IDELabRoute: Librería para la gestión de grafos escalable

F. Campos Gutiérrez [(1)] *J.P. de Castro Fernández* [(1)] *R. García Martín* [(1)]

[(1)] Laboratorio de Infraestructuras de Datos Espaciales (IDELab), Escuela Técnica Superior de Ingenieros de Telecomunicación, Campus Miguel Delibes, Universidad de Valladolid, 47011 Valladolid, fernando.campos@alumnos.es, {juacas,ricgar}@tel.uva.es.

RESUMEN

Las redes son un importante elemento topológico que tiene poco soporte en el software libre. Hay redes que cuentan con millones de nodos, lo que conlleva la necesidad de manejarlas de forma cuidadosa para optimizar los recursos.

Consultando el estado del arte, hemos concluido que existe cierta cantidad de librerías de código abierto disponible, que generalmente emplean un modelo de gestión de los grafos que genera una estructura mallada de objetos en memoria precisando grandes cantidades de memoria y tiempos de puesta en marcha elevados. Estas carencias adquieren especial relevancia cuando se trata de manejar grandes redes. Además las librerías analizadas no suelen ser aptas para procesado multihilo por lo que no pueden usarse en entornos de servidores. Para estos casos hemos puesto en marcha el proyecto IDELabRoute la solución propuesta consiste en una librería genérica de análisis de redes "thread-safe" con gestión dinámica de memoria; para lo cual, se usa una arquitectura modular con gestores de memoria intercambiables, que desde distintas fuentes de almacenamiento persistente (i.e. bases de datos o sistemas de ficheros), maneja grafos de forma dinámica atendiendo a criterios espaciales y/o topológicos. Se trata de una solución de compromiso, puesto que el precio a pagar por la reducción del tamaño de los objetos en memoria es un incremento en el tiempo de respuesta, debido a la gestión de memorias con diversos tiempos de respuesta. Se trata, por tanto, de un sistema de gestión de grafos dinámico que permite manejar grandes modelos de redes de forma escalable, por lo que puede resultar adecuado en entornos con pocos recursos en relación al tamaño total de la red. El primer objetivo práctico del proyecto es proporcionar a la comunidad del GIS libre un servicio WPS para el cálculo de rutas.

Palabras clave: SIG, cálculo de rutas, *grafo dinámico, gestor de memoria, WPS*

INTRODUCCIÓN

Conceptos básicos de teoría de grafos

Un grafo dirigido G está formado por un par (V, E), donde V es un conjunto finito de nodos y $E \subseteq V \times V$ es el conjunto de arcos del grafo. Denominaremos n como el número de nodos del grafo $\{n = |V|\}$ y m como su número de arcos $\{m = |E|\}$. Un camino en G es una secuencia de nodos $u_1, \dots, u_k$ tales que $(u_i, u_{i+1}) \in E$, con

$1 \leq i < k$. Un grafo, en el que no pueden existir múltiples arcos entre dos nodos, puede contener hasta n^2 arcos y si se permite la multiplicidad el número de elementos no tiene una cota superior. Un grafo se considera *disperso* si $m \in O(n)$.

En un grafo dirigido $G = (V, E)$ puede asociarse un peso c_{ij} a los arcos definidos entre los nodos u_i y u_j según una función $l: E \to \mathbb{R}$**,** de forma que el peso asociado a un camino corresponde a la suma del peso de sus arcos. Nótese que, en general, $c_{ij} \neq c_{ji}$. El problema de encontrar el camino más corto entre dos nodos consiste en encontrar el camino de menor peso entre un origen $s \in V$ y un destino $t \in V$.

El algoritmo clásico para el cálculo del camino más corto en un grafo dirigido es el de Dijkstra. Con este algoritmo se consigue un peor caso de $O(m + n \cdot \log(n))$. Sin embargo se puede aprovechar que se trata de un algoritmo basado en etiquetas y puede detenerse cuando se alcanza el nodo destino, por lo que no tiene por qué recorrer necesariamente todo el grafo. Aún así el número de nodos visitados aumenta rápidamente con la distancia y la densidad del grafo.

Por su parte, el algoritmo A* (pronunciado como "A star") introduce una función heurística que determina el orden en que se recorren los nodos durante la búsqueda [1]. Este algoritmo puede reducir el espacio de búsqueda de forma logarítmica, dependiendo de la calidad de la estimación heurística.

El modelo implementado en el proyecto se basa en grafos dirigidos con arcos dobles y múltiples arcos entre nodos. El API de servicio utiliza A* como algoritmo de cálculo de camino más corto puesto que se puede demostrar que el comportamiento de Dijskstra es un caso particular.

El problema de las grandes redes

Los problemas prácticos pueden encontrarse con redes de grandes dimensiones (en [2] se encuentra disponible para libre descarga una colección de más de 40 series de datos correspondientes a grandes redes que contienen, desde decenas de miles de nodos y arcos, hasta decenas de millones de ellos). Considérese, por ejemplo, la red de carreteras de California (ver Tabla 1). En esta red, las intersecciones y finales de vías se representan por nodos y las carreteras que conectan a los anteriores se representan por arcos no dirigidos. La mayor parte de librerías de cálculo de rutas estudiadas se basan en un modelo de gestión de los grafos que generan una estructura mallada de objetos en memoria. Esta aproximación resulta poco escalable, y no es apropiada para el trabajo con grandes redes con millones de nodos, como la comentada en el ejemplo anterior.

Tabla 1: Estadísticas del conjunto del grafo de la red de carreteras de California [5]

n	m	**Diámetro de la red** (*longest shortest path*)	**Coeficiente de *clustering* medio**	**Fracción de triángulos cerrados**
1965206	5533214	850	0.0464	0.06039

Este es el caso de la librería pgrouting [3]. Resulta muy eficaz para redes medianas y cálculos sencillos. Se ejecuta dentro de la propia base de datos y permite utilizar la potencialidad de las consultas relacionales y espaciales de forma muy sencilla, pero

tiene que generar y cargar todo el grafo en memoria por lo que puede provocar problemas en el gestor de base de datos. La ventaja es que se integra bien en las *queries*, a costa de usar un modelo de grafo demasiado simple y poco escalable. Otras librerías analizadas, como la extension *graph*[5] *de GeoTools*[4] adolecen del mismo problema y además no pueden usarse en entornos multi-hilo.

Librería de grafos IDELabRoute

La librería de grafos IDELabRoute[6], desarrollada en el IDELab (Laboratorio de Infraestructuras de Datos Espaciales), propone una solución más escalable capaz de manejar grandes modelos de redes. Para ello utiliza un mecanismo de carga dinámica que permite ir cargando los objetos en memoria a medida que se van necesitando y descartando los que ya no son convenientes.

En esta librería, el cálculo de la ruta más corta (de menor coste) mediante un algoritmo de cálculo de rutas (i.e. el algoritmo A*) en un grafo dirigido $G = (V, E)$ entre un origen $s \in V$ y un destino $t \in V$, se realiza de la forma siguiente: inicialmente se cargan en memoria los nodos origen y destino, y un conjunto de elementos del grafo seleccionados en función de su grado de relación con los ya utilizados (en la implementación más sencilla se cargan los elementos adyacentes). Se dice entonces que ciertos nodos están completamente cargados, mientras que algunos de los nodos vecinos lo están parcialmente. A continuación el *walker* o visitante, específico del algoritmo, recorre los componentes del grafo utilizando un iterador, hasta que se alcanza el destino de la ruta. La mediación entre visitante e iterador la realiza el elemento *traversal*. En cada iteración, tan solo se generan y cargan en memoria el subconjunto de componentes del grafo que son estrictamente necesarios para el correcto funcionamiento del algoritmo, y el gestor de memoria tiene la oportunidad de liberar recursos eliminando elementos que ya no son necesarios para el algoritmo.

Con esta aproximación se consigue que, una vez alcanzado el destino de la ruta, el número de objetos del grafo cargados en memoria sea siempre menor que con otras librerías. Esto es debido a qué, según se comentó anteriormente, los algoritmos de cálculo de rutas, como el de Dijkstra o A*, no tienen por qué recorrer necesariamente todo el grafo para alcanzar su objetivo.

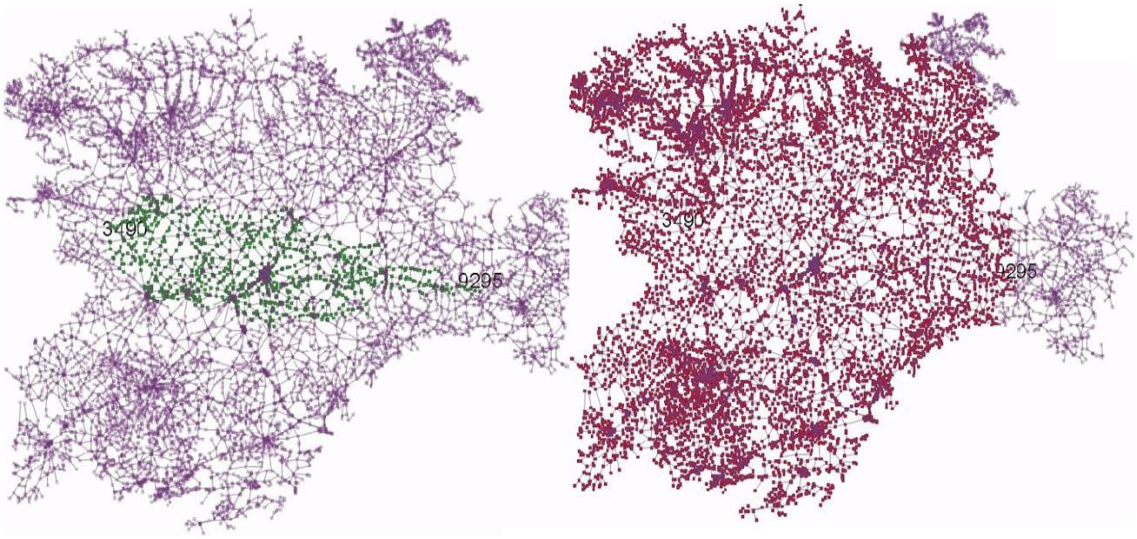

Figura 1: Componentes del grafo cargados (área oscurecida) por los algoritmos A (izda.) y Dijkstra (dcha.)*

La ganancia conseguida, medible como la proporción de objetos cargados en memoria frente al número total de objetos presentes en el grafo, al final de la ejecución del algoritmo, depende de las características del grafo en cuestión así como de la elección de los nodos involucrados en el cálculo de rutas. El diámetro de la red, coeficiente de agrupación o *clustering*, número de "triángulos", el algoritmo utilizado, la distancia en número de saltos entre los nodos origen y destino, son algunos factores que determinan en gran medida la ganancia conseguida.

En la Figura 1 se muestra un grafo de Castilla y León con 14.189 nodos. Sobre este grafo se han ejecutado los algoritmos de Dijkstra y A^* para el cálculo de la ruta de camino más corto entre los nodos origen y destino etiquetados como 3490 y 9295, respectivamente. Para el cálculo de esta ruta, el algoritmo de Dijkstra visita (y por tanto carga en memoria) 12.489 nodos (88% del total), mientras que el algoritmo A^* visita tan solo 1630 (12% del total). En este último caso se obtiene, por tanto, un ahorro muy significativo en el consumo de recursos de la máquina.

La carga dinámica de los elementos del grafo permite un menor consumo de recursos pero tiene como contrapartida un aumento en la latencia del algoritmo, pues aumenta el número de operaciones de Entrada/Salida por accesos al módulo de almacenamiento persistente, frente a un único acceso realizado según la estrategia de carga inicial en memoria de todo el grafo. Se trata, por tanto, de una solución de compromiso entre el consumo de recursos de memoria de la máquina y la latencia del algoritmo. Por otro lado, también se consigue un comportamiento más estable y escalable del sistema. Una adecuada política de carga y descarte de elementos puede hacer que el impacto en la latencia se desvanezca ante la repetición de cálculos espacialmente relacionados.

Considérese, sin embargo, un caso en el que el camino más corto se encuentre para un número reducido de iteraciones del algoritmo (supóngase para ello i.e. que en el grafo de la Figura 1 se hubiesen escogido como nodos origen y destino dos municipios cercanos y bien comunicados). La aproximación tradicional llevaría a cabo la tarea inicial de generación y carga en memoria de todo el grafo, con un tiempo asociado elevado. Por su parte, según la estrategia de carga dinámica implementada en la librería RouteEngine, se generarían y cargarían tan solo unos pocos nodos con lo que, en este caso, esta última implementación podría ser más ventajosa tanto en consumo de recursos como en tiempo de ejecución del algoritmo.

ARQUITECTURA DE LA LIBRERÍA IDELAB_ROUTE

La librería IDELabRoute está implementada íntegramente en Java heredando sus ventajas, como el funcionamiento multiplataforma en sistemas heterogéneos, sencillez, seguridad u orientación a objetos. Ha sido diseñada de forma que su arquitectura es modular, como se observa en la Figura 2. A continuación se describen los módulos principales de esta arquitectura.

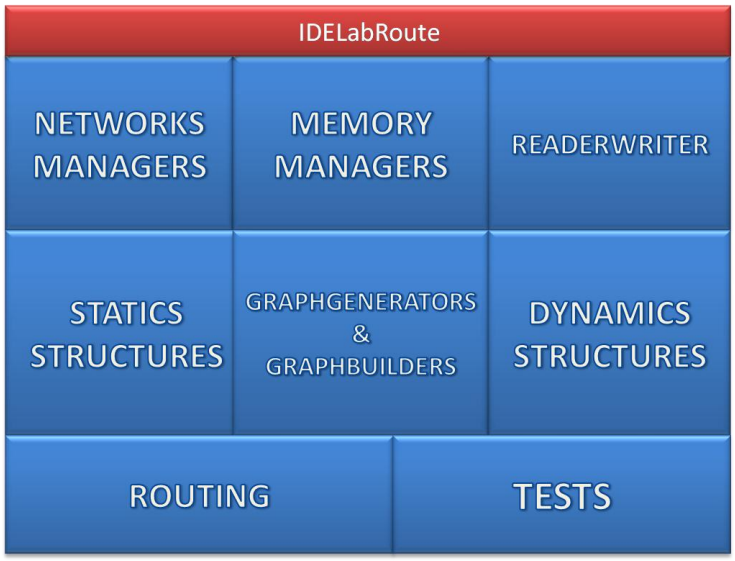

Figura 2: Módulos de la arquitectura de IDELabRoute

Gestión de memoria

En IDELabRoute, un grafo formado por un conjunto de nodos y arcos, se representa mediante una estructura "DynamicGraph" formada por objetos de tipo "DynamicNode" y "DynamicEdge", a los que nos referiremos como nodos y arcos dinámicos.

Tras la petición del cálculo de rutas se crea un grafo dinámico vacío, que guarda una relación directa con el gestor de memoria "GraphMemoryManager" utilizado. Actualmente se dispone de tres implementaciones de gestores de memoria, que se describen a continuación.

- **AllInMemoryManager:** Este gestor de memoria carga en memoria todos los elementos del grafo (nodos y arcos), y utiliza la propia zona de memoria como almacenamiento. Se mantiene en la librería por motivos de compatibilidad y para casos de grafos realmente pequeños.

- **AllInMemoryExternalSourceMemoryManager**: Esta implementación utiliza un almacenamiento persistente externo, controlado por un "GraphReaderWriter". Hereda las propiedades del anterior ya que también carga todos los elementos del grafo en memoria aunque, a diferencia del anterior, lo hace tras la primera petición de cualquier elemento que pertenezca al grafo. Carga toda la estructura del grafo a partir del almacén donde se encuentre éste. Todos los cambios de la red se guardan automáticamente en el almacén de persistencia sin necesitar intervención explícita del programa que utilice la librería.

- **BasicExternalSourceMemoryManager**: Este gestor precisa, al igual que el anterior, de un "GraphReaderWriter". A diferencia de éste, tras la petición de un elemento se carga completamente tan sólo el elemento solicitado, quedando sus elementos adyacentes, tanto nodos como arcos, parcialmente cargados en memoria. Esta implementación experimental se suministra para satisfacer los casos de prueba y como base para las implementaciones más optimizadas que ya se están implementando.

De los tres gestores de memoria comentados, el último es el que optimiza en mayor

medida el consumo de recursos de memoria, a costa de un aumento en el número de peticiones a realizar a la fuente de almacenamiento persistente.

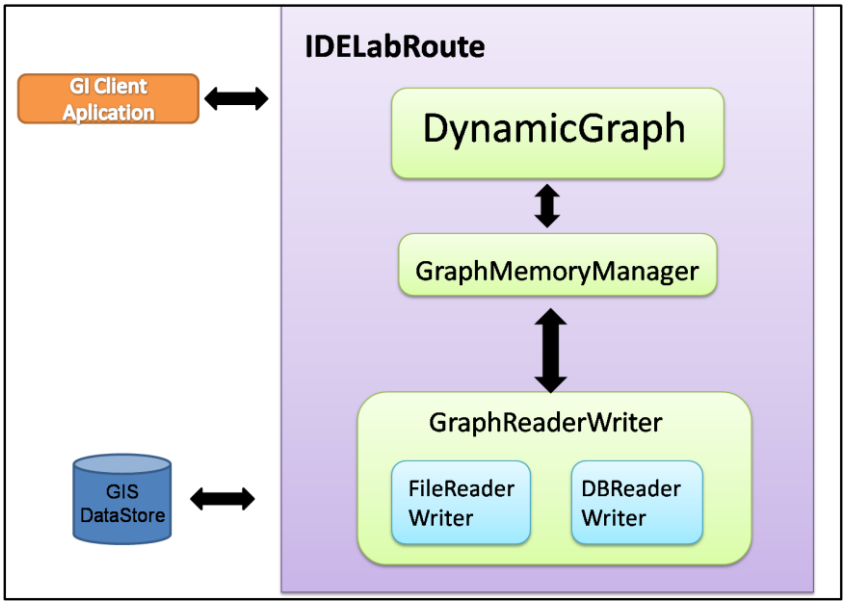

Figura 3: Componentes par la gestión de memoria y almacenamiento persistente

Módulo de almacenamiento persistente

La librería puede trabajar con distintas fuentes de almacenamiento persistente de los grafos. Actualmente ofrece la posibilidad de utilizar bases de datos o sistemas de ficheros, aunque dada la flexibilidad y modularidad de su arquitectura podrían añadirse nuevos mecanismos de persistencia de forma sencilla.

Como se puede observar en la Figura 3, el acceso a la fuente de almacenamiento persistente, tanto para lectura como para escritura de los grafos, se realiza a través de unos "*ReaderWriters*". Estos componentes ofrecen un nivel de abstracción que independiza a los gestores de memoria del tipo de persistencia utilizada.

Además los gestores de memoria basados en almacenes físicos de datos mantienen la integridad de los datos de forma automática y controlada.

Algoritmos de cálculo de rutas

La librería incluye implementaciones de las operaciones de análisis de redes más habituales:ShortestPath, *WithinCost* y *Traveling-Sales-Man* (TSP o problema del viajante).

Para cada uno de estas operaciones existe un buscador o caminante del camino en el grafo. Este buscador visita todos aquellos nodos que el iterador correspondiente al algoritmo en cuestión le va pasando. El buscador determinará cuando debe detenerse la búsqueda, tras la finalización del algoritmo, y proporciona el camino en caso que exista.

Para ello, el buscador se apoya en un *traversal* que se encarga de inicializar el iterador, en cuya inicialización se crean el mapa de nodos y la cola de prioridades. Posteriormente, al visitar los nodos del grafo se van creando por cada nodo un objeto del tipo de nodo adecuado al algoritmo, que son insertados en el mapa y en la cola anteriormente mencionada.

Plug-in* para *OpenJump

A modo de ejemplo de aplicación, se ha desarrollado uno de estos componentes para la interacción del cliente SIG de código libre *OpenJump* con la librería de grafos IDELabRoute. Este componente permite cargar los grafos almacenados tanto en base de datos como en sistema de ficheros en el cliente *OpenJump* para su visualización gráfica, así como la posibilidad de ejecutar cálculos de rutas sobre el grafo cargado. Permite seleccionar en el grafo los nodos involucrados en el cálculo de rutas, el tipo de algoritmo a utilizar y la posterior visualización del resultado de la ejecución del algoritmo (ver Figura 4).

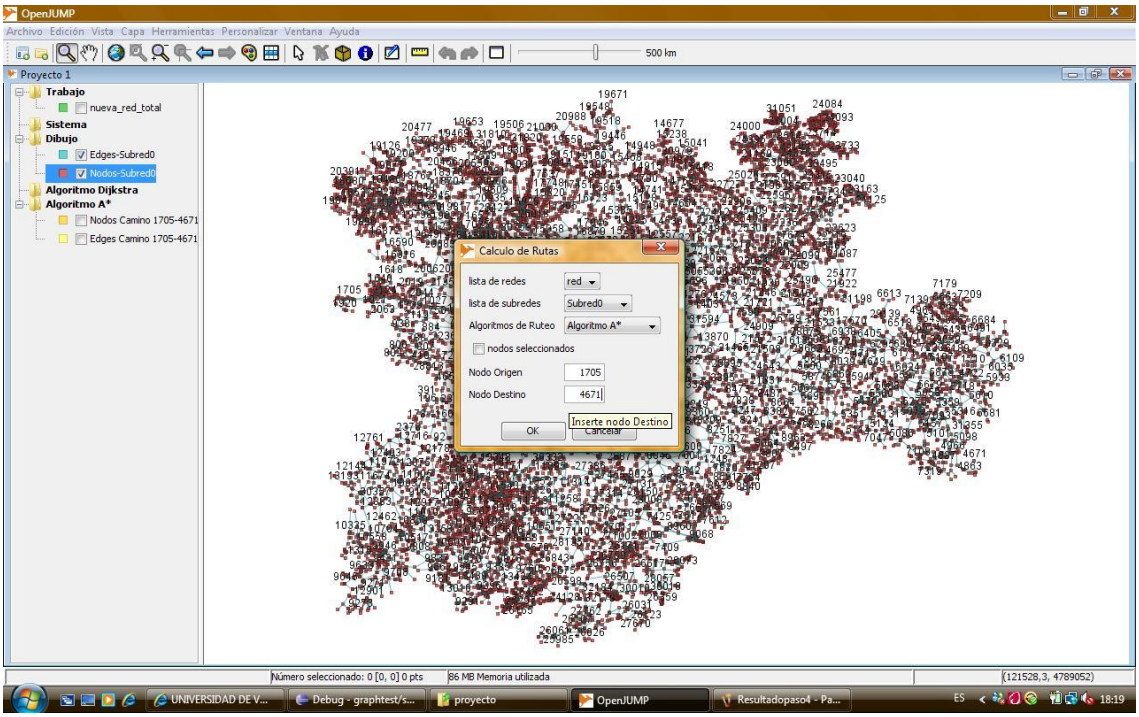

Figura 4: Plug-in de OpenJump para la librería IDELabRoute

CONCLUSIONES Y TRABAJO FUTURO

En este documento se ha presentado la librería *OpenSource* IDELabRoute para la gestión de grafos, desarrollada en el laboratorio IDELab de la Universidad de Valladolid. La librería está implementada en Java y su arquitectura modular y flexible permite la inclusión de gestores de memoria intercambiables, así como de distintas fuentes de almacenamiento persistente (i.e. bases de datos o sistemas de ficheros).

La librería IDELabRoute se está utilizando en el soporte de redes del proyecto LocalGIS, lo que ofrece una base de usuarios y de pruebas envidiable para el desarrollo futuro del proyecto.

Dispone de un innovador sistema para la gestión dinámica de la memoria, que permite realizar cálculos de rutas cargando en memoria solamente los componentes del grafo que son estrictamente necesarios para el funcionamiento del algoritmo. Esta aproximación reduce las necesidades en cuanto al consumo de recursos de la máquina y se perfila como una buena solución para el trabajo con grandes redes.

La librería ha sido diseñada para soportar procesado multihilo, lo cual, unido a su característica de gestión dinámica de memoria, la hace idónea para su uso en el lado

del servidor.

Se ha desarrollado un *plug-in* para la interacción con la librería de la aplicación de SIG de código libre *OpenJump*.

Actualmente es posible utilizar los servicios de esta librería mediante interfaces estándar como WPS y OpenLS, ambas del OGC. Se trata de una librería en continuo desarrollo por lo que en un futuro aumentará la funcionalidad ofrecida.

AGRADECIMIENTOS

El desarrollo de este trabajo ha sido posible gracias a la financiación por parte del Instituto Geográfico Nacional en el marco del Proyecto Conjunto al amparo del convenio de colaboración entre la dirección general del Instituto Geográfico Nacional y la Universidad de Valladolid en su edición de 2009.

REFERENCIAS

[1] P. Hart, N. Nilsson, y B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, 1968, págs. 100-107.

[2] J. Leskovec, “SNAP: Stanford Network Analysis Project” Available: http://snap.stanford.edu/.

[3] “Librería OpenSource pgRouting” Available: http://pgrouting.postlbs.org/.

[4] “GeoTools The Open Source Java GIS Toolkit — GeoTools v2.6 documentation” Available: http://www.geotools.org/.

[5] “Graphs - Codehaus” Available: http://docs.codehaus.org/display/GEOTDOC/Graphs.

[6] “IDELabRoute: Librería para la gestión de grafos escalables. | IDELab” Available: http://idelab.uva.es/proyectos/idelabroute.