIG y transporte público. Conselleria de Infraestructuras, Territorio y Medio Ambiente de Valencia, España.
- Creación de aplicaciones empresariales sobre gvSIG: un caso de estudio de gestión con SIG de una red de autobuses para el Ministerio de Transporte de Argelia. Alkante, Francia.

1.6. RESUMEN

Los Sistemas de información geográfica están generado un replanteamiento de las formas convencionales de planeación, administración pública, educación, por mencionar algunas.

gvSIG al ser un software libre, está expandiendo su uso, debido a que puede ser modificado de acuerdo a las necesidades específicas de cada usuario y para propósitos muy variados.

El algoritmo de Frank-Wolfe es útil para el análisis de redes viales, impacto ambiental, impacto en nueva infraestructura vial y de equipamiento urbano. Se describe de manera general con el siguiente diagrama de flujo.

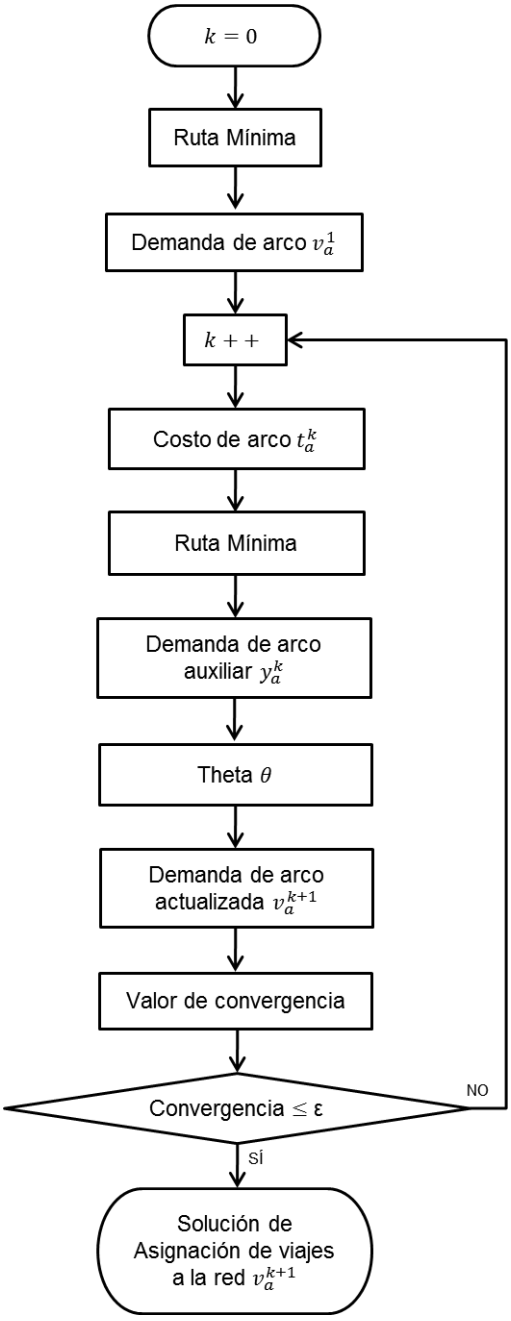


Figura 8. Diagrama de flujo del algoritmo Frank-Wolfe

CAPÍTULO 2. METODOLOGÍA DE DESARROLLO DE SOFTWARE

2.1. INTRODUCCIÓN

La primera parte de éste capítulo trata de gvSIG abordado desde la perspectiva del desarrollador de software, donde se expone la estructura general del código fuente para la versión 1.X de gvSIG. En los siguientes apartados se hace una descripción breve de la extensión de redes y de los recursos que hay en internet sobre la creación de extensiones para gvSIG versión 1.X. Por último, se explican partes de código fuente de la extensión Asignación por equilibrio, principalmente las relacionadas con el algoritmo de Frank-Wolfe.

No se abordan las características de gvSIG versión 2.X, principalmente porque aún no cuenta con la funcionalidad de análisis de redes, además de que ha modificado su arquitectura con el objetivo de mejorar su modularidad, facilitar el mantenimiento y la evolución de su tecnología.

2.2. GVSIG VERSIÓN 1.X

gvSIG ha sido programado en Eclipse IDE, un programa de cómputo provisto de un conjunto herramientas para programar; compuesto principalmente por un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Se puede utilizar otro entorno de desarrollo, pero en los manuales todos los ejemplos se han hecho con Eclipse, al igual que gvSIG al completo (Peñarrubia, 2010).

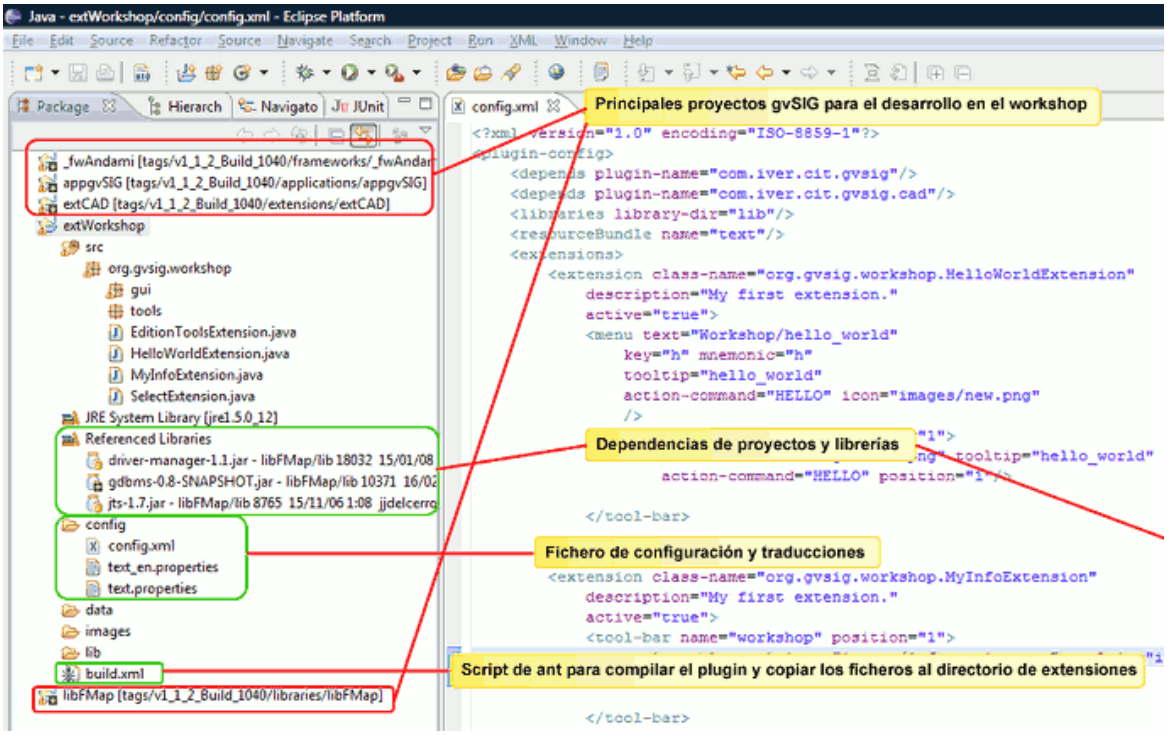


Figura 9. Estructura del directorio de una extensión, vista en el Entorno de Desarrollo Integrado Eclipse (Peñarrubia, 2010).

Una de las características de gvSIG es su modularidad, lo que permite ampliar sus funcionalidades sin tocar el código original. En gvSIG todo son extensiones, puesto que se estructura sobre un núcleo de funcionalidad relativamente pequeño, que contiene las partes esenciales del sistema, básicamente la gestión de ventanas y la carga de complementos (plug-ins) basada en archivos de configuración.

Los complementos se integran a la estructura de soporte de gvSIG mediante las extensiones, que son un conjunto de clases Java que añaden nuevas funcionalidades a la aplicación (Gavarró, 2011).

La plataforma de gvSIG está conformada en su núcleo por tres subsistemas (del Cerro, 2011): *gvSIG*, *FMap* y *Subdriver*.

2.2.1. Subsistema *gvSIG*

El subsistema *gvSIG* representa la parte visual de la aplicación. Está integrado por:

- *Andami*: gestor gráfico. Es la estructura de soporte sobre la que se construye gvSIG. Inicia la ejecución, realiza la carga de complementos e inicializa todos los subsistemas de la aplicación (Acevedo, 2010).
- *Project*: contenedor de la estructura de documentos.
- *Documents*: vistas, tablas y mapas.

- *View*: gráfica de cartografía y leyenda en la Tabla de Contenidos.
- *Layout*: vista apta para imprimir.
- *Table*: gráfica de datos alfanuméricos.
- *Layers*: capas que pueden insertarse en una vista.

2.2.2. Subsistema *FMap*

FMap proporciona un control de interfaz de usuario y herramientas para ello. Se compone de:

- *MapControl*: dibuja y mantiene la herramienta actual.
- *MapContext*: contexto de la parte gráfica.
- *Behavior*: controla el dibujado y el iniciador de los eventos de las herramientas.
- *Listeners*: gestionan eventos de las herramientas.
- *Layer*: contiene características de la capa.
- *Geometrías*: tipos de elementos gráficos que pueden representarse en una capa.
- *DataSources* y *drivers*: métodos para gestión de datos gráficos y alfanuméricos.

2.2.3. Subsistema *Subdriver*

Recupera los datos directamente de la fuente, enlazando a la aplicación con los datos. Lo conforman:

- *RemoteServices*: unifica acceso a datos remotos.
- *Drivers*: gestiona distintos tipos de datos.
- *DriverManager*: carga y acceso a drivers.
- *Writemanager*: carga y acceso a writers.
- *Writers*: operaciones de escritura.
- *DataSources*: acceso a datos alfanuméricos.
- *VectorialSources*: acceso a datos geométricos.
- *RasterSources*: acceso a datos raster.

2.3. EXTENSIÓN DE REDES

La extensión de redes es un complemento que proporciona a gvSIG las funcionalidades necesarias para el cálculo de caminos mínimos, optimización del orden de las paradas en una ruta, áreas de servicio, localización del evento más cercano, matrices de distancias, árbol de recubrimiento mínimo, conectividad; aplicando sobre una capa de ejes interconectados diversos algoritmos, con el fin de cubrir algunas necesidades típicas que aparecen en el ámbito de los SIG cuando se trabaja con redes de agua, redes viales, etc.

La extensión se estructura como se muestra en la siguiente figura.

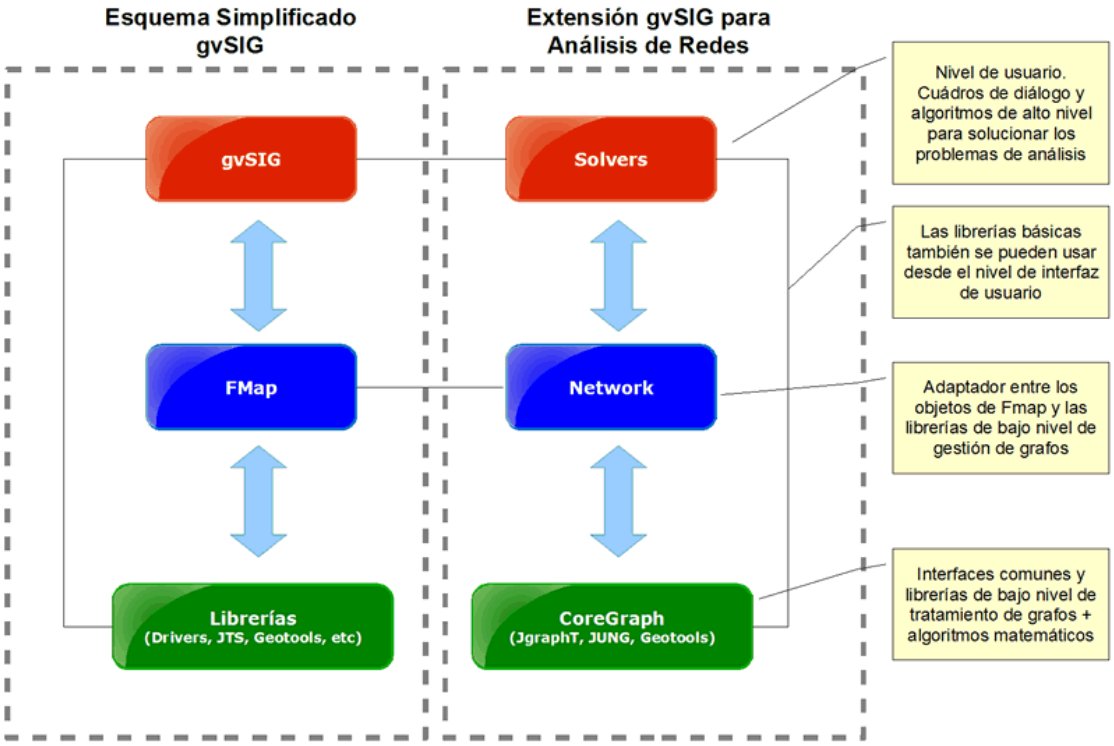


Figura 10. Esquema general de los componentes de la extensión de redes de gvSIG (Acevedo, 2011)

Dentro del subsistema *CoreGraph* se encuentran las estructuras e interfaces necesarias para trabajar con la definición matemática de la red. Se definen los nodos, los arcos, las aristas, el grafo y los algoritmos y eventos con los que trabajan los niveles superiores.

El subsistema *Network* es la interfaz entre la parte matemática y la librería *FMap*. Contiene el código fuente que crea la red a partir de una capa vectorial tipo línea, sitúa puntos sobre la red, especifica campos de costo primario y costo secundario, modifica costos de paso y giro.

En el subsistema *Solvers* están los algoritmos y geoprocesos necesarios para calcular los problemas de alto nivel: Ruta mínima, agente viajero, área de influencia, árbol de recubrimiento mínimo, matriz de distancias, conectividad de red, localización de la ubicación más cercana (Acevedo, 2011).

A nivel código fuente de la extensión, la terminología usada para los elementos que componen la red son:

- Arco (*IArc*). Tramo original entre 2 nodos. Los arcos que corresponden unívocamente a la entidad gráfica original. Un arco puede dar lugar a una o dos aristas.

- Arista (*IEdge*). En un grafo dirigido, es la línea que une 2 nodos y se recorre desde el nodo1 al nodo 2, y no en sentido contrario. Puede tener asociado un costo principal y un costo secundario (regularmente tiempo y distancia).
- Nodo (*INode*). Punto donde convergen 2 o más aristas. Puede tener asociados costes de giro.
- Grafo (*IGraph*). Es la colección de nodos y aristas. Hay funciones para añadir un nodo o arista, y recuperar el arco que originó una arista.

Otros términos que fueron clave para la solución del algoritmo Frank-Wolfe son:

- Ruta (*Route*). Es un arreglo que contiene los siguientes campos de cada arco que compone la ruta: ID, costo, longitud y nombre del arco.
- Parada (*GvFlag*). Es utilizada para multiples propósitos, en este caso define el punto de partida y de llegada de la ruta a calcular.
- Ruta mínima de Dijkstra (*ShortestPathSolverDijkstra*). Calcula la Ruta mínima entre dos puntos con el algoritmo de Dijkstra, y devuelve como resultado una Ruta (*Route*).

2.4. CREACIÓN DE EXTENSIONES PARA GVSIG VERSIÓN 1.X

Para desarrollar sobre gvSIG, es conveniente estar familiarizado con este software desde el punto de vista usuario. En la página web de gvSIG están disponibles los manuales de usuario para las diferentes versiones de gvSIG Desktop:

http://www.gvsig.org/web/projects/gvsig-desktop/docs/user

Navegando en la siguiente página web se pueden descargar manuales y los archivos utilizados en los ejemplos, a modo de poder reproducir paso a paso los ejercicios:

http://downloads.gvsig.org/download/documents/learning/

Al momento de desarrollar una extensión sobre una aplicación, se recomienda conocer dicha aplicación en profundidad. En este caso, la extensión Asignación por equilibrio, se desarrolló sobre la aplicación de Redes, y se profundizó el conocimiento de la aplicación mediante ejercicios con variaciones que permitieron evaluar el comportamiento de los algoritmos en cada caso.

Dentro de la documentación existente y de gran ayuda en el desarrollo de una extensión para gvSIG versión 1.X en Windows, están:

- Montar gvSIG 1.9 en Eclipse desde el repositorio SVN (Cristóbal)
- Construir gvSIG desde el repositorio SVN (Acevedo, 2011)
- Como desarrollar contra unos binarios de gvSIG 1.9 (Sevilla, 2010)

- Crear una extensión desde 0 en gvSIG (Piera)
- Adaptación y extensión de un SIG (Gavarró, 2011)
- gvSIG's extVRP Documentation (Pinheiro, 2012)

Crear una nueva extensión de gvSIG en Eclipse se puede hacer de dos formas (Sevilla, 2010):

- Crear un workspace con el código fuente de gvSIG (*libfmap*, *_fwandami*, etc...), además del proyecto para la extensión.
- Crear un workspace sólo con el proyecto de la extensión y utilizar los binarios de gvSIG para ejecutarla.

La extensión *extModelTransport* se desarrolló en un workspace con el código fuente mínimo necesario para trabajar con redes, lo que facilita el análisis de las clases cuyas funcionalidades interesan para la extensión de Asignación por Equilibrio.

El procedimiento para lograr un workspace como el mostrado en la siguiente figura (a excepción del proyecto *extModelTransport*), se describe en el documento "gvSIG's extVRP Documentation".

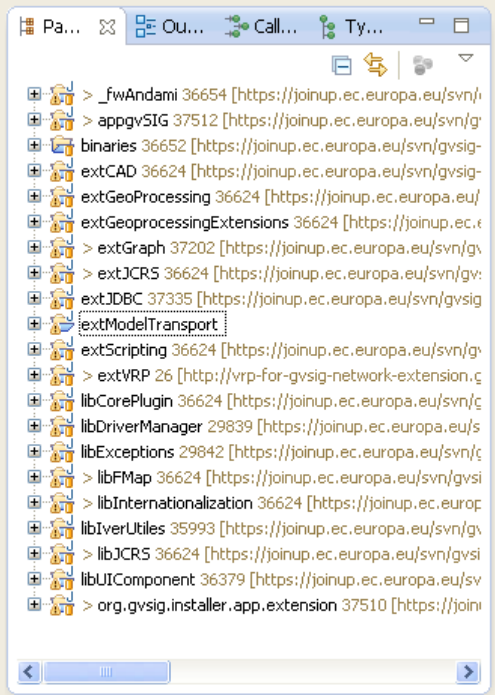


Figura 11. Proyectos básicos para trabajar con redes

Descargar el código fuente de gvSIG en Eclipse IDE permite analizarlo de manera fluida, haciendo uso de los comandos que se muestran dando clic derecho sobre la parte del código que interese: un método, una variable, una clase, etc. En la siguiente figura se muestran algunos comandos que sirven para navegar en el código fuente.

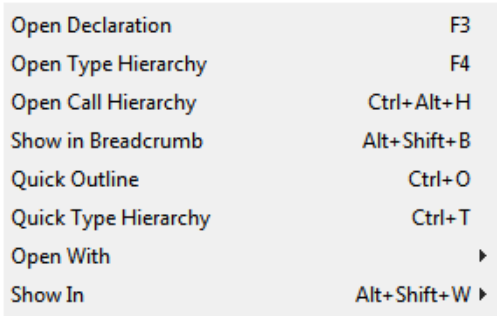


Figura 12. Comandos de Eclipse IDE para navegar en el código fuente

Para un mejor análisis y comprensión del código fuente de gvSIG, se recomienda la consulta de las siguientes referencias:

- Manual para desarrolladores gvSIG v1.1 (Peñarrubia, 2010)
- Guía de extensiones para gvSIG 1.9 (Acevedo, 2011)
- Guía de referencia para gvSIG 1.1 (Acevedo, 2010)
- API de Andami (andami API)
- API de gvSIG (gvSIG API)

2.5. COMPLEMENTO ASIGNACIÓN POR EQUILIBRIO

Un complemento o plug-in, es un programa de cómputo que aporta una nueva funcionalidad a un programa principal.

En gvSIG 1.X, un complemento contendrá los siguientes elementos:

- archivo *config.xml* que describe el complemento,
- archivos de traducciones *.properties*,
- código fuente del complemento (clases Java),
- achivos varios: *build.xml*, imágenes, *.jar*, *readme.txt*, etc.

El código fuente del complemento es un conjunto de clases Java que implementan la funcionalidad aportada por el complemento. Probablemente, el código irá acompañado de archivos de configuración, imágenes, datos geográficos o cualquier otro dato que sea necesario para la ejecución de éste. En la figura se muestra la estructura del módulo de Asignación por equilibrio (*extModelTransport*), que contiene: clases separadas en paquetes de acuerdo a su funcionalidad, el archivo de configuración *config.xml*, los archivos *text.properties* y *text_en.properties* con la traducción de los textos de la extensión en español y en inglés respectivamente, una carpeta donde se ubica la imagen que representa la funcionalidad de la extensión, el archivo *build.xml*, entre otros.

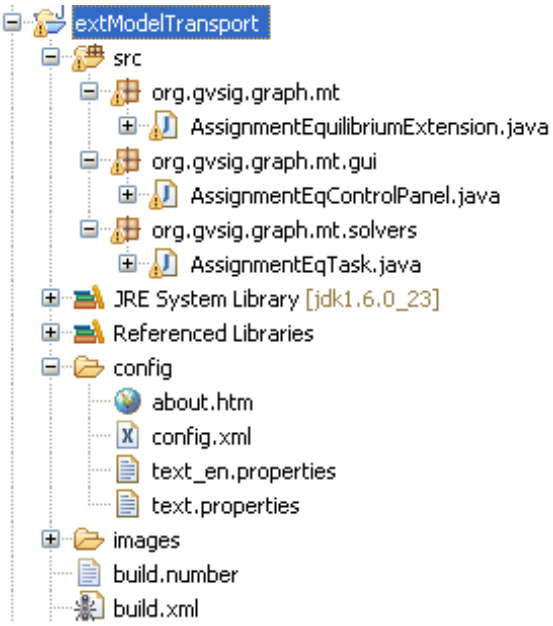


Figura 13. Estructura del módulo Modelos de transporte/Asignación por equilibrio

El título del proyecto *extModelTransport* se debe a que la asignación de viajes a la red forma parte de los modelos de transporte (mencionados en el Capítulo 1), y se pretende que futuros desarrollos con la misma temática vayan complementando el proyecto.

El nombre de los paquetes del proyecto *extModelTransport* está compuesto por un nombre de grupo *groupId org.gvsig*, de un identificador *artifactId graph*, de un identificador raíz del proyecto *mt*; así como de otros identificadores *gui* y *solvers*, con el propósito de separar la Interfaz gráfica de usuario de la parte lógica de la extensión.

La clase principal de extensión *AsignmentEquilibriumExtension.java* se encuentra en el paquete raíz del proyecto *org.gvsig.graph.mt*. La clase que suministra la Interfaz gráfica de usuario *AssignmentEqControlPanel.java* se ubica en el paquete correspondiente *org.gvsig.graph.mt.gui*. Y la parte lógica de la extensión, la clase *AssignmentEqTask.java* se ubica en el paquete *org.gvsig.graph.mt.solvers*.

2.5.1. Archivo de configuración *config.xml*

El archivo de configuración del complemento debe llamarse *config.xml*. En él se proporciona a gvSIG la información necesaria sobre el complemento para que pueda cargarlo e integrarlo en la interfaz gráfica. Entre otras cosas, especifica el directorio que contiene el código del complemento, las dependencias que tiene con otros complementos y los elementos de interfaz de usuario (menús, barras de herramientas, etc.) que añade a la interfaz de gvSIG (Gavarró, 2011). Debe ser escrito en XML (eXtensible Markup Language), un lenguaje que tiene una estructura marcada por etiquetas que deben abrirse

y posteriormente cerrarse en orden inverso al orden de apertura. El archivo *config.xml* obligatoriamente debe acompañar al código del complemento.

Cuando el usuario da clic en una herramienta o en una entrada de menú, *Andami* ejecuta el método *execute* de la extensión correspondiente. De esta forma, la extensión realizará las acciones necesarias para ejecutar la operación solicitada, posiblemente apoyándose en otras clases o librerías del complemento (Acevedo, 2010).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<plugin-config>
      <libraries library-dir="./lib">
      </libraries>
      <depends plugin-name="com.iver.cit.gvsig"/>
      <depends plugin-name="com.iver.cit.gvsig.geoprocessextensions"/>
      <depends plugin-name="com.iver.cit.gvsig.cad"/>
      <depends plugin-name="org.gvsig.graph"/>
            <resourceBundle name="text"/>
      <extensions>

<!-- ModelTransport -->
            <extension
            class-name="org.gvsig.graph.mt.AssignmentEquilibriumExtension"
            description=
            "Route_choice_under_congested_conditions:_deterministic_case"
            active="true">
                  <menu
                  text="Model_transport/Assignment_equilibrium"
                  icon="images/assignment.png"
                  tooltip="Car_route_choice"
                  action-command="MTEQUILIBRIUM"/>
            </extension>
<!-- /ModelTransport -->


      </extensions>
</plugin-config>
```

A continuación se describe el significado de las etiquetas que componen el archivo *config.xml* del complemento *extModelTransport*:

- *prólogo*: es la primera línea, se indica la versión del lenguaje XML utilizado y la codificación de caracteres con que se guardará el archivo.
- *plugin-config*: es la etiqueta que marca el inicio y fin del archivo, englobando todas las opciones de configuración.
- *libraries*: especifica el directorio del que se leerán las librerías del complemento.
- *depends*: indica los complementos de los que depende, a nivel de librerías.
- *extensions*: etiqueta donde se enlistan las extensiones del complemento.
- *extension*: declara una extensión de gvSIG que incluye el nombre de la clase Java que implementa esta extensión, y una lista de elementos de interfaz de usuario

que se añadirán a la aplicación y estarán asociados con ésta. Acepta los siguientes atributos:

  - *class-name*: nombre de la clase que implementa esta extensión.
  - *description*: descripción de la funcionalidad aportada por la extensión.
  - *active*: especifica si gvSIG carga o no la extensión.

- *menu*: crea una entrada de menú en la barra de estado de gvSIG. Al dar clic en el menú se lanzará el método *execute()* de la extensión asociada, pasándole como parámetro el *action-command* especificado en esta etiqueta. Con los siguientes atributos:
  - *text*: establece el nombre y jerarquía del menú mediante la separación por barras (/). Cada porción de texto empleado aquí constituye una clave de traducción, puesto que el texto real mostrado en los menús y submenús se traduce al idioma seleccionado por el usuario.
  - *icon*: ruta de la imagen que se mostrará dentro del botón.
  - *tooltip*: texto que se mostrará al mantener quieto el ratón sobre el botón durante unos segundos.
  - *action-comand*: especifica el identificador de acción. Cada vez que se active el menú, se llamará a la clase asociada pasándole el valor de este atributo, lo que permite asociar varios menús a una sola clase.
- *<!-- ModelTransport -->*: comentario que indica las extensiones pertenecientes al menú Modelos de transporte.

Es en este archivo *config.xml* donde se declara la opción de menú *Modelos de transporte*, submenú *Asignación por equilibrio*, con el identificador "*MTEQUILIBRIUM*". El resultado es el siguiente:

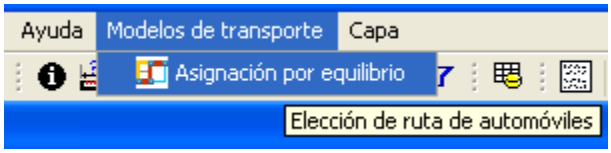


Figura 14. Opción de menú Modelos de transporte/Asignación por Equilibrio

2.5.2. Archivos de traducción *text_xx.properties*

Los archivos con extensión *.properties*, que se encuentran dentro de la carpeta *config*, contienen la traducción de los textos, que están en modo de clave en el código fuente, a un idioma en particular. Por ejemplo, en el recuadro las primeras líneas del archivo *text.properties*, el texto clave está a la izquierda del signo igual y a la derecha está el texto en español.

```
#text.properties
Assignment_equilibrium=Asignación por equilibrio
Car_route_choice=Elección de ruta de automóviles
Route_choice_under_congested_conditions-_deterministic_case=Elección de ruta
en condiciones de congestionamiento: caso determinístico
Deterministic_car_route_choice=Elección de ruta de automóviles: caso
determinístico
Origins=Origenes
```

En este otro ejemplo, con las primeras líneas del archivo *text_en.properties*, el texto clave está a la izquierda del signo igual y a la derecha está el texto en inglés.

```
#text_en.properties
Assignment_equilibrium=Assignment equilibrium
Car_route_choice=Car route choice
Route_choice_under_congested_conditions-_deterministic_case=Route choice
under congested conditions: deterministic case
Deterministic_car_route_choice=Deterministic car route choice
Origins=Origins
```

Con el método *getText* de la clase *PluginServices* de *Andami*, se obtiene el texto al idioma actual seleccionado en las preferencias de gvSIG, con el propósito de que los textos de Interfaces gráficas y mensajes mostrados al usuario, puedan ser leídos en diferentes idiomas. De esta manera el menú y la Interfaz gráfica de usuario de la extensión pueden leerse también en inglés.

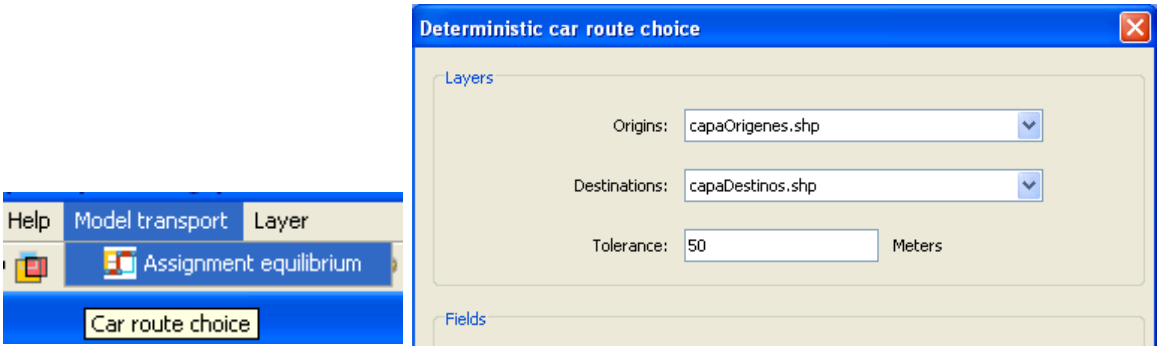


Figura 15. Menú, submenú e Interfaz gráfica de usuario en inglés, de la extensión Asignación por equilibrio

2.5.3. Extensión *Asignación por equilibrio*

Con el fin de separar las funcionalidades y facilitar su mantenimiento, acorde a la modularidad requerida en la documentación de desarrollo gvSIG, el código fuente de la extensión *Asignación por Equilibrio* está conformado por tres clases:

- *AssignmentEquilibriumExtension*: es la clase que enlaza a gvSIG con las funcionalidades del complemento.
- *AssignmentEqControlPanel*: es la clase encargada de la Interfaz gráfica de usuario, por medio de la cual obtendremos los datos de entrada que necesita el algoritmo de Frank-Wolfe.
- *AssignmentEqTask*: contiene la parte lógica que resuelve el problema de asignación de viajes a la red con el algoritmo Frank-Wolfe.

En la siguiente figura se muestra un diagrama UML (Unified Modeling Language) de las clases que componen la extensión *Asignación por equilibrio* y su principal relación con las clases de gvSIG.

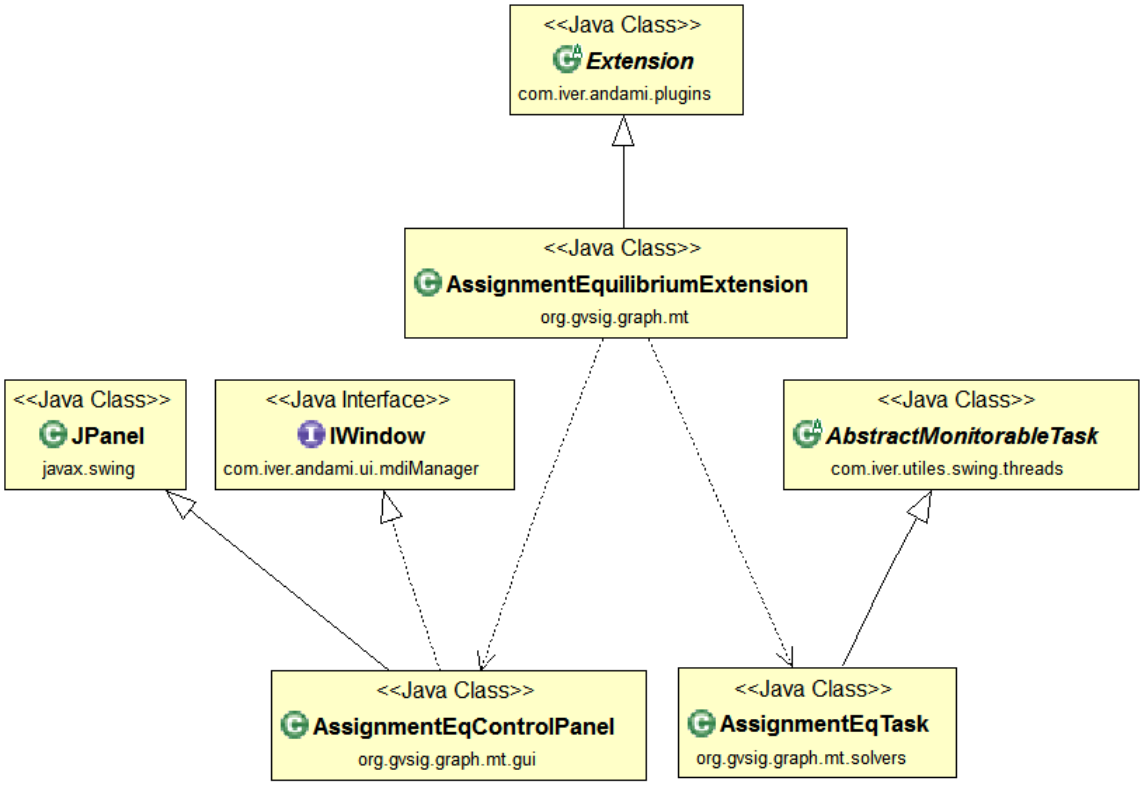


Figura 16. Diagrama de clases UML de la extensión Asignación por equilibrio

Se observa en el diagrama que la clase principal *AssignmentEquilibriumExtension* hereda de la clase abstracta *Extension* y para su funcionamiento depende de las clases *AssignmentEqControlPanel* y *AssignmentEqTask*. A su vez, *AssignmentEqControlPanel*

hereda de *JPanel* e implementa la clase *IWindow*; y la clase *AssignmentEqTask* hereda de *AbstractMonitorableTask*.

En la tabla se describe la simbología de relación entre las clases, usada en el diagrama anterior.

Tabla 2. Simbología de relación de cases en el diagrama UML

Símbolo	Representa	Función	Se lee
A ◁——— *B*	Generalización	Indica que el modelo del elemento *B* es una especialización del modelo del elemento *A*.	*B* hereda de *A*
A ◁- - - - - *B*	Implementación	Indica que el modelo del elemento *A* proporciona especificaciones al modelo del elemento *B* que lo implementa.	*B* implementa *A*
A <- - - - - *B*	Dependencia	Indica que el modelo del elemento *B* requiere del modelo del elemento *A* para su completo funcionamiento.	*B* depende de *A*

Las clases de la extensión se muestran en su forma completa con la simbología UML, en la siguiente figura; donde en el primer recuadro se lee el nombre de la clase en negritas, en el del medio se muestran los atributos y en el inferior los métodos de la clase.

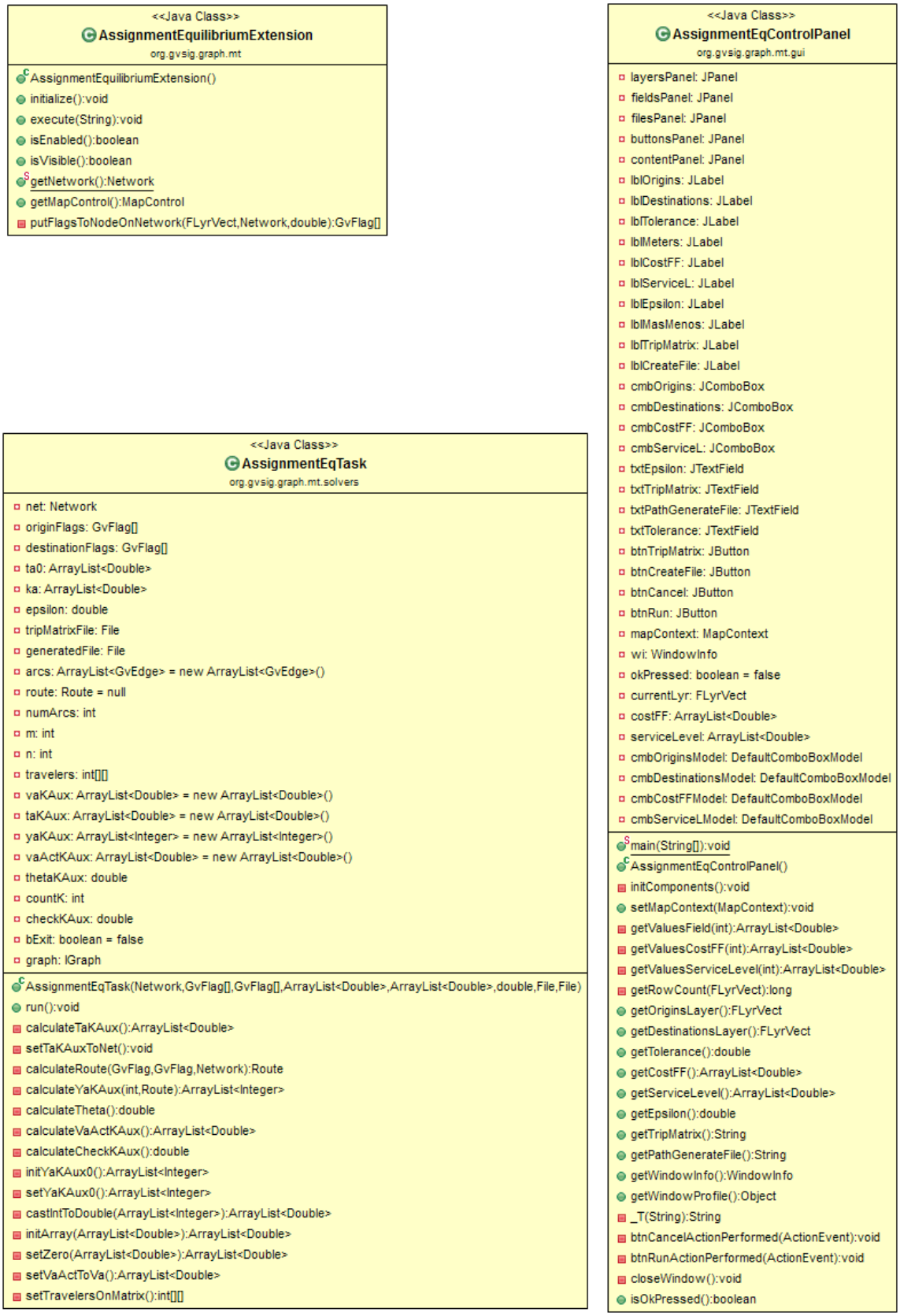


Figura 17. Representación UML de las clases de la extensión Asignación por equilibrio

2.5.3.1. Clase *AssignamentEquilibriumExtension*

AssignamentEquilibriumExtension es la clase encargada de recibir la notificación de las acciones realizadas por el usuario (clic en el submenú *Asignación por equilibrio*) y ejecutará el código apropiado en respuesta. Para ello, esta clase debe extender la clase abstracta *Extension* e implementar los métodos abstractos: *initialize()*, *execute(String)*, *isEnabled()* e *isVisible()*. Toda clase que hereda o extiende la clase *Extension*, es una extensión, y todas las extensiones están obligadas a implementar estos cuatro métodos.

Estos métodos han sido implementados en la clase *AssignamentEquilibriumExtension* de la siguiente manera:

- *void initialize()*: se utiliza para inicializar la extensión. En este método se llaman a funciones que se encarguen de reservar recursos, leer configuraciones, etc. En este caso, este método se utiliza para obtener la imagen del ícono en el menú de la extensión.

```
public void initialize() {
	PluginServices.getIconTheme().
		registerDefault("assignment",this.getClass().
		getClassLoader().getResource("images/assignment.png")
		);
}
```

- *void execute(String actionCommand)*: este método es llamado cada vez que el usuario accede mediante un clic a alguno de los elementos de interfaz de usuario asociado a la extensión. Es el encargado de ejecutar el código que proporciona la funcionalidad de la extensión. Recibe como argumento el valor del atributo *action-comand*, definido en el archivo *config.xml*, del elemento al que se ha accedido. Como se observa a continuación, es en este método donde se crean objetos de las clases *AssignmentEqControlPanel* y *AssignmentEqTask*.

```
public void execute(String actionCommand) {
	...
	if (actionCommand.equals("MTEQUILIBRIUM")) {
		while (it.hasNext())
		{...
			if ( net != null)
			{
				AssignmentEqControlPanel ctrlDlg =
					new AssignmentEqControlPanel();
				try {...
					if (ctrlDlg.isOkPressed()) {
						AssignmentEqTask task =
						new AssignmentEqTask(...);
					...
					}
				}catch (BaseException e) {
					e.printStackTrace();
				}
			}
		}
	}

}
```

- *boolean isEnabled()*: controla si la extensión se habilita o no. Cuando la extensión no está habilitada, los controles como botones y menús, se muestran en tonos grises y no son accesibles. Se muestra en el siguiente código, que la extensión sólo se habilitará si la capa activa tiene una red en sus atributos.

```
public boolean isEnabled() {
	Network net = getNetwork();
	if ( net != null)
	{
		return true;
	}
	else
		return false;
}
```

- *boolean isVisible()*: controla si son visibles, o no, los elementos de interfaz de usuario asociados a la extensión. En el siguiente código se indica que los elementos de la extensión serán visibles si la ventana activa es una ventana Vista (*View*).

```
	public boolean isVisible() {
		IWindow f = PluginServices.getMDIManager().getActiveWindow();
		if (f == null) {
		    return false;
		}
		if (f instanceof View) {
			return true;
		}
		return false;
	}
```

2.5.3.2. Clase *AssignmentEqControlPanel*

Como su nombre lo indica, la clase *AssignmentEqControlPanel*, es la clase que configura el diseño y funcionalidad de la Interfaz Gráfica de Usuario (GUI, por sus siglas en inglés), cuyo objetivo principal es facilitar el ingreso de datos requeridos por el algoritmo que soluciona el problema de asignación de viajes a la red.

La Interfaz gráfica de usuario de la extensión *Asignación por equilibrio*, se diseñó en base al tipo de datos de entrada y de salida. Se buscó que los datos de entrada fuesen lo más genérico posible, que permitieran el ingreso de archivos de gran tamaño, de datos que no necesiten de un proceso adicional para ingresarlos al programa, que sean datos crudos, con formatos comunes (*shp*, *txt*) y de preferencia que los datos se ocupen para más de un proceso. Así también, se procuró que los datos de salida fuesen fáciles de comprender para la toma de decisiones, que contengan información valiosa (información + valor agregado), que puedan servir para alimentar a un proceso posterior y que su formato sea común.

La Interfaz gráfica de usuario se divide en cuatro áreas: capas, campos, archivos y botones, como se observa en la figura.

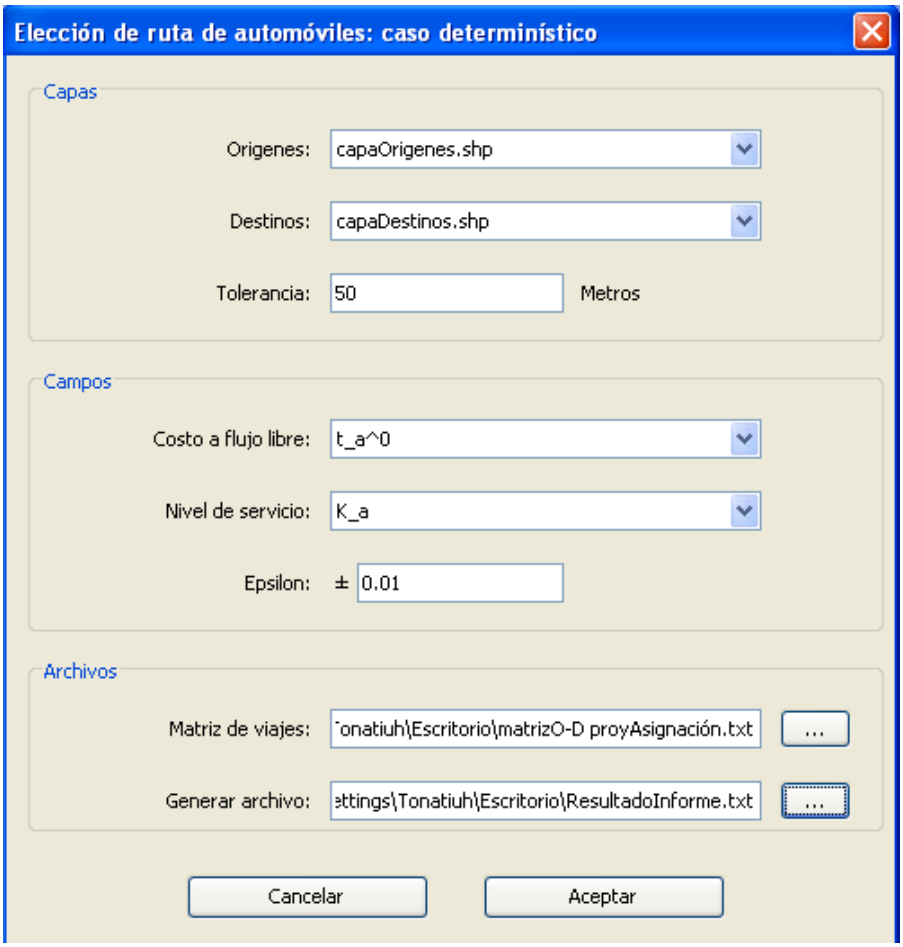


Figura 18. Interfaz Gráfica de Usuario (GUI) de la extensión Asignación por equilibrio

En el primer recuadro se ingresan la capa correspondiente a los orígenes, la capa correspondiente a los destinos y una distancia en metros como tolerancia para agregar los puntos de origen y destino a los nodos de la red, que por defecto es de 50 metros.

Para el ingreso de las capas, se hace uso de una lista desplegable (*JComboBox*), que en su contenido muestra todas aquellas capas *shp* tipo punto o multipunto que estén en la Tabla de Contenidos (TOC) de gvSIG, haciendo un filtro que facilita y asegura que la selección de los datos de entrada sean correctos. El código encargado del filtro es el siguiente.

```
if (lyr instanceof FLyrVect) {
      FLyrVect lyrVect = (FLyrVect) lyr;
      if ((lyrVect.getShapeType() == FShape.POINT)
            || (lyrVect.getShapeType() == FShape.MULTIPOINT)) {
                        FLyrVect lyrVectPoint = (FLyrVect) lyrVect;
                        arrayLayers.add(lyrVectPoint);
                  }
```

En el segundo recuadro se ingresan los datos relacionados con los arcos de la red, que son el Costo a flujo libre t_a^0, el Nivel de servicio K_a, y un valor numérico llamado Épsilon ε, que será el nivel de convergencia deseado para que el algoritmo llegue a la solución, que por defecto es 0.01.

Hay que considerar que la capa activa deberá ser vectorial, de formato *shp* tipo línea, y contar con una red en sus atributos; y que la Tabla de atributos de la capa activa deberá tener ingresados previamente los campos con los valores correspondientes al Costo a flujo libre t_a^0 y el Nivel de servicio K_a para cada arco. Para facilitar su selección en la GUI, se hace un filtro, que se muestra en el siguiente código, y en la lista desplegable sólo se muestran los campos cuyo valor es numérico.

```
if ((lyrVect.getShapeType() == FShape.LINE) && (lyrVect.isActive())) {
	this.currentLyr = (FLyrVect) lyrVect;
	try {
		String[] aux = currentLyr.getRecordset().getFieldNames();
		for (int i = 0; i < aux.length; i++) {
			switch (currentLyr.getRecordset().getFieldType(i)) {
			case Types.BIGINT:
			case Types.DECIMAL:
			case Types.DOUBLE:
			case Types.FLOAT:
			case Types.INTEGER:
			case Types.NUMERIC:
			case Types.REAL:
			case Types.SMALLINT:
			case Types.TINYINT:
				arrayFields.add(aux[i]);
				break;
			}
		}
	} catch (Exception e) {
		e.printStackTrace();
	}
}
```

En el recuadro correspondiente a los archivos, se ingresa la ruta donde se ubica el archivo de texto *.txt* que contiene la matriz de viajes; o bien, dando clic en el botón con puntos suspensivos (...), se abre una ventana para seleccionar el archivo de manera cómoda y precisa. Así mismo, se ingresa la ruta donde se desea que se guarde el archivo generado, que contendrá un informe sobre las iteraciones y la evolución de los valores hacia la convergencia del equilibrio.

La cuarta área, en la parte baja de la GUI, se encuentran dos botones, uno es para dar inicio al proceso y otro es para cancelarlo.

2.5.3.3. Clase *AssignmentEqTask*

La clase *AssignmentEqTask* es la encargada de procesar los datos de entrada que alimentan el algoritmo de Frank-Wolfe, y resolver el problema determinístico de asignación de viajes a la red.

Los pasos del algoritmo que dan solución al problema de minimización de desutilidad

$$\min_{(T_{ijr})} U'_R = \sum_a \int_0^{v_a} g_a(v)dv$$

De acuerdo con la bibliografía Oppenheim, 1995, pueden ser resumidos como se muestra a continuación.

Linealización del problema determinístico de asignación de viajes para automóvil

Cada paso del algoritmo estará acompañado por el código fuente encargado de realizar dicha tarea. Para ver el código en contexto, puede consultar el Apéndice A.

Inicialización

1. Fijar el contador de iteración $k = 0$.

```
countK = 0;
```

2. Fijar la Demanda de los arcos en cero: $v_a^0 = 0$.

   No es necesario, puesto que t_a^0 se da como dato.

3. Determinar los correspondientes Costos de arco t_a^0 de la función de desempeño del arco.

$$t_a = t_a^0\left[1 + 0.15\left(\frac{0}{K_a}\right)^4\right] = t_a^0; \qquad \forall\ a$$

   Se da como dato.

4. Sumar los costos de los arcos que componen la ruta para obtener el Costo de ruta.

El propósito del paso 4 es identificar la Ruta mínima.

5. Asignar cada uno de los viajeros T_{ij} dados, a la correspondiente Ruta mínima entre i y j, utilizando el algoritmo de Ruta mínima con los costos iniciales del arco t_a^0.

En vez de asignar los viajeros de i a j a la Ruta mínima r_{ij}, se asignan a los arcos que forman a dicha ruta, pasando del paso 4 al 6.

6. Sumar los viajeros de las rutas que utilizan el arco a, dando como resultado la Demanda de arco v_a^1.

Sólo en este paso, la Demanda de arco se calcula de la misma manera que la Demanda de arco auxiliar y_a.

```
for (int i = 0; i < originFlags.length; i++) {
      for (int j = 0; j < destinationFlags.length; j++) {
            if (travelers[i][j] == 0)
                  continue;
            route = null;
//Paso 4. Cálculo de Ruta Mínima
            route = calculateRoute(originFlags[i],
                  destinationFlags[j], net);
            calculateYaKAux(travelers[i][j], route);
      }
}
//Paso 6. Cálculo de la Demanda de Arco v_a^k+1
vaKAux = castIntToDouble(yaKAux);
```

Pasos a iterar

Incrementar k en 1, $k = k + 1$ ó $k++$.

```
countK++;

while (!bExit) { del paso 7 al 13}
```

7. Actualice los Costos de arco en función de la congestión de éste.

$$t_a^k = t_a^0\left[1 + 0.15\left(\frac{v_a^k}{k_a}\right)^4\right]; \qquad \forall\ a$$

En este paso se calculan los nuevos Costos de arco con el método *calculateTaKAux()*.

```
private ArrayList<Double> calculateTaKAux() {
	for (int i = 0; i < numArcs; i++) {
		taKAux.set(i, (ta0.get(i)*(1+0.15*
				(Math.pow((vaKAux.get(i)/ka.get(i)), 4)))));
	}
	return taKAux;
}
```

8. Sumar los Costos de los arcos que componen la ruta para obtener el Costo de ruta.

$$t_r^k = \sum_{a \in r} t_a$$

En vez de ello, el código asigna los nuevos Costos de arco a la red, con el método *setTaKAuxToNet()*.

```
private void setTaKAuxToNet(){
	for (int i = 0; i < numArcs; i++) {
		arcs.get(i).setDistance(taKAux.get(i));
		arcs.get(i).setWeight(taKAux.get(i));
	}
}
```

Como el objetivo en este paso es encontrar la Ruta mínima, el método *calculateRoute()* necesita que la red que se le pasa como argumento tenga los nuevos Costos de Arco.

```
private Route calculateRoute(GvFlag origen, GvFlag destino, Network net)
throws GraphException {
	ShortestPathSolverDijkstra solver = new ShortestPathSolverDijkstra();
	Route route = null;
	try {
		net.removeFlags();
		solver.setNetwork(net);
		net.addFlag(origen);
		net.addFlag(destino);
		String fieldStreetName = (String) net.getLayer().
			getProperty("network_fieldStreetName");
		solver.setFielStreetName(fieldStreetName);
		route = solver.calculateRoute();
	} catch (GraphException ex) {
		ex.printStackTrace();
	}
	if (route.getFeatureList().size() == 0) {
		JOptionPane.showMessageDialog((JComponent) PluginServices.
			getMDIManager().getActiveWindow(),
			PluginServices.getText(this,
			"shortest_path_not_found"));
	}
	return route;
}
```

Se observa a continuación, cómo se calcula la Ruta mínima para cada origen-destino y en seguida se asignan los viajeros T_{ij} a los arcos que forman parte de la Ruta mínima.

```
for (int i = 0; i < originFlags.length; i++) {
	for (int j = 0; j < destinationFlags.length; j++) {
		if (travelers[i][j] == 0)
			continue;
		route = null;
//Cáculo de Ruta Mínima para cada origen-destino
		route = calculateRoute(originFlags[i],
			destinationFlags[j], net);
//Asignación de los viajeros Tij a los arcos de la Ruta Mínima calculada
		calculateYaKAux(travelers[i][j], route);
	}
}
```

9. Asignar cada uno de los viajeros T_{ij} dados al correspondiente costo mínimo generalizado de ruta entre i y j, utilizando el algoritmo de Ruta mínima con los costos actuales del arco t_a^k para obtener la Demanda de ruta auxiliar Z_r^k.

En vez de asignar los viajeros de i a j a la Ruta mínima r_{ij}, se asignan a los arcos que forman a dicha ruta, pasando del paso 8 al 10.

10. Se suman los viajeros T_{ij} de las rutas que utilizan el arco a, obteniendo como resultado la Demanda de arco auxiliar y_a^k.

$$y_a^k = \sum_{r \ni a} T_{ij}$$

El cálculo de la Demanda de arco auxiliar y_a^k, se realiza con el método *calculateYaKAux()*.

```
private ArrayList<Integer> calculateYaKAux (int travelers_i_j, Route route){
	ArrayList featureList = route.getFeatureList();
	for (int i = 0; i < featureList.size(); i++) {
		int id = 0;
		IFeature feature = (IFeature) featureList.get(i);
		id = ((IntValue)feature.
			getAttribute(Route.ID_ARC_INDEX)).getValue();
		yaKAux.set(id, (yaKAux.get(id)+travelers_i_j));
	}
	return yaKAux;
}
```

11. Resolver el valor de θ que minimice la desutilidad de viaje del viajero representativo

$$\min_{(0 \le \theta \le 1)} U'_R = \sum_a \int_0^{v^k + \theta(y^k - v^k)} g_a(v)dv$$

Evaluando directamente la siguiente ecuación con el método de bisección.

$$\frac{dU'_R}{d\theta} = \sum_a (y_a^k - v_a^k) t_a^k \left[1 + 0.15\left(\frac{v_a^k + \theta^k(y_a^k - v_a^k)}{K_a}\right)^4\right]$$

El método para calcular θ se llama *calculateTheta()*.

```
private double calculateTheta(){
	double a = 0;
	double b = 1;
	double eval = 0;
	boolean bEnd = false;

	while (!bEnd) {
		thetaKAux = (a+b)/2;
		for (int i = 0; i < numArcs; i++) {
			eval += ((yaKAux.get(i)-vaKAux.get(i))*(ta0.get(i))*
					(1+0.15*(Math.pow(((vaKAux.get(i)+thetaKAux*
					(yaKAux.get(i)-vaKAux.get(i)))/
					ka.get(i)), 4))));
		}
		if (0<=eval) {
			b = thetaKAux;
		} else {
			a = thetaKAux;
		}
		if (Math.abs(eval)<1) {
			bEnd = true;
			break;
		}
		eval = 0;
	}
	return thetaKAux;
}
```

12. Actualizar el valor de la Demanda de arco v_a^{k+1} con la siguiente ecuación.

$$v_a^{k+1} = v_a^k + \theta\left(y_a^k - v_a^k\right); \qquad \forall\ a$$

El método *calculateVaActKAux()* calcula la Demanda de arco actualizada.

```
private ArrayList<Double> calculateVaActKAux() {
	for (int i = 0; i < numArcs; i++) {
		vaActKAux.set(i,(vaKAux.get(i)+thetaKAux*
			(yaKAux.get(i)-vaKAux.get(i))));
	}
	return vaActKAux;
}
```

13. Con la siguiente ecuación, verificar la convergencia.

$$\sum_a \left(y_a^k - v_a^{k+1}\right) t_a^0 \left[1 + 0.15\left(\frac{v_a^{k+1}}{K_a}\right)^4\right] \leq \varepsilon$$

Donde épsilon ε es la tolerancia acordada. Si ocurre la condición, detener, sino, iterar a partir del paso 7.

El método *calculateCheckKAux()* es el encargado de evaluar la ecuación.

```
private double calculateCheckKAux() {
	checkKAux = 0;
	for (int i = 0; i < numArcs; i++) {
		checkKAux +=((yaKAux.get(i)-vaActKAux.get(i))*
			(ta0.get(i))*(1+0.15*(Math.pow(
			(vaActKAux.get(i)/ka.get(i)), 4))));
		}
		return checkKAux;
	}
```

El código que evalúa la condición con épsilon y determina si se sigue iterando, o no, es el siguiente.

```
if (Math.abs(checkKAux)<=epsilon) {
	bExit = true;
	break;
}
```

Una vez que se cumpla que el criterio de convergencia sea menor a épsilon, el problema determinístico de asignación de viajes a la red bajo condiciones de congestionamiento, habrá sido solucionado con los valores de la Demanda de arco actualizada v_a^{k+1} de la última iteración, que en otras palabras, es el volumen de viajes en cada arco, que causa el equilibrio de la red.

2.6. RESUMEN

La documentación de desarrollo para gvSIG, que se incluye en la bibliografía, es esencial para el aprendizaje de quien desea desarrollar, modificar o personalizar este Sistema de información geográfica.

El entorno de programación Eclipse facilita el análisis de código, la programación misma, el diseño de la Interfaz gráfica de usuario, etc.; por ello se recomienda el aprendizaje de su uso, ya que gvSIG está programado en su totalidad con Eclipse IDE.

En el diagrama de flujo siguiente, se muestra el proceso general para el desarrollo de una extensión para gvSIG.

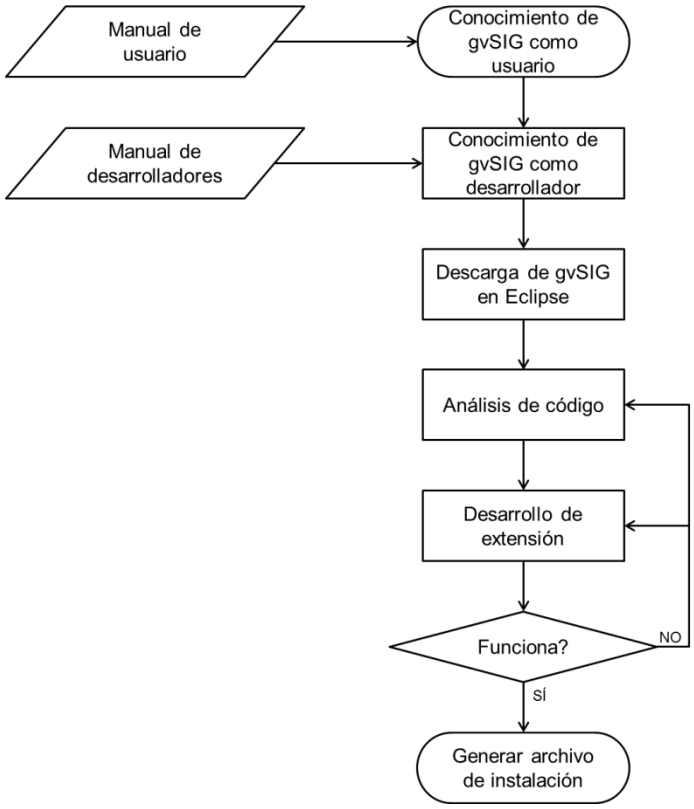


Figura 19. Diagrama de flujo para el desarrollo de extensiones para gvSIG

gvSIG al ser software libre promueve el crecimiento del mismo y el aprendizaje para quien lo hace crecer. Esta es la razón que ha hecho posible el desarrollado del módulo de software que se expone en este trabajo.

El código del módulo *Asignación por equilibrio* se puede consultar y descargar en las páginas:

http://extmodeltransport.blogspot.mx/

https://code.google.com/p/ext-model-transport/

CAPÍTULO 3. RESULTADO

3.1. INTRODUCCIÓN

En este capítulo se muestra cómo ingresar los datos en la Interfaz gráfica de usuario, con el ejemplo ilustrativo 5.2.4: Aplicación del algoritmo de linealización, del libro Urban Travel Demand Modeling (Oppenheim, 1995). Se compara la solución de la extensión *Asignación por equilibrio* con la del libro y se representa gráficamente la distribución de viajes en la red mediante simbología de grosor de línea.

Se indica la mejor manera de crear la topología de la red para el uso de la extensión, las características de las capas de puntos de los nodos orígenes y destinos, la forma en que se ingresan los valores de Costo a flujo libre y el Nivel de servicio a la Tabla de atributos de la capa activa, las especificaciones de la Matriz de viajes en el archivo de texto plano, entre otras.

3.2. PLANTEAMIENTO DEL PROBLEMA EJEMPLO

Se ilustrará la aplicación del algoritmo Frank-Wolfe con la red hipotética mostrada en la figura siguiente.

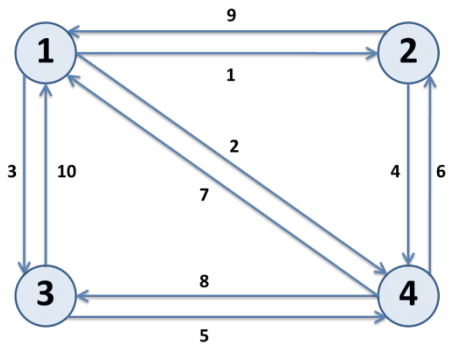


Figura 20. Red de ejemplo

La importancia de este ejercicio es que los tiempos de viaje de los arcos, en vez de tener valores fijos, tienen valores variables en función de la demanda en cada arco. Se asume que no hay arcos cargados inicialmente. El número de viajes T_{12} y T_{14} del nodo 1 al nodo 2 y del nodo 1 al nodo 4 respectivamente, son los siguientes:

$$T_{12} = 250$$

$$T_{14} = 150$$

La tabla de características de los arcos se presenta a continuación, donde t_a^0 es el Tiempo de flujo en el arco a con Demanda 0, ó Costo a flujo libre, y K_a es el Nivel de servicio.

Tabla 3. Características de los arcos

Arco	t_a^0	K_a
1	5	55
2	15	50
3	6	60
4	8	50
5	7	55
6	8	60
7	15	55
8	7	50
9	5	50
10	6	55

3.3. DATOS DE ENTRADA

Para hacer uso del menú *Modelos de transporte*/*Asignación por equilibrio* en gvSIG, es necesario tener los siguientes datos: la capa shape tipo línea que represente gráficamente a la red, una red generada de ésta, las capas shape tipo punto que indiquen geográficamente los orígenes y destinos de los viajes, el Costo a flujo libre y Nivel de servicio de cada arco, y la Matriz de viajes; que se generan e ingresan como se describe a continuación en los siguientes apartados.

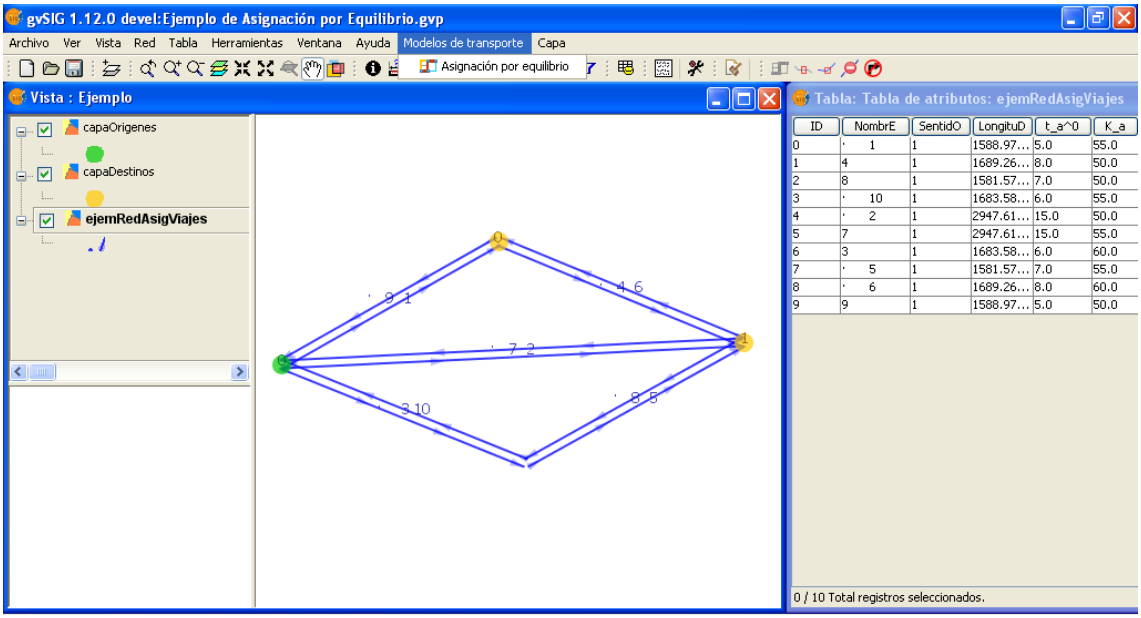


Figura 21. Datos de entrada para el módulo Asignación por equilibrio en gvSIG

Para más detalle sobre el manejo de datos en gvSIG, puede consultar “Curso de gvSIG 1.9” (CIT. Generalitat Valenciana, 2009), o “gvSIG-Desktop 1.12. Manual de usuario” (Madrid, 2012).

3.3.1. Red

Inicialmente se requiere de una capa shape tipo línea con las entidades gráficas de la red, es decir, líneas que representan los arcos que generarán la red. En la siguiente figura se observa que hay un arco para cada sentido. La simbología de la capa se configuró de tal manera que desplaza las líneas de su eje para mostrar de manera clara que hay dos líneas que unen cada nodo, el sentido en que se dibujó la línea y el nombre de cada arco.

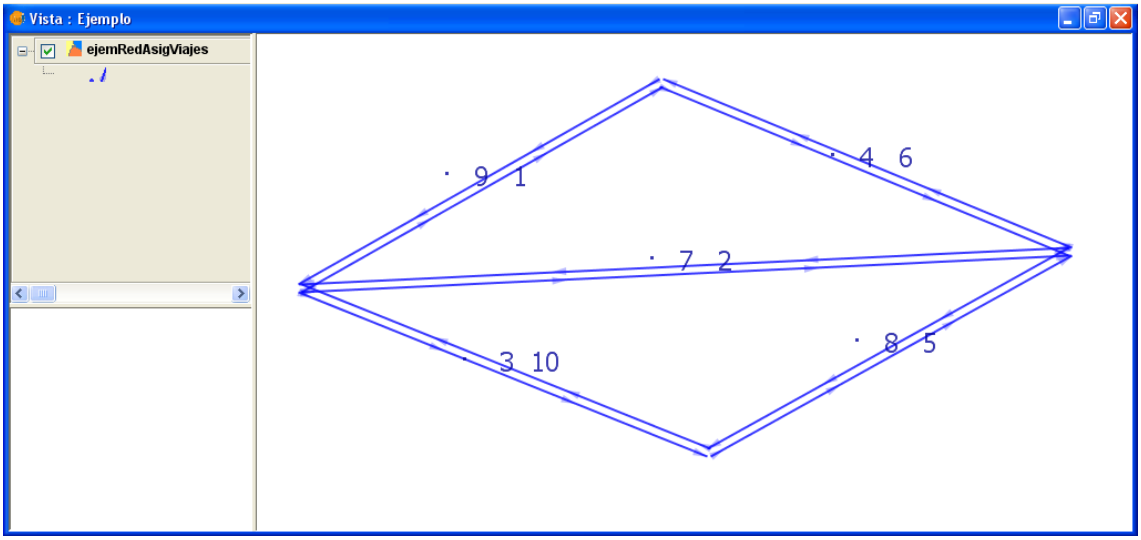
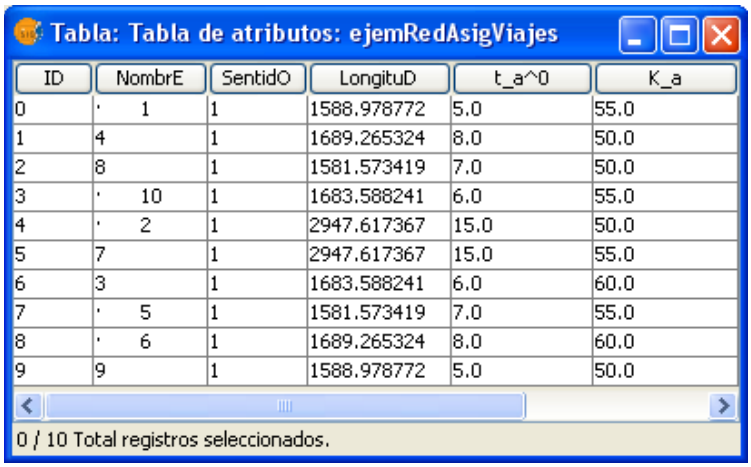


Figura 22. Capa shape tipo línea, entidad gráfica de la red

A la Tabla de atributos de la capa activa, se le edita para añadirle los campos de Costo a flujo libre (t_a^0) y Nivel de servicio (K_a), que se dan como dato. Es importante que el nombre del campo no sea muy extenso, que no rebase los 10 caracteres, puesto que al momento de guardarse la tabla, el nombre es recortado a sólo 10 caracteres y puede ocurrir un error al ejecutar el módulo *Asignación por equilibrio*, por no coincidir el nombre extenso del campo con el nombre recortado a 10 caracteres.



ID	NombrE	SentidO	LongituD	t_a^0	K_a
0	1	1	1588.978772	5.0	55.0
1	4	1	1689.265324	8.0	50.0
2	8	1	1581.573419	7.0	50.0
3	10	1	1683.588241	6.0	55.0
4	2	1	2947.617367	15.0	50.0
5	7	1	2947.617367	15.0	55.0
6	3	1	1683.588241	6.0	60.0
7	5	1	1581.573419	7.0	55.0
8	6	1	1689.265324	8.0	60.0
9	9	1	1588.978772	5.0	50.0

Figura 23. Adición de campos a la Tabla de atributos de la capa de la red

El campo Nombre del arco es un atributo de la línea que se utiliza para identificar el arco de acuerdo con la figura de la red de ejemplo, sin embargo, el campo que se utiliza a nivel código es el del ID, que inicia en cero su numeración. El campo del Sentido se define de acuerdo a la digitalización de la línea del arco, es 1 si el sentido coincide con el digitalizado, es 2 si el sentido es contrario al digitalizado, si se quiere que un arco sea de doble sentido es 0, y ese arco genera dos aristas, una para cada sentido.

Se recomienda que se dibuje un arco para cada sentido de circulación, con el fin de graficar el volumen de viajes que fluye en cada sentido. Recomendación que será requisito si los arcos tienen Costos o Niveles de servicio diferentes para cada sentido, como es el caso del ejemplo.

Posteriormente se construye la red, para lo cual puede consultarse la documentación de gvSIG “Redes 0.1.0” (Grupo gvSIG, 2010), o en el Anexo 7 del “Curso de gvSIG 1.9” (CIT. Generalitat Valenciana, 2009), sobre cómo generar la topología o cargar la red desde un archivo. La configuración de los ejes de la red deberá estar cuidadosamente dibujada para evitar correcciones topológicas.

La particularidad que se necesita en la red, es que el Costo principal y secundario de cada arco, sean los valores correspondientes al Costo a flujo libre (t_a^0), asignándolos al momento de generar la topología de red.

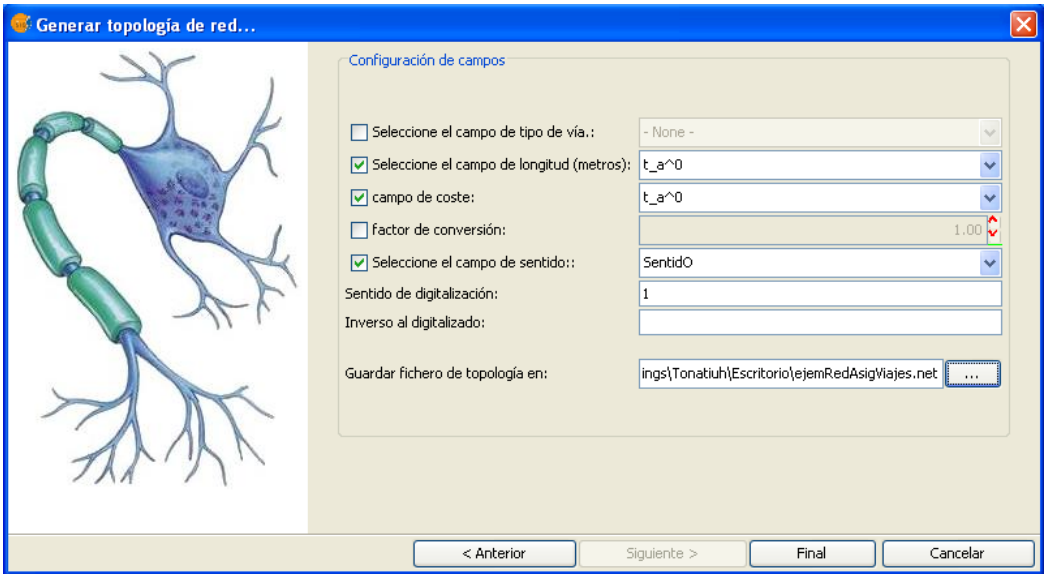


Figura 24. Asignación de Costo a flujo libre (t_a^0) como Costo principal y Costo secundario

Finalmente se carga la red generada, o si previamente se ha creado una con las características mencionadas, se carga la red desde el archivo. Para que se habilite la extensión *Asignación por Equilibrio* debe haber una red asociada a la capa activa.

No es recomendable el uso de Costos de giro ni Prohibición de arcos en la red a utilizar, ya que a nivel código se modifica la red continuamente y no se evaluó su comportamiento con tales características.

3.3.2. Orígenes y destinos de viajes

Las capas correspondientes a los orígenes y destinos de los viajes, serán shape tipo punto. Éstas se pueden generar con el *Gestor de paradas* del menú *Red*, o puede ser una capa de puntos correspondiente a los centroides de las Áreas Geo Estadísticas Básicas (AGEBS), por ejemplo.

Los puntos orígenes y destinos de nuestro cálculo no necesariamente han de estar situados sobre la red de ejes con la que estamos trabajando. Para ello se define el parámetro Tolerancia, que es la distancia máxima que se tendrá en cuenta desde el nodo más cercano al punto de cálculo. Si la distancia entre el punto y el nodo de la red más cercano, es mayor que dicha tolerancia, ese punto no será tenido en cuenta en el cálculo, y se mostrará un mensaje que informe al usuario que hay puntos fuera de la red.

Para mayor precisión en los datos de entrada en cuanto a orígenes y destinos se refiere, puede hacerse uso de la herramienta *Situar parada encima de nodo* y definir los puntos orígenes sobre los nodos de la red. Una vez ubicados los puntos se guardan, con el botón *Salvar paradas*, en un archivo formato *shp*, seleccionando la opción que inserta en la vista la capa creada. Se repite el mismo procedimiento para los puntos destinos.

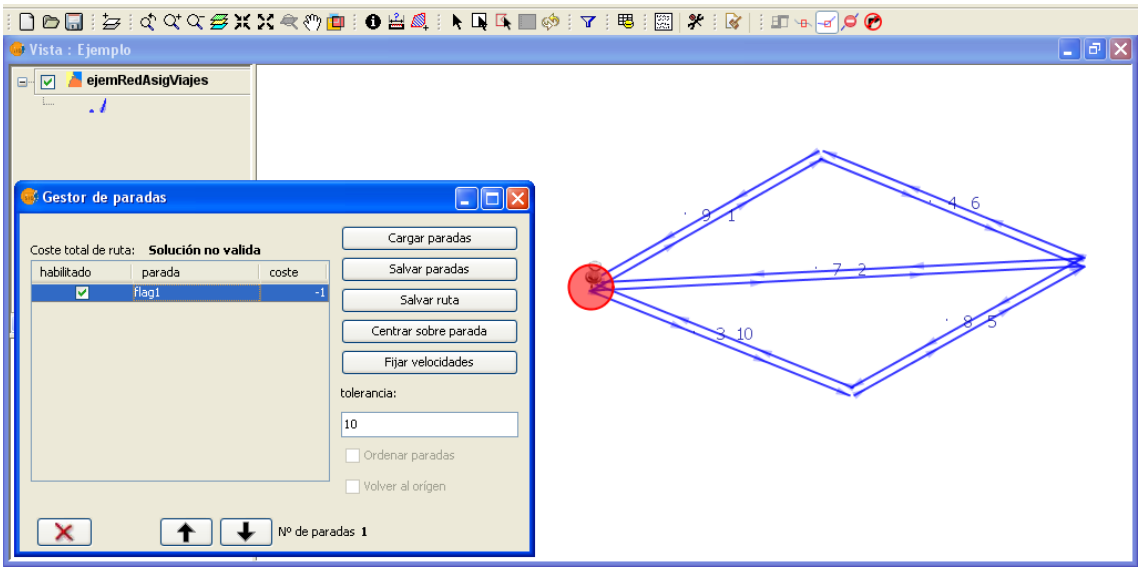


Figura 25. Generar capa de puntos con el Gestor de paradas

Modificando la simbología de las capas de puntos, se distinguen los orígenes en verde de los destinos en amarillo. Es importante notar que la numeración del Identificador (ID) de los puntos debe iniciar en cero, ya que es el ID del punto con el que se construirá la Matriz de viajes y con el que se realizarán los cálculos de Ruta mínima. En la figura se observan, etiquetados con su ID, en verde el punto origen correspondiente al lugar de salida de los viajes, y en amarillo los puntos destino correspondientes a los lugares de llegada de los viajes.

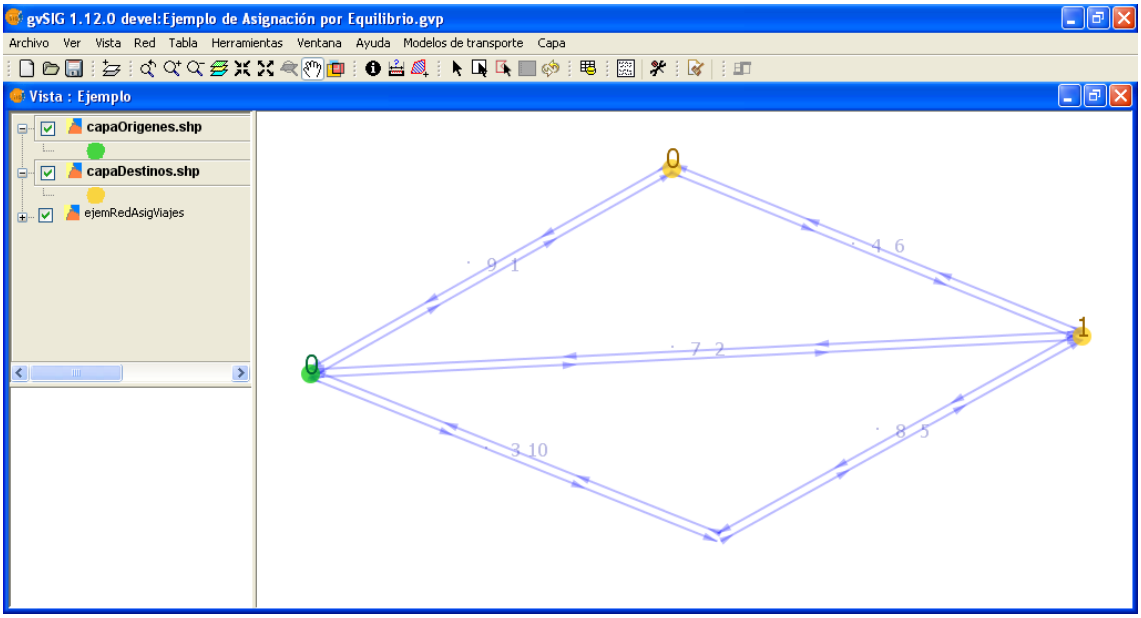


Figura 26. Capas de puntos Origen y Destino

El ingreso de estos datos en la Interfaz gráfica de usuario es mediante una lista desplegable que muestra las capas shape tipo punto que hay en la Tabla de Contenidos, eligiendo la correspondiente a los *Orígenes* y la correspondiente a los *Destinos*. En seguida se indica la *Tolerancia* en unidades de *Metro*; por defecto tiene 50 metros, pero puede ser modificado de acuerdo a las necesidades del analista.

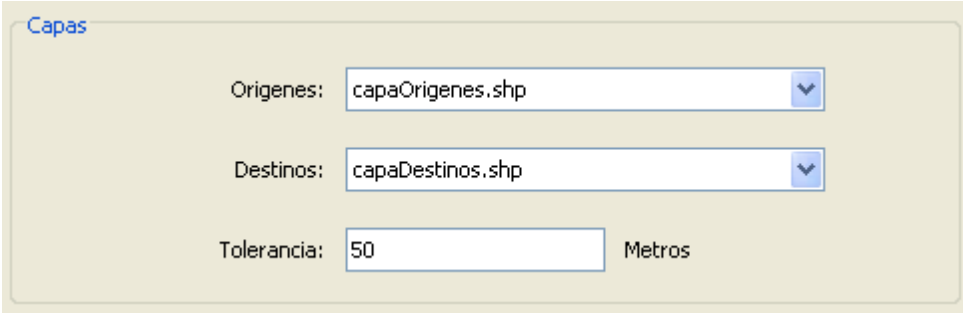


Figura 27. Ingreso de Orígenes y Destinos de viajes, y Tolerancia de puntos a la red

3.3.3. Costo a flujo libre y Nivel de servicio de los arcos

Los datos de *Costo a flujo libre* t_a^0 y *Nivel de servicio* K_a, previamente han sido añadidos a la Tabla de atributos de la capa de la red. Para ingresarlos al algoritmo, en la Interfaz gráfica de usuario se despliega una lista con aquellos campos, de la Tabla de atributos de la red, cuyo valor es numérico, seleccionando el que corresponda al valor solicitado, como se muestra en la figura. Adicionalmente se ingresa *Épsilon*, el valor numérico que determina la precisión de convergencia a la solución del problema de asignación a la red;

por defecto tiene $\varepsilon = \pm 0.01$, pudiendo ser modificado para una mayor o menor precisión, reflejándose en el número de iteraciones y el tiempo de ejecución. No se recomienda el uso de valores muy pequeños como $\varepsilon = \pm 0.00001$, ya que muy posiblemente nunca llegue tal nivel de convergencia.

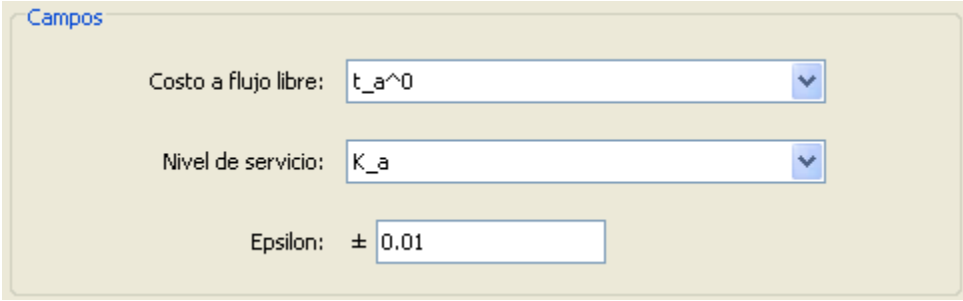


Figura 28. Ingreso de las características de los arcos y Precisión de convergencia a la solución

3.3.4. Matriz de viajes

La *Matriz de viajes* se ingresa mediante un archivo en texto plano con extensión *.txt*. El contenido del archivo se estructura en tres columnas separadas por un espacio tabulador (\t), como se muestra en la figura. La primer columna debe corresponder al ID (Número Identificador o Índice) del punto origen i del viaje, la segunda columna al ID del punto destino j del viaje, y la tercera columna debe corresponder al número de viajes de i a j. El archivo de texto puede tener más columnas, pero sólo se leerán las tres primeras; también pueden no estar en orden, ascendente o descendente.

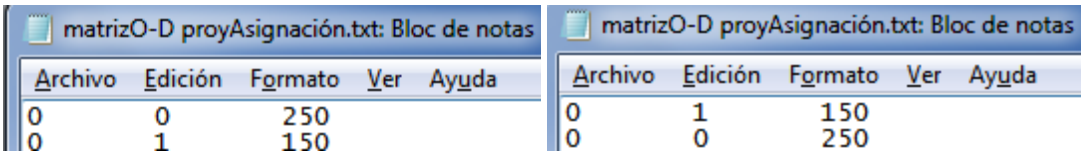


Figura 29. Opciones de Matriz de viajes

Los requisitos principales son que: coincida el ID de los puntos con el número de viajes, las columnas estén separadas por un espacio tabulador (\t), las tres primeras columnas contengan la información solicitada arriba y que el archivo sea de texto plano.

A nivel código se crea una matriz de dimensiones $m \times n$ (Orígenes m $\times$ Destinos n), inicializada en ceros, donde solamente se modifican en la matriz los valores $[i, j]$ leídos en el archivo, quedando el resto en cero. Esto facilita la escritura de la matriz, puesto que sólo será necesario indicar los pares origen-destino que contengan viajes.

Para ingresar el archivo de la *Matriz de viajes* sólo basta ir al directorio y abrir el archivo correspondiente, usando el botón con puntos suspensivos (...).

Figura 30. Ingreso de Matriz de viajes y Directorio donde se guardará el resultado.

Por último, se selecciona la ruta donde se guardará el resultado, usando el botón correspondiente de puntos suspensivos; al escribir el nombre con que se guardará el archivo hay que agregarle la extensión *.txt*, para facilitar su apertura en un Bloc de notas.

3.4. EJECUCIÓN DEL ALGORITMO FRANK-WOLFE

Para dar inicio a la ejecución del algoritmo Frank-Wolfe se da clic en el botón Aceptar. Una vez iniciada la ejecución o antes de iniciarla, si por cualquier motivo el usuario desea cancelarla, lo puede hacer dando clic en el botón Cancelar.

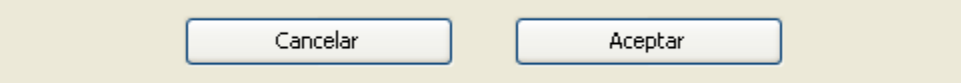


Figura 31. Botones para Cancelar o Iniciar la ejecución del algoritmo Frank-Wolfe

Una vez terminada la ejecución de la extensión, se genera un archivo de texto en el directorio indicado previamente. El archivo contiene un informe, donde al inicio se imprimen los valores de: Nivel de servicio, Costo a flujo libre, el contador $k = 0$ y la Demanda del arco; seguido de los valores dentro de los pasos iterativos: Costo de arco, Demanda de arco auxiliar, Theta, Demanda de arco actualizada y Valor de convergencia; al final del informe aparece la leyenda: Fin.

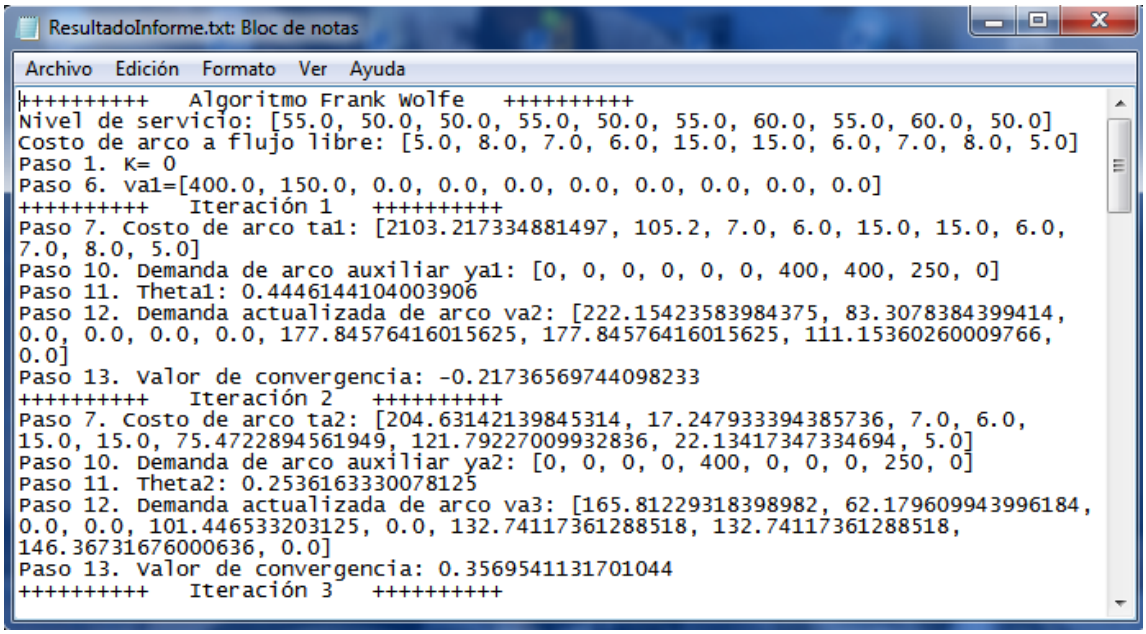


Figura 32. Archivo generado, parte inicial

La solución del problema de asignación de viajes a la red, está en los valores de la Demanda de arco actualizada v_a^{k+1}, paso 12 de la última iteración. El primer valor ubicado en la posición cero del arreglo, 171.67, corresponde al volumen de viajes en el arco con ID cero; el valor ubicado en la posición uno del arreglo, 37.78, corresponde al volumen de viajes en el arco con ID uno; así sucesivamente hasta el último valor.

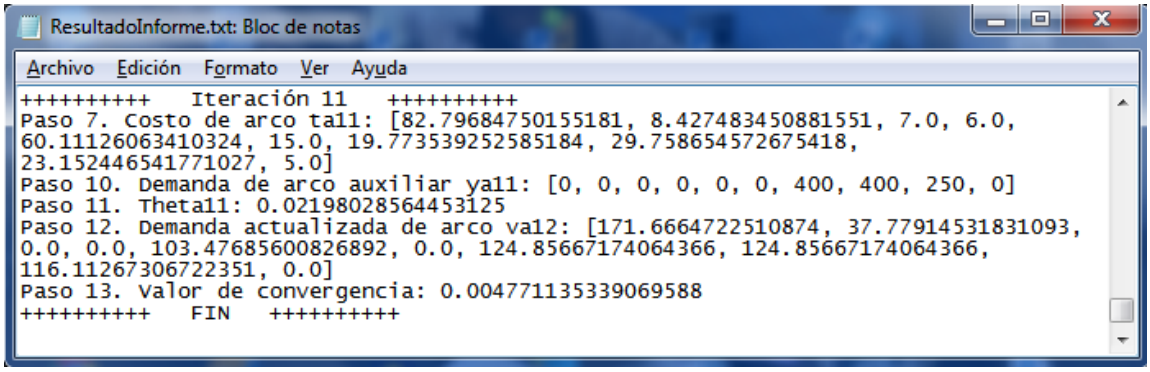
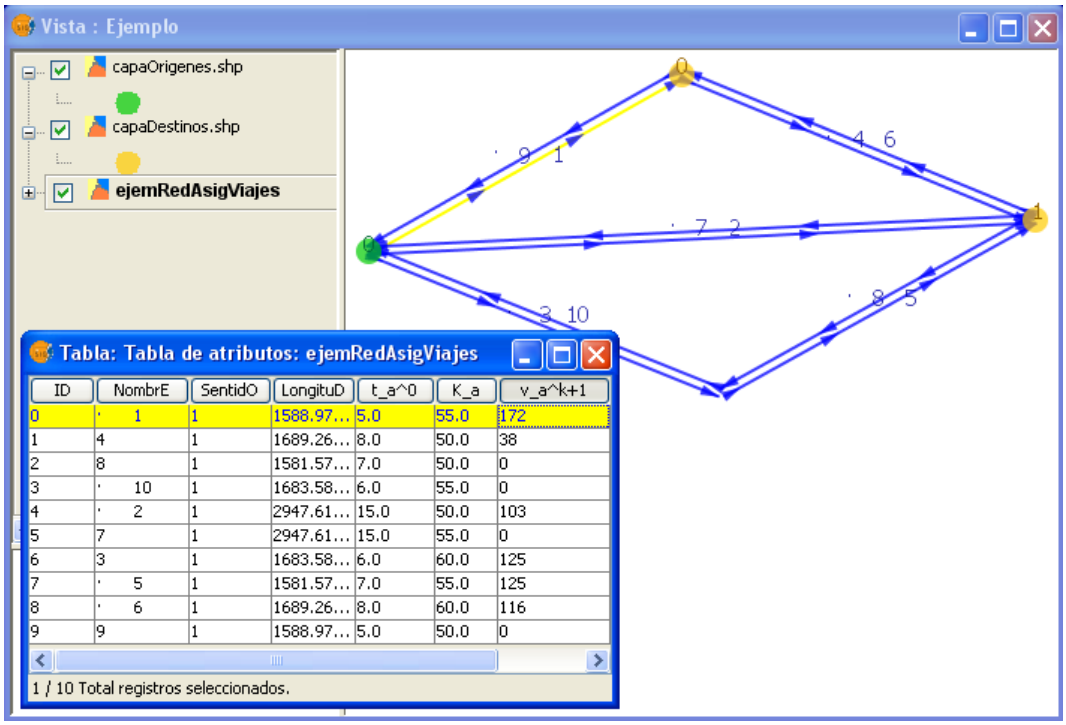


Figura 33. Archivo generado, parte final

3.5. VISUALIZACIÓN DE ASIGNACIÓN DE VIAJES A LA RED

Los valores de la Demanda de arco actualizada se redondean y transcriben a la Tabla de atributos de la capa de la red, quedando de la siguiente manera.



ID	NombrE	SentidO	LongituD	t_a^0	K_a	v_a^k+1
0	1	1	1588.97...	5.0	55.0	172
1	4	1	1689.26...	8.0	50.0	38
2	8	1	1581.57...	7.0	50.0	0
3	10	1	1683.58...	6.0	55.0	0
4	2	1	2947.61...	15.0	50.0	103
5	7	1	2947.61...	15.0	55.0	0
6	3	1	1683.58...	6.0	60.0	125
7	5	1	1581.57...	7.0	55.0	125
8	6	1	1689.26...	8.0	60.0	116
9	9	1	1588.97...	5.0	50.0	0

Figura 34. Tabla de atributos con valores de Demanda de arco actualizada

Para visualizar gráficamente los valores recién añadidos, se editan las propiedades de simbología de la capa, utilizando símbolos graduados para la representación de los valores del campo correspondiente a la Demanda de arco actualizada (v_a^k+1).

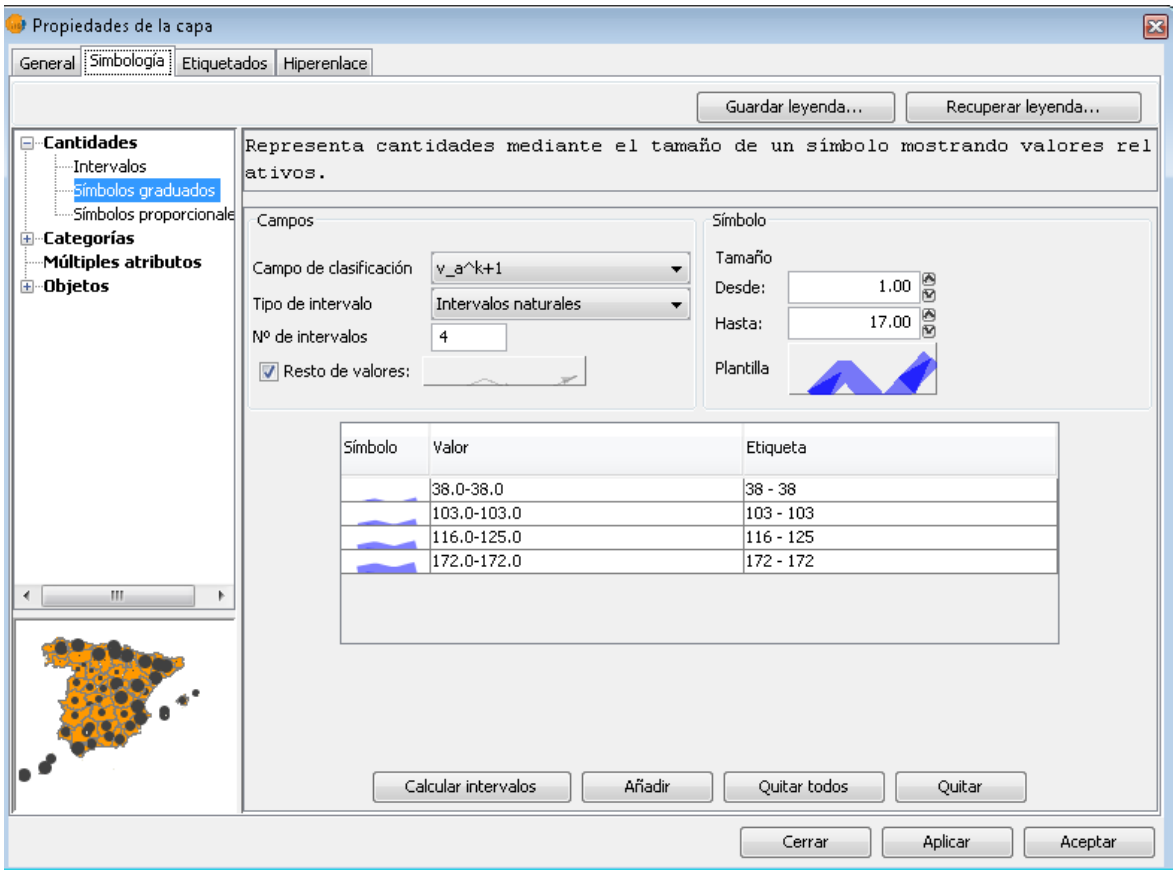


Figura 35. Configuración de las propiedades de simbología de la capa

Con la configuración mostrada en la figura anterior, se logra visualizar gráficamente la asignación de viajes a la red, de tal manera que el grosor de las líneas corresponde al flujo en cada arco.

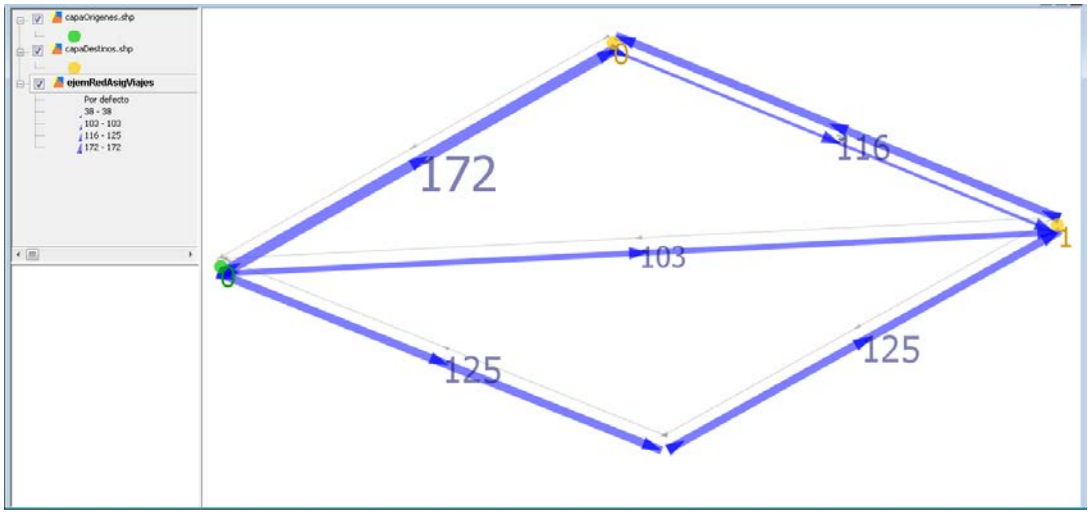


Figura 36. Visualización gráfica de asignación de viajes a la red, $\varepsilon = \pm 0.01$

3.6. COMPARACIÓN DE SOLUCIONES

En la solución mostrada en el libro, se observa que del nodo 1 salen 401 viajes, de los cuales 250 llegan al nodo 2 y 151 viajes llegan al nodo 4. De acuerdo al planteamiento del problema, son 400 viajes los que se distribuyen en la red, sin embargo, es por cuestiones de redondeo de números durante el proceso de cálculo que hay una pequeña afectación.

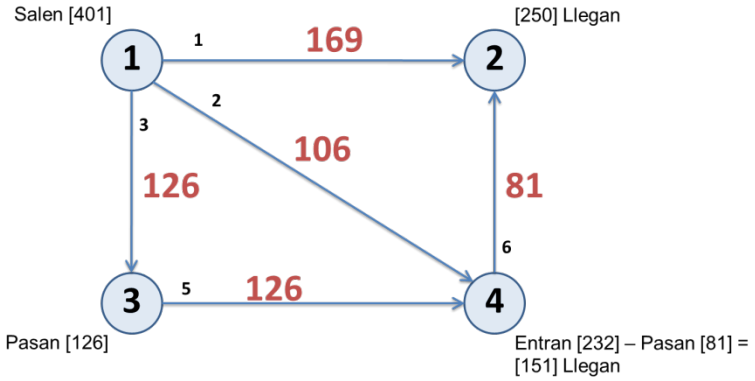


Figura 37. Solución de asignación de viajes a la red del libro de Norbert Oppenheim

En la solución del módulo *Asignación por equilibrio*, con $\varepsilon = \pm 0.01$, mostrada en la siguiente figura, hay una diferencia de viajes entre el nodo 2 y el 4; se observa que restando 38 de 116 resultan 78 viajes que salen del nodo 4 y llegan al nodo 2.

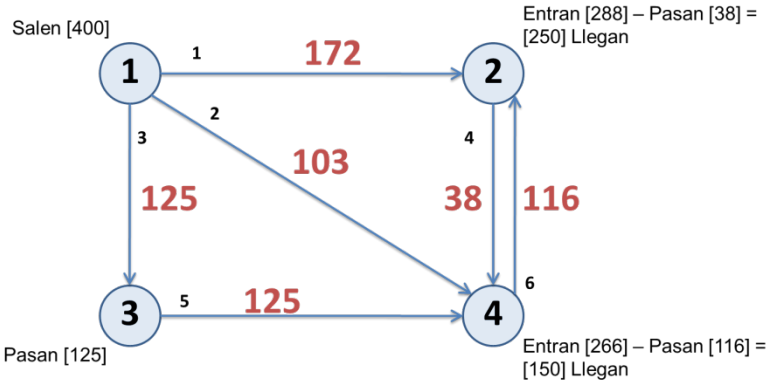


Figura 38. Solución del módulo Asignación por equilibrio, $\varepsilon = \pm 0.01$

Para acercarnos a la solución mostrada en el libro, se aumenta la precisión modificando el valor de $\varepsilon = \pm 0.0001$, obteniendo la siguiente solución.

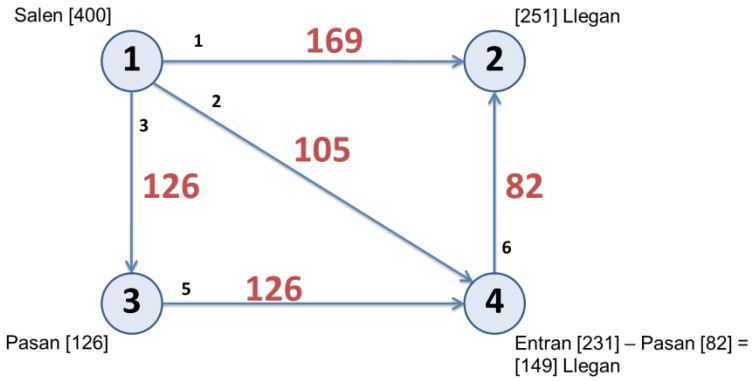


Figura 39. Solución del módulo Asignación por equilibrio, $\varepsilon = \pm 0.0001$

La correspondiente visualización en gvSIG se muestra en la siguiente figura.

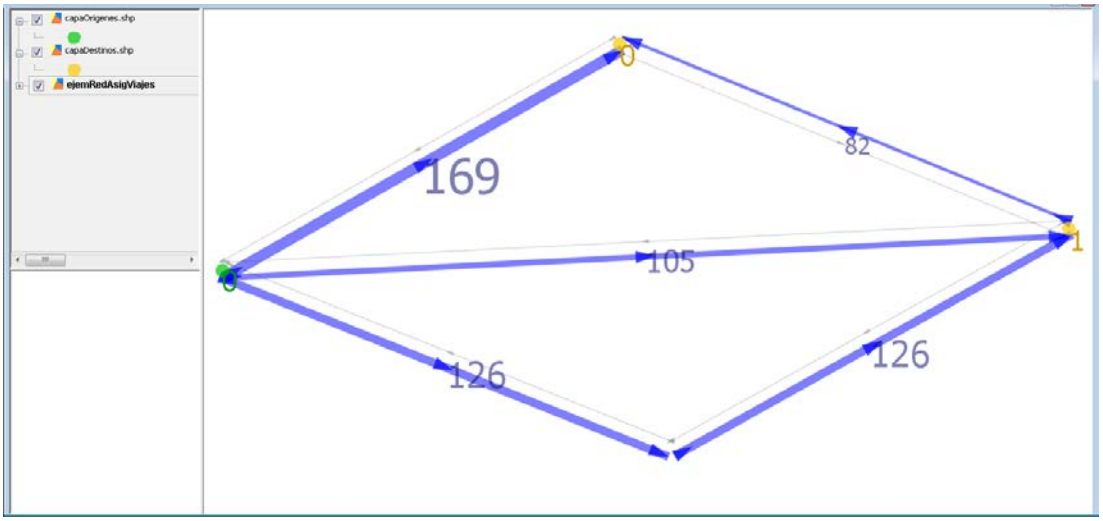


Figura 40. Visualización gráfica de asignación de viajes a la red, $\varepsilon = \pm 0.0001$

Comparando las soluciones, se aclara que las variaciones dependen: del valor de convergencia establecido épsilon ε, del número de decimales con que se hayan manejado los datos y de la precisión en el cálculo del valor de Theta.

En el ejercicio mostrado, se observa que a medida que se disminuye el valor de épsilon, es decir, se aumenta el nivel de convergencia, el flujo en el arco 4 tiende a cero y el flujo del arco 6 tiende a la diferencia entre el arco 6 y 4. Este hecho hace notar la ventaja del uso del algoritmo Frank-Wolfe en su tendencia hacia una solución única.

3.7. RESUMEN

La manera en que se visualiza gráficamente la solución en gvSIG permite al analista una pronta interpretación de la distribución de viajes en la red.

Con la comparación de soluciones, se constata la correcta funcionalidad del módulo *Asignación por equilibrio*, quedando para desarrollos futuros la adición de funcionalidades como: el ingreso automático de los valores de solución de Demanda de arco actualizada en la Tabla de atributos de la capa de la red, características especiales de la red como costos de giro y prohibiciones de paso, así como otros modelos de transporte para ir generando una herramienta que facilite una planeación y gestión adecuada del transporte público y privado.

En la siguiente figura se muestra un diagrama de flujo para el uso del módulo *Asignación por equilibrio*.

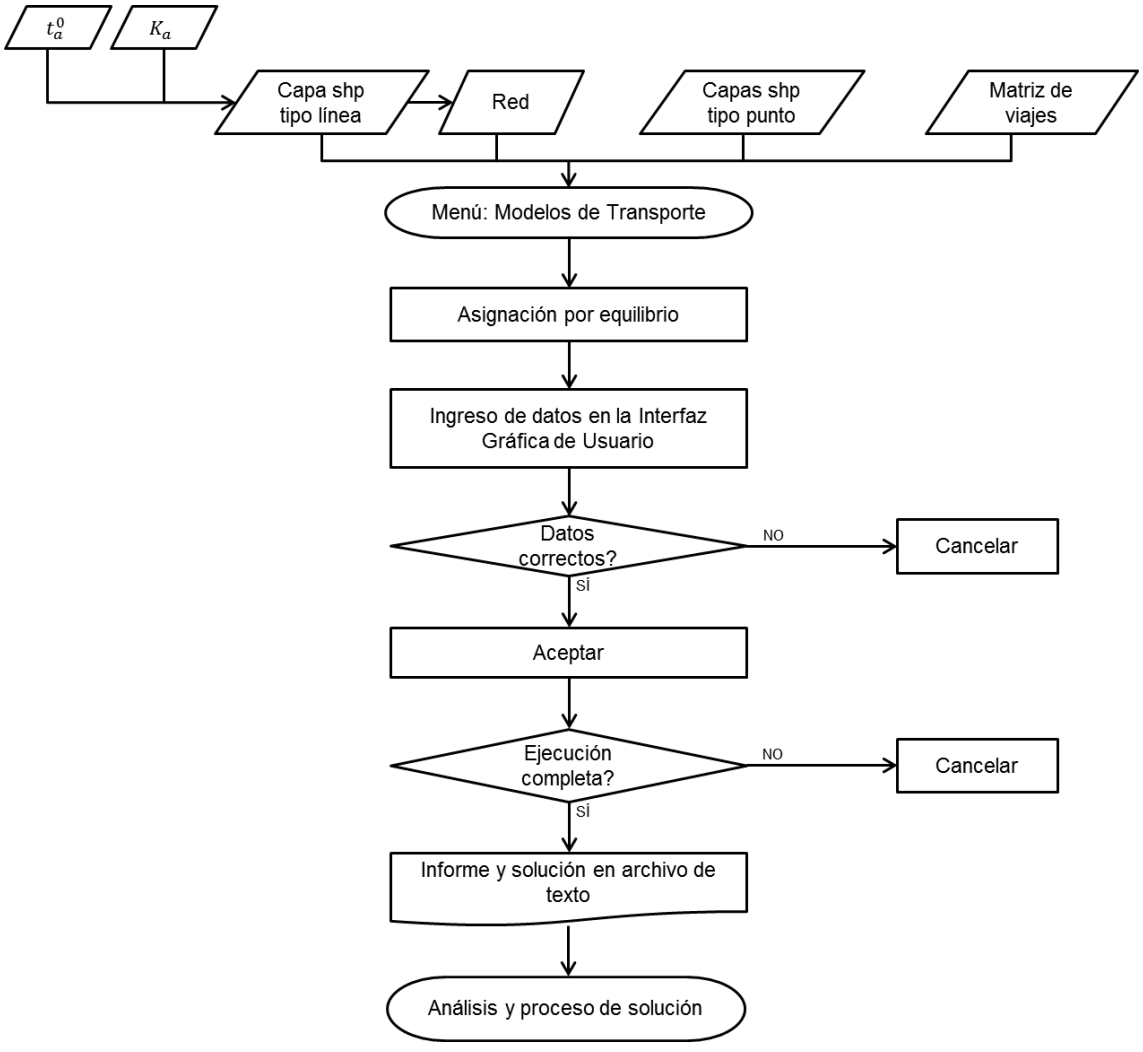


Figura 41. Diagrama de flujo para el uso del módulo Asignación por equilibrio

CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

4.1. INTRODUCCIÓN

En este capítulo se exponen las conclusiones más importantes obtenidas a lo largo del desarrollo del módulo *Asignación por equilibrio*. Se incluyen recomendaciones que pueden ser consideradas para usuarios o desarrollos futuros, relacionados con los Sistemas de información geográfica aplicados al transporte.

4.2. CONCLUSIONES

El desarrollo del módulo *Asignación por equilibrio* cumple con el objetivo de este trabajo, ya que se ejecuta adecuadamente sobre el Sistema de información geográfica gvSIG Desktop y proporciona soluciones confiables que pueden ser representadas gráficamente con símbolos graduados que reflejen en el grosor de línea la cantidad de viajes en cada arco, facilitando el análisis y toma de decisiones.

La nueva funcionalidad aportada, resuelve satisfactoriamente la asignación de viajes a la red bajo condiciones de congestionamiento con el algoritmo de Frank-Wolfe, ampliando las posibilidades de análisis con gvSIG, lo que puede alentar a planeadores de transporte y nuevos usuarios en utilizar esta herramienta para la planeación y gestión del transporte mediante diversos análisis, hipotéticos o no, como: impacto de nuevo equipamiento urbano, impacto de nueva infraestructura vial, emisión de contaminantes por arco, flujo vehicular estimado en cada arco de la red vial, etc.

La mayoría de las empresas, gobierno y centros de investigación, tienen una dependencia muy fuerte con software comercial de transporte. Por ello, es importante la aportación de este tipo de funcionalidades a un SIG libre y es algo que a nivel nacional e internacional traería mucho beneficio al campo de transporte. Algunos software abusan en precios y condiciones de compra, aprovechándose de que no hay muchas alternativas a sus productos, y sobre todo que no hay alternativas libres.

El módulo *Asignación por equilibrio*, como parte de gvSIG Desktop, puede ser instalado en diversos Sistemas operativos de cómputo: Linux, Windows, Mac OS X. Por estar programado en Java puede ejecutarse virtualmente casi en cualquier computadora, sin costo monetario para su instalación. El código fuente del módulo está disponible en internet bajo licencia GNU General Public License, versión 2, en las siguientes páginas:

http://extmodeltransport.blogspot.mx/

https://code.google.com/p/ext-model-transport/

El proceso de aprendizaje para el desarrollo del módulo de software, ha dejado en el autor un conocimiento valioso que se comparte en este trabajo, proporcionando un compendio de material consultado disponible en internet, necesarios para un proceso de desarrollo más liviano.

El código fuente generado para el módulo *Asignación por equilibrio*, puede ser reutilizado como base o complemento para nuevas funcionalidades, sumándose a la promoción de uso, aprendizaje y crecimiento de software libre. Fue el análisis del código abierto lo que permitió al autor, aprender a programar sobre gvSIG y lograr que ahora haya una alternativa de software libre con la funcionalidad particular para transporte de asignar viajes a la red bajo condiciones de congestionamiento.

Ligado a lo anterior, gvSIG tiene excelentes cualidades como material didáctico para la enseñanza de programación, Sistemas de información geográfica, análisis espacial, ingeniería civil, por mencionar algunos. Su ventaja como software libre genera oportunidades para la educación y la administración pública, ya que permite personalizarlo y adaptarlo a necesidades específicas que facilitan su uso.

4.3. RECOMENDACIONES

El código de la extensión *Asignación por equilibrio*, puede ampliarse para modelos de asignación dinámica de tráfico, añadiendo funciones de costo con demora al algoritmo de Frank-Wolfe, que de acuerdo con (Londoño & Lozano, 2012), pueden ser: funciones combinadas de Webster y BPR, y Webster y Akcelik; con la posibilidad de ser implementado en tiempo real mediante una integración con Sistemas inteligentes de transporte.

También queda para desarrollos futuros, la disposición de más algoritmos para encontrar la ruta mínima, que optimicen el desempeño computacional del módulo *Asignación por equilibrio*; así como otros métodos de asignación como el método basado en el origen, que de acuerdo con (Bar-Gera & Boyce, 2004), es más rápido computacionalmente que el método Frank-Wolfe, especialmente para redes de gran escala.

Para un desarrollo a mediano plazo está: que los valores de asignación de viajes a la red se ingresen a la Tabla de atributos de la capa de manera automática desde código, que se pueda trabajar con redes que necesiten el uso de costos de giro y prohibición de paso, que pueda ser la misma capa la de origen y destino de viajes, y que sea factible el uso de dos aristas en un arco.

REFERENCIAS

Acevedo, V. (01 de Junio de 2010). *Guía de referencia para gvSIG 1.1*. Recuperado el 02 de Mayo de 2013, de https://gvsig.org/web/docdev/reference

Acevedo, V. (08 de Marzo de 2011). *Construir gvSIG desde el repositorio SVN*. Recuperado el 02 de Mayo de 2013, de https://gvsig.org/web/docdev/building-from-svn/building-gvsig-from-svn

Acevedo, V. (08 de Febrero de 2011). *Guía de extensiones para gvSIG 1.9*. Recuperado el 02 de Mayo de 2013, de https://gvsig.org/web/docdev/trunk-extensions-guide

andami API. (s.f.). Recuperado el 02 de Mayo de 2013, de http://projetsigle.free.fr/ressources/documentation/gvsig/gvSIG-0_4-doc-en/andami-api/index.html

Asociación gvSIG. (s.f.). *gvSIG Desktop*. Recuperado el 02 de Mayo de 2013, de http://www.gvsig.com/productos/gvsig-desktop

Banister, D. (2002). *Transport Planning.* Spon Press.

Bar-Gera, H., & Boyce, D. (2004). Origin-based network assignment. En M. Patriksson, & M. Labbé, *Transportation planing. State of the Art.* Kluwer Academic Publishers.

Bauzer, C., & Zimányi, E. (2009). Preface to SeCoGIS. En C. A. Heuser, & G. Pernul, *Advances in Conceptual Modeling - Challenging Perspectives.* Springer.

Bonsall, P. W. (2006). Principles of transport analysis and forecasting. En C. A. O'Flaherty, *Transport Planning and Traffic Engineering.* Elsevier.

Carrera, M. (30 de Abril de 2013). *Funcionalidades, gvSIG-Desktop*. Recuperado el 02 de Mayo de 2013, de http://www.gvsig.org/web/projects/gvsig-desktop/funcionalidades

CIT. Generalitat Valenciana. (2, 3 y 4 de Diciembre de 2009). *Curso de gvSIG 1.9.* Recuperado el 02 de Mayo de 2013, de http://downloads.gvsig.org/download/documents/learning/gvsig-courses/gvsig_des_1.9_u_2/Course_gvSIG_1.9-es.pdf

Cristóbal, E. (s.f.). *Montar gvSIG 1.9 en Eclipse desde el Repositorio.* Recuperado el 02 de Mayo de 2013, de http://downloads.gvsig.org/download/documents/devel-documentation/collaborations/Gvsig-1-9-en-Eclipse.pdf

de Bay, R. A. (2001). *Principles of Geographic Information Systems. An introductory textbook.* ITC Educational Textbook Series.

del Cerro, J. J. (04 de Agosto de 2011). *Guía de referencia para gvSIG 1.1*. Recuperado el 02 de Mayo de 2013, de http://www.gvsig.org/web/docdev/reference/

Escobar, J. C. (s.f.). Notas.Universidad Nacional de Colombia. Posgrado de Ingeniería.

Gavarró, A. (2011). *Adaptación y extensión de un SIG.* Recuperado el 02 de Mayo de 2013, de http://downloads.gvsig.org/download/documents/learning/collaborations/ce_1104_01/Programacion_personalizacion_SIG_2.pdf

Goulias, K. G. (2003). *Transportation systems planning: methods and applications.* CRC PRESS.

Grupo gvSIG. (29 de Octubre de 2010). *Redes 0.1.0*. Recuperado el 02 de Mayo de 2013, de http://www.gvsig.org/web/projects/gvsig-desktop/docs/user/ext/redes/network-analisys-0-1-0/network-analisys-0-1-0/gvsig_freemind_toc_view?content=Redes

gvSIG API. (s.f.). Recuperado el 21 de Julio de 2012, de http://geosysin.iict.ch/html/javadoc-gvsig/allclasses-frame.html

gvSIG Outreach. (2013). *gvSIG Outreach*. Recuperado el 02 de Mayo de 2013, de http://outreach.gvsig.org/case-studies

Laurini, R. (2001). *Information systems for urban planning.* Taylor & Francis.

Londoño, G., & Lozano, A. (2012). Enfoques analíticos en la asignación dinámica de tráfico: reflexiones sobre los avances reales para análisis macroscópico de redes urbanas y consideraciones para una formulación binivel. *Primera Reunión Nacional de Modelos Matemáticos para Transporte.*

Lozano, A. (2012). Problema de asignación de tráfico multiclase y multicriterio. *Primera Reunión Nacional de Modelos Matemáticos para Transporte.*

Madrid, M. (30 de Julio de 2012). *gvSIG-Desktop 1.12. Manual de usuario (en línea)*. Recuperado el 02 de Mayo de 2013, de http://www.gvsig.org/web/projects/gvsig-desktop/docs/user/gvsig-desktop-1-12-manual-de-usuario/

Oppenheim, N. (1995). *Urban travel demand modeling: from individual choices to general equilibrium.* A Wiley-Interscience Publication.

Ortúzar S., J. d. (2000). *Modelos de demanda de transporte.* Alfaomega.

Ortúzar, J. d., & Willumsen, L. G. (2008). *Modelos de transporte.* Universidad de Cantabria.

Pendyala, R. M. (2003). Time Use and Travel Behavior in Space and Time. En K. G. Goulias, *Transportation systems planning: methods and applications.* CRC Press.

Peñarrubia, F. J. (27 de septiembre de 2010). *Manual para desarrolladores gvSIG v1.1*. Recuperado el 02 de Mayo de 2013, de http://www.gvsig.org/web/docdev/manual-para-desarrolladores-gvsig/

Piera, J. (s.f.). *Crear una extensión desde 0 en gvSIG.* Recuperado el 02 de Mayo de 2013, de http://joinup.ec.europa.eu/sites/default/files/Crear_extensiones_en_gvSIG.pdf

Pinheiro, D. S. (Agosto de 2012). *gvSIG's extVRP Documentation.* Recuperado el 02 de Mayo de 2013, de https://vrp-for-gvsig-network-extension.googlecode.com/files/extVRP_Documentation.pdf

Sevilla, L. (01 de Junio de 2010). *Como desarrollar contra unos binarios de gvSIG 1.9*. Recuperado el 02 de Mayo de 2013, de http://www.gvsig.org/web/docdev/docs/desarrollo/comos/desarrollo-contra-binarios

Thill, J.-C. (2000). *Geographic Information Systems in transportation research.* Pergamon.

Valencia, V. G. (s.f.). Notas. Universidad Nacional de Colombia.

Victoria Transport Policy Institute. (5 de Enero de 2012). *TDM Encyclopedia*. Recuperado el 02 de Mayo de 2013, de Transportation Demand: http://www.vtpi.org/tdm/tdm132.htm

APÉNDICE A. CÓDIGO FUENTE

A.1. CLASE ASSIGNMENTEQUILIBRIUMEXTENSION.JAVA

```
/*
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 */

package org.gvsig.graph.mt;

import java.io.File;
import java.util.ArrayList;

import org.gvsig.exceptions.BaseException;
import org.gvsig.graph.core.GraphException;
import org.gvsig.graph.core.GvFlag;
import org.gvsig.graph.core.Network;
import org.gvsig.graph.core.NetworkUtils;
import org.gvsig.graph.mt.gui.AssignmentEqControlPanel;
import org.gvsig.graph.mt.solvers.AssignmentEqTask;

import com.iver.andami.PluginServices;
import com.iver.andami.plugins.Extension;
import com.iver.andami.ui.mdiManager.IWindow;
import com.iver.cit.gvsig.fmap.MapContext;
import com.iver.cit.gvsig.fmap.MapControl;
import com.iver.cit.gvsig.fmap.core.IFeature;
import com.iver.cit.gvsig.fmap.drivers.IFeatureIterator;
import com.iver.cit.gvsig.fmap.layers.FLayer;
import com.iver.cit.gvsig.fmap.layers.FLyrVect;
import com.iver.cit.gvsig.fmap.layers.ReadableVectorial;
import com.iver.cit.gvsig.fmap.layers.SingleLayerIterator;
import com.iver.cit.gvsig.project.documents.view.gui.View;
import com.vividsolutions.jts.geom.Coordinate;
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.MultiPoint;
import com.vividsolutions.jts.geom.Point;

/**
 * @author Tonatiuh Miguel (tonatiuh.sit@gmail.com)
 *
 */
public class AssignmentEquilibriumExtension extends Extension{
```

```
private static MapControl mapControl;

public void initialize() {
   PluginServices.getIconTheme().registerDefault("assignment",
         this.getClass().getClassLoader().
         getResource("images/assignment.png")
      );
}

public void execute(String actionCommand) {

   View v = (View) PluginServices.getMDIManager().getActiveWindow();
   MapContext map = v.getMapControl().getMapContext();
   SingleLayerIterator it = new SingleLayerIterator(map.getLayers());

   if (actionCommand.equals("MTEQUILIBRIUM")) {
      while (it.hasNext())
      {
         FLayer aux = it.next();
         if (!aux.isActive())
            continue;
         Network net = (Network) aux.getProperty("network");

         if ( net != null)
         {
            AssignmentEqControlPanel ctrlDlg =
                  new AssignmentEqControlPanel();
            try {
               ctrlDlg.setMapContext(map);
               PluginServices.getMDIManager().addWindow(ctrlDlg);
               if (ctrlDlg.isOkPressed()) {
                  if (net.getLayer().getISpatialIndex() == null) {
                     System.out.println("Calculando índice espacial " +
                           "(QuadTree, que es más rápido)...");
                     net.getLayer().setISpatialIndex(NetworkUtils.
                           createJtsQuadtree(net.getLayer()));
                     System.out.println("Índice espacial calculado.");
                  }

                  FLyrVect layerOrigins = ctrlDlg.getOriginsLayer();
                  FLyrVect layerDestinations =
                        ctrlDlg.getDestinationsLayer();
                  double tolerance = ctrlDlg.getTolerance();
                  ArrayList<Double> costFreeFlow = ctrlDlg.getCostFF();
                  ArrayList<Double> serviceLevel =
                        ctrlDlg.getServiceLevel();
                  double epsilon = ctrlDlg.getEpsilon();
                  File tripMatrixFile =
                        new File(ctrlDlg.getTripMatrix());
                  File generatedFile =
                        new File(ctrlDlg.getPathGenerateFile());

                  GvFlag[] originFlags =
                        this.putFlagsToNodeOnNetwork
```

```
                        (layerOrigins, net, tolerance);
                    GvFlag[] destinationFlags =
                        this.putFlagsToNodeOnNetwork
                        (layerDestinations, net, tolerance);

                    AssignmentEqTask task = new AssignmentEqTask(net,
                        originFlags, destinationFlags,
                        costFreeFlow, serviceLevel, epsilon,
                        tripMatrixFile, generatedFile);
                    PluginServices.cancelableBackgroundExecution(task);

                    return;
                }
            }catch (BaseException e) {
                e.printStackTrace();
            }
        }
    }

}

public boolean isEnabled() {
    Network net = getNetwork();
    if ( net != null)
    {
        return true;
    }
    else
        return false;
}

public boolean isVisible() {
    IWindow f = PluginServices.getMDIManager().getActiveWindow();
    if (f == null) {
        return false;
    }
    if (f instanceof View) {
        return true;
    }
    return false;
}

public static Network getNetwork(){
    IWindow window = PluginServices.getMDIManager().getActiveWindow();
    if (window instanceof View)
    {
        View v = (View) window;
          mapControl = v.getMapControl();
        MapContext map = mapControl.getMapContext();

        SingleLayerIterator it = new SingleLayerIterator(map.getLayers());
        while (it.hasNext())
        {
            FLayer aux = it.next();
```

```
            if (!aux.isActive())
               continue;
            Network net = (Network) aux.getProperty("network");
            return net;
         }
      }
      return null;
   }

   public MapControl getMapControl(){
      return mapControl;
   }

   private GvFlag[] putFlagsToNodeOnNetwork(FLyrVect layer, Network net,
         double tolerance) throws BaseException {
      ReadableVectorial reader = layer.getSource();
      reader.start();
      IFeatureIterator it = reader.getFeatureIterator();
      int i = 0;
      ArrayList<GvFlag> flags = new ArrayList<GvFlag>();
      while (it.hasNext()) {
         IFeature feat = it.next();
         Geometry geo = feat.getGeometry().toJTSGeometry();
         if (!((geo instanceof Point) || (geo instanceof MultiPoint)))
            continue;

         Coordinate[] coords = geo.getCoordinates();
         for (int j = 0; j < coords.length; j++) {
            GvFlag flag =
                  net.addFlagToNode(coords[j].x, coords[j].y, tolerance);
            if (flag == null) {
               GraphException e = new GraphException("Punto " + i
                     + " fuera de la red. Tolerancia=" + tolerance);
               e.setCode(GraphException.FLAG_OUT_NETWORK);
               throw e;
            }
            System.out.println("Situando flag " + i + " de la capa " +
               layer.getName() + " en idArc=" + flag.getIdArc());
            flags.add(flag);
            flag.getProperties().put("rec", new Integer(i));
            break;
         }
         i++;
      }
      it.closeIterator();
      reader.stop();
      return flags.toArray(new GvFlag[0]);
   }
}
```

A.2. CLASE ASSIGNMENTEQCONTROLPANEL.JAVA

```
/*
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 */

package org.gvsig.graph.mt.gui;

import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Types;
import java.util.ArrayList;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

import com.hardcode.gdbms.driver.exceptions.ReadDriverException;
import com.hardcode.gdbms.engine.values.ValueWriter;
import com.iver.andami.PluginServices;
import com.iver.andami.ui.mdiManager.IWindow;
import com.iver.andami.ui.mdiManager.WindowInfo;
import com.iver.cit.gvsig.fmap.MapContext;
import com.iver.cit.gvsig.fmap.core.FShape;
import com.iver.cit.gvsig.fmap.layers.FLayer;
import com.iver.cit.gvsig.fmap.layers.FLayers;
import com.iver.cit.gvsig.fmap.layers.FLyrVect;
import com.iver.cit.gvsig.fmap.layers.LayersIterator;
import com.iver.cit.gvsig.fmap.layers.SelectableDataSource;

import javax.swing.SwingConstants;

import org.gvsig.exceptions.BaseException;
import org.gvsig.fmap.util.LayerListCellRenderer;
```

```java
/**
 * @author Tonatiuh Miguel (tonatiuh.sit@gmail.com)
 *
 */
public class AssignmentEqControlPanel extends JPanel implements IWindow{

   private JPanel layersPanel, fieldsPanel, filesPanel,
                  buttonsPanel, contentPanel;

   private JLabel lblOrigins, lblDestinations, lblTolerance, lblMeters;
   private JLabel lblCostFF, lblServiceL, lblEpsilon, lblMasMenos;
   private JLabel lblTripMatrix, lblCreateFile;

   private JComboBox cmbOrigins, cmbDestinations, cmbCostFF, cmbServiceL;

   private JTextField txtEpsilon, txtTripMatrix,
                      txtPathGenerateFile, txtTolerance;

   private JButton btnTripMatrix, btnCreateFile, btnCancel, btnRun;

   private MapContext mapContext;
   private WindowInfo wi;
   private boolean okPressed = false;

   private FLyrVect currentLyr;
   private ArrayList<Double> costFF;
   private ArrayList<Double> serviceLevel;

   private DefaultComboBoxModel cmbOriginsModel, cmbDestinationsModel;
   private DefaultComboBoxModel cmbCostFFModel, cmbServiceLModel;

   public static void main(String[] args) throws ClassNotFoundException,
         InstantiationException, IllegalAccessException {
      try {
         UIManager.
         setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      } catch (UnsupportedLookAndFeelException e) {
         e.printStackTrace();
      }
      AssignmentEqControlPanel panel = new AssignmentEqControlPanel();
      JFrame test = new JFrame();

      test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      test.getContentPane().add(panel);
      test.setBounds(panel.getBounds());
      test.setVisible(true);
   }

   public AssignmentEqControlPanel() {
      initComponents();
   }

   private void initComponents(){

      //Layers section
```

```
lblOrigins = new JLabel((_T("Origins")+":"),JLabel.RIGHT);
lblOrigins.setBounds(15, 30, 145, 22);
lblDestinations = new JLabel((_T("Destinations")+":"),JLabel.RIGHT);
lblDestinations.setBounds(15, 70, 145, 22);
lblTolerance = new JLabel((_T("Tolerance")+":"),JLabel.RIGHT);
lblTolerance.setBounds(15, 110, 145, 22);
lblMeters = new JLabel((_T("meters")),JLabel.LEFT);
lblMeters.setBounds(310, 110, 100, 22);
cmbOrigins = new JComboBox();
cmbOrigins.setBounds(170, 30, 240, 22);
cmbDestinations = new JComboBox();
cmbDestinations.setBounds(170, 70, 240, 22);
txtTolerance = new JTextField(50);
txtTolerance.setText("50");
txtTolerance.setBounds(170, 110, 130, 22);

//Fields section
lblCostFF = new JLabel((_T("Cost_free_flow")+":"),JLabel.RIGHT);
lblCostFF.setBounds(15, 30, 145, 22);
lblServiceL = new JLabel((_T("Service_level")+":"),JLabel.RIGHT);
lblServiceL.setBounds(15, 70, 145, 22);
lblEpsilon = new JLabel((_T("Epsilon")+":"),JLabel.RIGHT);
lblEpsilon.setBounds(15, 110, 145, 22);
lblMasMenos = new JLabel((_T("+-")), SwingConstants.CENTER);
lblMasMenos.setBounds(170, 110, 15, 22);
cmbCostFF = new JComboBox();
cmbCostFF.setBounds(170, 30, 240, 22);
cmbServiceL = new JComboBox();
cmbServiceL.setBounds(170, 70, 240, 22);

cmbCostFF.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      int selectedFieldIndex = 0;
      try {
         String selectedField =
               cmbCostFF.getSelectedItem().toString();
         selectedFieldIndex = currentLyr.getRecordset().
               getFieldIndexByName(selectedField);
      } catch (ReadDriverException ex){
         ex.printStackTrace();
      }

      costFF = getValuesCostFF(selectedFieldIndex);
   }
});

cmbServiceL.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      int selectedFieldIndex = 0;
      try {
         String selectedField =
               cmbServiceL.getSelectedItem().toString();
         selectedFieldIndex = currentLyr.getRecordset().
               getFieldIndexByName(selectedField);
      } catch (ReadDriverException ex) {
```

```
            ex.printStackTrace();
        }

        serviceLevel = getValuesServiceLevel(selectedFieldIndex);
    }
});

txtEpsilon = new JTextField();
txtEpsilon.setText("0.01");
txtEpsilon.setBounds(185, 110, 115, 22);

//Files section
lblTripMatrix = new JLabel((_T("Trip_matrix")+":"),JLabel.RIGHT);
lblTripMatrix.setLocation(15, 30);
lblTripMatrix.setSize(145, 22);
lblCreateFile = new JLabel((_T("Create_file")+":"),JLabel.RIGHT);
lblCreateFile.setLocation(15, 70);
lblCreateFile.setSize(145, 22);
txtTripMatrix = new JTextField();
txtTripMatrix.setLocation(170, 30);
txtTripMatrix.setSize(240, 22);
txtPathGenerateFile = new JTextField();
txtPathGenerateFile.setLocation(170, 70);
txtPathGenerateFile.setSize(240, 22);

btnTripMatrix = new JButton();
btnTripMatrix.setText("...");
btnTripMatrix.setLocation(420, 30);
btnTripMatrix.setSize(40, 22);
btnTripMatrix.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JFileChooser dlgTM = new JFileChooser();
        if (dlgTM.showOpenDialog((Component) PluginServices.
                getMainFrame()) == JFileChooser.APPROVE_OPTION)
        {
            txtTripMatrix.setText(dlgTM.getSelectedFile().getPath());
        }
    }
});

btnCreateFile = new JButton();
btnCreateFile.setText("...");
btnCreateFile.setLocation(420, 70);
btnCreateFile.setSize(40, 22);
btnCreateFile.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JFileChooser dlgCF = new JFileChooser();
        if (dlgCF.showSaveDialog((Component) PluginServices.
                getMainFrame()) == JFileChooser.APPROVE_OPTION)
        {
            txtPathGenerateFile.
                    setText(dlgCF.getSelectedFile().getPath());
        }
    }
});
```

```
//Buttons section
btnCancel = new JButton(_T("Cancel"));
btnCancel.setBounds(90, 11, 135, 25);
btnCancel.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent evt) {
      btnCancelActionPerformed(evt);
   }
});

btnRun = new JButton(_T("Run"));
btnRun.setBounds(255, 11, 135, 25);
btnRun.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent evt) {
      btnRunActionPerformed(evt);
   }
});

//JPanels section
layersPanel = new JPanel(null);
layersPanel.setBounds(10, 11, 480, 150);
layersPanel.setBorder(BorderFactory.createTitledBorder(_T("Layers")));
layersPanel.setLayout(null);
layersPanel.add(lblOrigins);
layersPanel.add(lblDestinations);
layersPanel.add(lblTolerance);
layersPanel.add(lblMeters);
layersPanel.add(cmbOrigins);
layersPanel.add(cmbDestinations);
layersPanel.add(txtTolerance);

fieldsPanel = new JPanel(null);
fieldsPanel.setBounds(10, 172, 480, 150);
fieldsPanel.setBorder(BorderFactory.createTitledBorder(_T("Fields")));
fieldsPanel.setLayout(null);
fieldsPanel.add(lblCostFF);
fieldsPanel.add(lblServiceL);
fieldsPanel.add(lblEpsilon);
fieldsPanel.add(lblMasMenos);
fieldsPanel.add(cmbCostFF);
fieldsPanel.add(cmbServiceL);
fieldsPanel.add(txtEpsilon);

filesPanel = new JPanel(null);
filesPanel.setBounds(10, 333, 480, 100);
filesPanel.setBorder(BorderFactory.createTitledBorder(_T("Files")));
filesPanel.add(lblTripMatrix, null);
filesPanel.add(lblCreateFile, null);
filesPanel.add(txtTripMatrix, null);
filesPanel.add(txtPathGenerateFile, null);
filesPanel.add(btnTripMatrix, null);
filesPanel.add(btnCreateFile, null);

buttonsPanel = new JPanel();
buttonsPanel.setLayout(null);
```

```
        buttonsPanel.setBounds(10, 444, 480, 50);
        buttonsPanel.add(btnCancel);
        buttonsPanel.add(btnRun);

        contentPanel = new JPanel(null);
        contentPanel.setSize(500, 500);
        contentPanel.add(layersPanel);
        contentPanel.add(fieldsPanel);
        contentPanel.add(filesPanel);
        contentPanel.add(buttonsPanel);

        this.setLayout(null);
        this.add(contentPanel);
    }

    public void setMapContext(MapContext mapContext) throws BaseException {
        this.mapContext = mapContext;
        FLayers layers = mapContext.getLayers();
        LayersIterator it = new LayersIterator(layers);
        Vector<FLyrVect> arrayLayers = new Vector<FLyrVect>();
        Vector<String> arrayFields = new Vector<String>();
        while (it.hasNext()) {
            FLayer lyr = it.nextLayer();
            if (!lyr.isAvailable())
                continue;
            if (lyr instanceof FLyrVect) {
                FLyrVect lyrVect = (FLyrVect) lyr;
                if ((lyrVect.getShapeType() == FShape.POINT)
                        || (lyrVect.getShapeType() == FShape.MULTIPOINT)) {
                    FLyrVect lyrVectPoint = (FLyrVect) lyrVect;
                    arrayLayers.add(lyrVectPoint);
                }
                if ((lyrVect.getShapeType() == FShape.LINE) &&
                        (lyrVect.isActive())) {
                    this.currentLyr = (FLyrVect) lyrVect;
                    try {
                        String[] aux = currentLyr.getRecordset().getFieldNames();
                        for (int i = 0; i < aux.length; i++) {
                            switch (currentLyr.getRecordset().getFieldType(i)) {
                            case Types.BIGINT:
                            case Types.DECIMAL:
                            case Types.DOUBLE:
                            case Types.FLOAT:
                            case Types.INTEGER:
                            case Types.NUMERIC:
                            case Types.REAL:
                            case Types.SMALLINT:
                            case Types.TINYINT:
                                arrayFields.add(aux[i]);
                                break;
                            }
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
```

```
            }
        }
    }
    cmbOriginsModel = new DefaultComboBoxModel(arrayLayers);
    cmbDestinationsModel = new DefaultComboBoxModel(arrayLayers);
    cmbOrigins.setModel(cmbOriginsModel);
    cmbDestinations.setModel(cmbDestinationsModel);
    cmbOrigins.setRenderer(new LayerListCellRenderer());
    cmbDestinations.setRenderer(new LayerListCellRenderer());

    cmbCostFFModel = new DefaultComboBoxModel(arrayFields);
    cmbServiceLModel = new DefaultComboBoxModel(arrayFields);
    cmbCostFF.setModel(cmbCostFFModel);
    cmbServiceL.setModel(cmbServiceLModel);
}

private ArrayList<Double> getValuesField(int column) {
    ArrayList<Double> values = new ArrayList<Double>();
    SelectableDataSource dataSource = null;
    try {
        dataSource = currentLyr.getRecordset();
    } catch (ReadDriverException ex){
        ex.printStackTrace();
    }
    long count = this.getRowCount(currentLyr);
    if (count<1) {
        JOptionPane.showMessageDialog((Component) PluginServices.
              getMDIManager().getActiveWindow(),
              _T("Field_without_values"));
    } else {
        for (int i = 0; i < count; i++) {
            String value = null;
            try {
                value = dataSource.getFieldValue(i,column).
                      getStringValue(ValueWriter.internalValueWriter);
                values.add(Double.parseDouble(value));
            } catch (ReadDriverException ex){
                ex.printStackTrace();
            }
        }
    }

    return values;
}

private ArrayList<Double> getValuesCostFF(int column) {
    return this.getValuesField(column);
}

private ArrayList<Double> getValuesServiceLevel(int column) {
    return this.getValuesField(column);
}

private long getRowCount(FLyrVect selectedLayer){
    SelectableDataSource dataSource;
```

```
    long numberFields = 0;
    try {
       dataSource = selectedLayer.getRecordset();
       numberFields = dataSource.getRowCount();
    } catch (ReadDriverException e) {
       e.printStackTrace();
    }
    return numberFields;
}

public FLyrVect getOriginsLayer() {
    return (FLyrVect) cmbOrigins.getSelectedItem();
}

public FLyrVect getDestinationsLayer() {
    return (FLyrVect) cmbDestinations.getSelectedItem();
}

public double getTolerance() {
    return Double.parseDouble(txtTolerance.getText());
}

public ArrayList<Double> getCostFF() {
    return costFF;
}

public ArrayList<Double> getServiceLevel() {
    return serviceLevel;
}

public double getEpsilon() {
    return Double.parseDouble(txtEpsilon.getText());
}

public String getTripMatrix() {
    return txtTripMatrix.getText();
}

public String getPathGenerateFile() {
    return txtPathGenerateFile.getText();
}

public WindowInfo getWindowInfo() {
    if (wi == null) {
       wi = new WindowInfo(WindowInfo.MODALDIALOG);
       wi.setWidth(500);
       wi.setHeight(500);
       wi.setTitle(_T("Deterministic_car_route_choice"));
    }
    return wi;
}

public Object getWindowProfile() {
    return WindowInfo.DIALOG_PROFILE;
}
```

```
    private String _T(String str) {
        return PluginServices.getText(this, str);
    }

    private void btnCancelActionPerformed(ActionEvent evt) {
        closeWindow();
    }

    private void btnRunActionPerformed(ActionEvent evt) {
        if (getPathGenerateFile().equalsIgnoreCase(""))
        {
            JOptionPane.showMessageDialog((Component) PluginServices.
                  getMDIManager().getActiveWindow(),
                  _T("Please_enter_a_valid_created_file"));
            return;
        }
        if (getTripMatrix().equals(""))
        {
            JOptionPane.showMessageDialog((Component) PluginServices.
                  getMDIManager().getActiveWindow(),
                  _T("Please_enter_a_valid_trip_matrix_file"));
            return;
        }
        try {
            double tol = getTolerance();
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog((Component) PluginServices.
                  getMDIManager().getActiveWindow(),
                  _T("Please_enter_a_valid_tolerance_number"));
            return;
        }
        try {
            double eps = getEpsilon();
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog((Component) PluginServices.
                  getMDIManager().getActiveWindow(),
                  _T("Please_enter_a_valid_epsilon_number"));
            return;
        }

        okPressed = true;
        closeWindow();
    }

    private void closeWindow() {
        if (PluginServices.getMainFrame() !=null) {
            PluginServices.getMDIManager().closeWindow(this);
        }
    }

    public boolean isOkPressed() {
        return okPressed;
    }
}
```

A.3. CLASE ASSIGNMENTEQTASK.JAVA

```
/*
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 */

package org.gvsig.graph.mt.solvers;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import javax.swing.JComponent;
import javax.swing.JOptionPane;

import org.gvsig.graph.core.GraphException;
import org.gvsig.graph.core.GvEdge;
import org.gvsig.graph.core.GvFlag;
import org.gvsig.graph.core.IGraph;
import org.gvsig.graph.core.Network;
import org.gvsig.graph.solvers.Route;
import org.gvsig.graph.solvers.ShortestPathSolverDijkstra;

import com.hardcode.gdbms.engine.values.IntValue;
import com.iver.andami.PluginServices;
import com.iver.cit.gvsig.fmap.core.IFeature;
import com.iver.utiles.swing.threads.AbstractMonitorableTask;

/**
 * @author Tonatiuh Miguel (tonatiuh.sit@gmail.com)
 *
 */
public class AssignmentEqTask extends AbstractMonitorableTask{

   private Network net;
   private GvFlag[] originFlags;
   private GvFlag[] destinationFlags;
   private ArrayList<Double> ta0;
   private ArrayList<Double> ka;
   private double epsilon;
   private File tripMatrixFile;
   private File generatedFile;
```

```
   private ArrayList<GvEdge> arcs = new ArrayList<GvEdge>();
   private Route route = null;
   private int numArcs;

   private int m;
   private int n;
   private int travelers[][];

   private ArrayList<Double> vaKAux = new ArrayList<Double>();
   private ArrayList<Double> taKAux = new ArrayList<Double>();
   private ArrayList<Integer> yaKAux = new ArrayList<Integer>();
   private ArrayList<Double> vaActKAux = new ArrayList<Double>();
   private double thetaKAux;

   private int countK;
   private double checkKAux;
   private boolean bExit = false;

   private IGraph graph;

   public AssignmentEqTask(Network net,
         GvFlag[] originFlags, GvFlag[] destinationFlags,
         ArrayList<Double> costFreeFlow, ArrayList<Double> serviceLevel,
         double epsilon, File tripMatrixFile, File generatedFile){
      this.net = net;
      this.originFlags = originFlags;
      this.destinationFlags = destinationFlags;
      this.ta0 = costFreeFlow;
      this.ka = serviceLevel;
      this.epsilon = epsilon;
      this.tripMatrixFile = tripMatrixFile;
      this.generatedFile = generatedFile;

      setInitialStep(0);
      setDeterminatedProcess(false);
      setStatusMessage(PluginServices.
            getText(this, "Calculating_Assignment_Equilibrium"));
      setFinalStep(arcs.size());
   }

   public void run() throws Exception {
      try {


         BufferedWriter output =
               new BufferedWriter(new FileWriter(generatedFile, true));

         graph = net.getGraph();

//         creo un arreglo de arcos
         for (int i = 0; i < graph.numEdges(); i++) {
            arcs.add(graph.getEdgeByID(i));
         }
//         define el número de arcos numArcs
```

```
            numArcs = arcs.size();

//          define matriz de viajes
            m = originFlags.length;
            n = destinationFlags.length;
            this.travelers = new int [m][n];
            setTravelersOnMatrix();

//          inicializa ArrayList con n campos en ceros
            initYaKAux0();
            initArray(taKAux);
            initArray(vaActKAux);

//          inicio algoritmo Frank Wolfe
            output.write("++++++++++   Algoritmo Frank Wolfe   ++++++++++");
            output.newLine();
            output.write("Nivel de servicio: "+ka.toString());
            output.newLine();
            output.write("Costo de arco a flujo libre: "+ta0.toString());
            output.newLine();

            countK = 0;
            output.write("Paso 1. K= "+countK);
            output.newLine();

//          paso 5: calcular ruta mínima y yaKAux
            for (int i = 0; i < originFlags.length; i++) {
               for (int j = 0; j < destinationFlags.length; j++) {
                  if (travelers[i][j] == 0)
                     continue;
                  route = null;
                  route = calculateRoute
                        (originFlags[i], destinationFlags[j], net);
                  calculateYaKAux(travelers[i][j], route);
               }
            }

            countK++;

//          paso 6: calcular vaKAux
            vaKAux = castIntToDouble(yaKAux);
            output.write("Paso 6. va"+countK+"="+vaKAux.toString());
            output.newLine();

            while (!bExit) {
               output.write("++++++++++   Iteración "+countK+"   ++++++++++");
               output.newLine();

//               paso 7: calcula costo de arco taKAux
               calculateTaKAux();
               output.write("Paso 7. Costo de arco ta"+countK+": "+
                     taKAux.toString());
               output.newLine();

//               paso 8: calcula ruta mínima, 9: demanda de la ruta, y
```

```
//                  10: demanda de arco yaKAux
            setTaKAuxToNet();
            setYaKAux0();
            for (int i = 0; i < originFlags.length; i++) {
               for (int j = 0; j < destinationFlags.length; j++) {
                  if (travelers[i][j] == 0)
                     continue;
                  route = null;
                  route = calculateRoute
                        (originFlags[i], destinationFlags[j], net);
                  calculateYaKAux(travelers[i][j], route);
               }
            }
            output.write("Paso 10. Demanda de arco auxiliar ya"+
                  countK+": "+yaKAux.toString());
            output.newLine();

//            paso 11: theta
            calculateTheta();
            output.write("Paso 11. Theta"+countK+": "+thetaKAux);
            output.newLine();

//            paso 12: demanda actualizada de arco vaActKAux
            calculateVaActKAux();
            output.write("Paso 12. Demanda actualizada de arco va"+
                  (countK+1)+": "+vaActKAux.toString());
            output.newLine();

//            paso 13: valor de convergencia checkKAux
            calculateCheckKAux();
            output.write("Paso 13. Valor de convergencia: "+checkKAux);
            output.newLine();

            if (Math.abs(checkKAux)<=epsilon) {
               bExit = true;
               break;
            }

            countK++;
            setVaActToVa();
//            ArrayList en CEROS para nueva iteración
            setZero(taKAux);
            setZero(vaActKAux);

         }
         output.write("++++++++++   FIN   ++++++++++");
         output.newLine();
         output.flush();
         output.close();

      } catch (IOException e) {
         e.printStackTrace();
      } catch (GraphException e) {
         e.printStackTrace();
      }
```

```
    }


//   paso 7: Calcula el Costo del Arco taKAux
    private ArrayList<Double> calculateTaKAux() {
        for (int i = 0; i < numArcs; i++) {
            taKAux.set(i, (ta0.get(i)*(1+0.15*
                  (Math.pow((vaKAux.get(i)/ka.get(i)), 4)))));
        }
        return taKAux;
    }

//   paso 8: Asigna los Costos de Arco taKAux a la Red
    private void setTaKAuxToNet(){
        for (int i = 0; i < numArcs; i++) {
            arcs.get(i).setDistance(taKAux.get(i));
            arcs.get(i).setWeight(taKAux.get(i));
        }
    }

//   paso 9: Calcula la Ruta mínima para cada Origen-Destino
    private Route calculateRoute(GvFlag origen, GvFlag destino, Network net)
            throws GraphException {
        ShortestPathSolverDijkstra solver = new ShortestPathSolverDijkstra();
        Route route = null;
        try {
            net.removeFlags();
            solver.setNetwork(net);
            net.addFlag(origen);
            net.addFlag(destino);
            String fieldStreetName = (String) net.getLayer().
                  getProperty("network_fieldStreetName");
            solver.setFielStreetName(fieldStreetName);
            route = solver.calculateRoute();
        } catch (GraphException ex) {
            ex.printStackTrace();
        }
        if (route.getFeatureList().size() == 0) {
            JOptionPane.showMessageDialog((JComponent) PluginServices.
                  getMDIManager().getActiveWindow(),
                  PluginServices.getText(this, "shortest_path_not_found"));
        }
        return route;
    }

//   paso 10: Calcula el valor de Demanda de Arco Auxiliar yaKAux
    private ArrayList<Integer> calculateYaKAux
            (int travelers_i_j, Route route){
        ArrayList featureList = route.getFeatureList();
        for (int i = 0; i < featureList.size(); i++) {
            int id = 0;
            IFeature feature = (IFeature) featureList.get(i);
            id = ((IntValue)feature.
                  getAttribute(Route.ID_ARC_INDEX)).getValue();
            yaKAux.set(id, (yaKAux.get(id)+travelers_i_j));
```

```
      }
      return yaKAux;
   }

//   paso 11: Calcula el valor de theta.
   private double calculateTheta(){
      double a = 0;
      double b = 1;
      double eval = 0;
      boolean bEnd = false;

      while (!bEnd) {
         thetaKAux = (a+b)/2;
         for (int i = 0; i < numArcs; i++) {
            eval += ((yaKAux.get(i)-vaKAux.get(i))*(ta0.get(i))*
                  (1+0.15*(Math.pow(((vaKAux.get(i)+thetaKAux*
                        (yaKAux.get(i)-vaKAux.get(i)))/ka.get(i)), 4))));
         }
         if (0<=eval) {
            b = thetaKAux;
         } else {
            a = thetaKAux;
         }
         if (Math.abs(eval)<1) {
            bEnd = true;
            break;
         }
         eval = 0;
      }
      return thetaKAux;
   }

//   paso 12: Se calcula la Demanda de Arco Actualizada vaActKAux
   private ArrayList<Double> calculateVaActKAux() {
      for (int i = 0; i < numArcs; i++) {
         vaActKAux.set(i,(vaKAux.get(i)+
               thetaKAux*(yaKAux.get(i)-vaKAux.get(i))));
      }
      return vaActKAux;
   }

//   paso 13: Estimar el valor de convergencia.
   private double calculateCheckKAux() {
      checkKAux = 0;
      for (int i = 0; i < numArcs; i++) {
         checkKAux +=((yaKAux.get(i)-vaActKAux.get(i))*
               (ta0.get(i))*(1+0.15*(Math.pow
                     ((vaActKAux.get(i)/ka.get(i)), 4))));
      }
      return checkKAux;
   }

//   Otros métodos
   private ArrayList<Integer> initYaKAux0() {
      for (int i = 0; i < numArcs; i++) {
```

```
            yaKAux.add(i, 0);
        }
        return yaKAux;
    }

    private ArrayList<Integer> setYaKAux0() {
        for (int i = 0; i < numArcs; i++) {
            yaKAux.set(i, 0);
        }
        return yaKAux;
    }

    private ArrayList<Double> castIntToDouble
            (ArrayList<Integer> arrInteger){
        ArrayList<Double> arrDouble = new ArrayList<Double>();
        for (int i = 0; i < arrInteger.size(); i++) {
            arrDouble.add((double)arrInteger.get(i));
        }
        return arrDouble;
    }

    private ArrayList<Double> initArray(ArrayList<Double> arrayEmpty) {
        for (int i = 0; i < numArcs; i++) {
            arrayEmpty.add(i, 0d);
        }
        return arrayEmpty;
    }

    private ArrayList<Double> setZero(ArrayList<Double> arrayData) {
        for (int i = 0; i < numArcs; i++) {
            arrayData.set(i, 0d);
        }
        return arrayData;
    }

    private ArrayList<Double> setVaActToVa(){
        for (int i = 0; i < numArcs; i++) {
            vaKAux.set(i, vaActKAux.get(i));
        }
        return vaKAux;
    }

//   Método que lee el archivo de viajeros y los pasa a una matriz
    private int [][] setTravelersOnMatrix() {
        BufferedReader input = null;
        String linea = null;
        try {
            input = new BufferedReader(new FileReader(tripMatrixFile));
            while ((linea=input.readLine()) != null) {
                int fila;
                int columna;
                String[] numero = linea.split("\t");
                fila = Integer.parseInt(numero[0]);
                columna = Integer.parseInt(numero[1]);
                travelers[fila][columna] = Integer.parseInt(numero[2]);
```

```
            }
            input.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        return travelers;
    }

}
```