Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Rodríguez-Puente, Rafael; Lazo-Cortés, Manuel S.

Modelo para la representación de redes y búsqueda de caminos óptimos en Sistemas de Información Geográfica

Ingeniare. Revista Chilena de Ingeniería, vol. 21, núm. 3, diciembre, 2013, pp. 394-407
Universidad de Tarapacá
Arica, Chile

Modelo para la representación de redes y búsqueda de caminos óptimos en Sistemas de Información Geográfica

Model for network representation and optimal path search in Geographic Information Systems

Rafael Rodríguez-Puente[1] Manuel S. Lazo-Cortés[1]

Recibido 2 de septiembre de 2011, aceptado 23 de abril de 2013
Received: September 2, 2011 Accepted: April 23, 2013

RESUMEN

Una de las funcionalidades presente en los sistemas de información geográfica es la búsqueda de caminos óptimos. En la actualidad, este tipo de funcionalidad se implementa sobre modelos que no garantizan escalabilidad y eficiencia cuando las redes son grandes.
En este artículo se propone un modelo de representación de redes en Sistemas de Información Geográfica basado en el concepto de grafos reducidos. Este modelo permite realizar búsquedas de caminos óptimos en redes grandes de forma eficiente y escalable. Una característica relevante del modelo propuesto es la posibilidad de realizar análisis a escala en la red.

Palabras clave: Representación de redes, sistemas de información geográfica, análisis de redes, búsqueda de caminos óptimos, reducción de grafos.

ABSTRACT

One of the features of Geographic Information Systems is related to optimal paths searches. This kind of functionality is implemented on models that do not guarantee scalability and efficiency when networks are large.
This article proposes a model for network representation in Geographic Information Systems. It is based on the concept of reduced graph. This model allows efficient and scalable optimal path searches in large networks. A relevant feature of the proposed model is the capability to perform scale analysis on the network.

Keywords: Network representation, Geographic Information Systems, network analysis, optimal path search, graph reduction.

INTRODUCCIÓN

Desde un punto de vista práctico un Sistema de Información Geográfica (SIG) es un sistema informático capaz de gestionar datos geográficos georreferenciados. Por georreferenciados se entiende que estos datos tienen asociadas coordenadas geográficas (longitud, latitud). También deben facilitar la relación de datos de diversa índole (densidad de población, información financiera, etc.) con datos geográficos.

Un SIG está formado por cuatro componentes: *hardware*, *software*, datos y recursos humanos [1].

Como parte de los datos los mapas tienen vital importancia. De forma intuitiva se puede decir que un mapa es un modelo que representa el "mundo

[1] Departamento de Técnicas de Programación. Facultad 3. Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 ½, Torrens, Boyeros, La Habana, Cuba. E-mail: rafaelrp@uci.cu; mlazo@uci.cu

real" y se almacena utilizando varios formatos, por ejemplo: Shape [2], TAB [3], entre otros. Por otra parte, existen proyectos que utilizan el modelo relacional extendido con soporte de tipos de datos espaciales (punto, línea, polígono, etc.) para almacenar los mapas; tal es el caso del proyecto OpenStreetMap [4].

Existen dos tipos de datos en el ámbito de los SIG: el vectorial y el raster. En el primer caso se utilizan puntos, líneas (definidas por una serie de puntos) y polígonos (delimitados por líneas) para representar los objetos geográficos. En el segundo caso los datos consisten en filas de celdas; a cada celda se pueden asociar datos de diversos tipos (medida, nombre, etc.) [5].

Para realizar análisis de redes se utilizan los SIG vectoriales, o sea, aquellos que usan el tipo de datos vectorial para representar la información geográfica; por lo que en el marco de este trabajo solo se hará referencia a este tipo de sistema.

Varios SIG cuentan con funcionalidades para el análisis de redes; entre estas se pueden mencionar las siguientes:

- ¿Cuál es el camino óptimo entre x e y?
- ¿Cuál es el camino de costo mínimo entre x e y según un determinado criterio?
- ¿Cómo llegar desde el lugar x al y pasando por los lugares $x_1, x_2, \ldots x_n$?

Para responder este tipo de preguntas existen varios modelos de representación de redes en un SIG. Entre ellos los más relevantes son: el modelo relacional extendido y sistemas de archivos propios de determinadas herramientas informáticas. Por otra parte, cuando se habla de análisis de redes en SIG, no se pueden dejar de mencionar los grandes proveedores de servicios como Google, Microsoft, etc.; a pesar de que los mismos no publican los detalles de los mecanismos de almacenamiento y análisis que utilizan para brindar este tipo de funcionalidad.

Entre los sistemas que utilizan el modelo relacional extendido para la búsqueda de caminos óptimos se puede mencionar a pgRouting [6]. Este software es una extensión del Sistema Gestor de Bases de Datos (SGBD) objeto-relacional PostgreSQL, lo que permite utilizar las potencialidades que brinda dicho SGBD; entre las cuales destaca el sistema de almacenamiento y el de consultas. A pesar de lo anterior, el sistema presenta las siguientes deficiencias:

- Utiliza un modelo de grafo demasiado simple y poco escalable [7].
- Se recomienda su uso cuando las redes modeladas en la cartografía son de tamaño pequeño o mediano [7].
- Presenta problemas en la búsqueda de caminos cuando el origen o el destino no coinciden con una intersección de calles [8].

Estas deficiencias se deben, en gran medida, a que la representación interna de las distintas redes presentes en un mapa, haciendo uso del modelo relacional extendido, es ineficiente [9]. Esto se sustenta en el hecho de que las operaciones definidas sobre los grafos (grado, adyacentes, etc.) no están definidas sobre el modelo relacional, por lo que su implementación haciendo uso de este modelo trae consigo la realización de operaciones adicionales; aumentando su complejidad. Además, en [10] se plantea que cuando el modelado de un dominio específico de la realidad (modelo lógico) responde a una estructura de grafo, utilizar una base de datos de grafo es la alternativa lógica para almacenar los datos. En [11] se muestran los resultados experimentales de la realización de recorridos en datos que, por su naturaleza, son representados haciendo uso de grafos dirigidos acíclicos. Los recorridos fueron realizados sobre el SGBD MySQL y el motor de persistencia de grafos Neo4j. Los resultados muestran de forma general que los recorridos realizados en Neo4j son más eficientes que en la base de datos relacional.

Por otra parte, existen varios sistemas que tienen una implementación propia para el almacenamiento de los mapas, destacándose entre ellos el líder ArcGIS con el formato Shape. Esta herramienta brinda la posibilidad de definir una jerarquía de calles, de forma tal que la búsqueda de caminos óptimos se realiza priorizando las calles de mayor nivel en la jerarquía cuando la red de viales es grande. La jerarquía se utiliza para disminuir el tiempo de respuesta al usuario, lo cual introduce un error en el camino obtenido según la descripción en la ayuda del propio sistema. Este sistema también brinda la posibilidad de realizar el análisis obviando la

jerarquía mencionada, lo que implica un menor rendimiento.

La interpretación del concepto "grande" que se hace en este trabajo es relativa y no se basa en un número particular de elementos presentes en una red. Este concepto puede depender de factores como características del *hardware* disponible, capacidad para la interpretación de características presentes en una red, entre otros.

El sistema gvSIG cuenta con un módulo de análisis de redes que provee diversas funcionalidades, facilitando la corrección de la topología, la creación de un grafo que representa una red de viales, así como la búsqueda de caminos óptimos. Este módulo de análisis de redes hace uso de los algoritmos clásicos de búsqueda de caminos óptimos, como el algoritmo de Dijkstra [12] y el A* [13]. El algoritmo de Dijkstra no es escalable respecto del tamaño del grafo y el A* no garantiza la obtención del óptimo en todos los casos.

El sistema Geographic Resources Analysis Support System (GRASS) es un SIG que se utiliza para manipulación de datos, procesamiento de imágenes, producción de gráficos y visualización de datos. GRASS utiliza la biblioteca Directed Graph Library [14] liberada bajo la licencia GPL para realizar análisis de redes. La idea original del proyecto se basa en el desarrollo de una biblioteca, que soporte el análisis sobre grafos de mediano tamaño en memoria de acceso aleatorio (RAM), haciendo uso de una estructura estática [15]; por lo que no es conveniente utilizarla cuando los grafos son grandes.

IDELabRoute es una biblioteca genérica para realizar análisis de redes con gestión dinámica de memoria, que surge porque en la práctica varios problemas necesitan tratar con redes de grandes dimensiones. Esta biblioteca permite realizar análisis de redes utilizando diversas fuentes de datos. De forma general, en IDELabRoute se propone hacer un uso racional de la RAM (mayor escalabilidad) a cambio de un decremento de la eficiencia en la búsqueda de caminos óptimos. Debido a esta razón no se considera conveniente usar este resultado cuando los grafos sobre los que se realizan análisis son grandes y es necesario mantener la eficiencia en la búsqueda de caminos óptimos.

La teoría de grafos provee una representación adecuada de una red, así como conceptos y algoritmos que permiten estudiar las propiedades de las mismas [16]. Por otra parte, en [17] se afirma que una red puede ser representada con la misma estructura de datos que es utilizada para la representación de grafos, por lo que son aplicables los algoritmos definidos para estos últimos. También se define una red como un conjunto de nodos y un conjunto de relaciones entre ellos [18], definición que está acorde con el concepto de grafo.

Luego del estudio de los modelos más utilizados para realizar análisis de rutas en SIG y teniendo en cuenta las definiciones de red y de grafo, se puede afirmar que es conveniente estudiar una red haciendo uso del concepto de grafo.

En el ámbito del uso de los grafos en la búsqueda de caminos óptimos, se han realizado varios trabajos para disminuir el tiempo de respuesta ante este tipo de petición en redes grandes haciendo uso de algoritmos heurísticos. Se han diseñado varias heurísticas y diversos algoritmos para resolver esta problemática [19-22]. Sin embargo, con la introducción de las heurísticas se introduce un error en el resultado; por lo que se puede afirmar que estos algoritmos no garantizan la obtención del camino óptimo en todos los casos.

Teniendo en cuenta el análisis realizado se plantea como objetivo de esta investigación: diseñar un modelo basado en grafos reducidos para representar redes en SIG, como punto de partida para la realización de búsquedas de caminos óptimos de forma eficiente y escalable.

El artículo se organiza de la siguiente forma: primero se presenta el modelo propuesto; luego se describen los componentes del mismo, así como las relaciones entre estos; después se muestran resultados experimentales y por último se presentan las conclusiones.

MODELO PARA LA REPRESENTACIÓN DE REDES EN SISTEMAS DE INFORMACIÓN GEOGRÁFICA

Existe una amplia variedad de definiciones de modelo. A pesar de sus diferencias, todas las definiciones tienen en común que un modelo es una

representación teórica o gráfica de un fenómeno; en el mismo se trata de especificar los elementos fundamentales que caracterizan el fenómeno así como las relaciones entre estos.

En el marco de este trabajo se utilizará la definición de François E. Cellier, que expone que un modelo (M) para un sistema (S) y un experimento (E) es cualquier cosa a la cual se le puede aplicar E para responder preguntas sobre S [23].

En la Figura 1 se muestra un esquema con los componentes del modelo propuesto para la representación de redes viales. A continuación, se expone en detalle cada uno de los componentes de este modelo, así como las relaciones existentes entre los mismos.

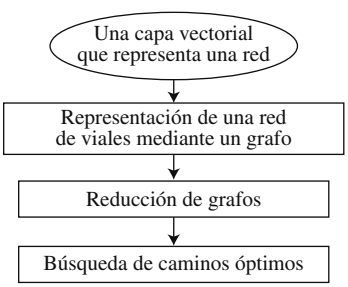

Figura 1. Modelo de representación de redes de viales.

Representación de una red mediante un grafo

Para representar una red mediante un grafo se parte de una capa vectorial que represente dicha red. En primer lugar, se propone el uso de un algoritmo que obtenga las intersecciones entre cada par de líneas de la red, para ello se puede utilizar el algoritmo FINDINTERSECTIONS presentado en [24]. Esta variante de solución tiene complejidad temporal O(n log n + I log n), donde n es la cantidad de segmentos e I la cantidad de intersecciones entre todos los segmentos.

Este algoritmo tiene como salida el conjunto de intersecciones entre todas las líneas que recibe como parámetro, así como las líneas relacionadas con cada intersección. A partir de estos datos se puede crear un grafo haciendo uso del Algoritmo 1.

Algoritmo 1: *ObtenerGrafo*
Entrada: Lista de líneas (*l_lineas*) y una lista de puntos por cada segmento (*l_intersec*) que representa las intersecciones que contiene cada uno de estos con todos los demás
Salida: Un grafo

1. $G = (\{\},\{\})$
2. ***Para todo*** *linea* ∈ *l_lineas* **hacer**
3. *aristas=DeterminarAristas(linea, l_intersec[c])*
4. *adicionarAristas(G, aristas)*
5. ***Fin Para***
6. ***Retornar*** *G*

Para la determinación de las aristas (paso 3 del Algoritmo 1) se utiliza el Algoritmo 2. En el mismo se debe tener en cuenta que una arista está conformada por cuatro valores: vértice de origen, vértice de destino, costo de la arista (puede ser un vector de valores que represente distintos tipos de costo) y su geometría. Como esta arista modela una parte de una red existente en el mundo real, es importante tener acceso a su geometría. Esto puede ser útil si en un futuro se desean obtener imágenes relacionadas con análisis realizados sobre el grafo; por ejemplo, para dibujar en un mapa el camino mínimo buscado sobre una determinada red. La función *Costo* permitirá obtener el costo asociado a una arista en dependencia de la información disponible de la red.

Algoritmo 2: *DeterminarAristas*
Entrada: Una línea (*linea*) que representa una porción de la red y una lista de puntos (*l_puntos*)
Salida: Un conjunto de aristas que representan la línea de entrada.

1. *aristas* = { }
2. ***Si*** *l_puntos*[0] ≠ *linea* [0] ***entonces***
   Insertar(*l_puntos*, 0, *linea*[0])
3. ***Si***
   l_puntos [*Lon*(*l_puntos*)] ≠
   linea[*Cant_Puntos*(*linea*)] ***entonces***
   Adicionar(*l_puntos,linea*[*CantPuntos*(*linea*)])
4. Calcular el costo desde el primer punto de la línea hasta cada punto de la lista de puntos
5. Ordenar la lista de puntos según la distancia calculada
6. ***Para*** *i* = 0 ***hasta*** *i* < *Lon*(*l_puntos*)–1 **hacer**
7. *a = ExtGeometria(linea, l_puntos*[*i*], *l_puntos*[*i*+1])
8. *Adicionar*(*aristas*, *l_puntos*[*i*], *l_puntos*[*i*+1], *Costo*(*a*), *a*)
9. ***Fin Para***
10. ***Retornar*** *aristas*

ALGORITMO DE REDUCCIÓN DE GRAFOS

En este epígrafe se presentan definiciones, notaciones y algoritmos relacionados con el proceso de reducción de grafos propuesto como parte del modelo.

Una regla de reescritura de grafos es un formalismo utilizado para transformar un grafo en otro siguiendo determinados principios. En el marco de este trabajo se utilizan las reglas de reescritura para garantizar que el proceso de reducción sea reversible y por tanto, que no exista pérdida de información.

A continuación se introducen las definiciones de regla de reescritura de grafos, grafo reducido, grafo reducido a partir de un grafo y vértice interior y exterior; las cuales pertenecen a los autores del presente trabajo.

Definición 1. Una regla de reescritura de grafos sobre un grafo $G = (V, E, f_c)$ es un cuádruplo de la forma $(G_i, G_j, \psi_{in}, \psi_{out})$, donde:

- $G_i = (\{v_i\},\{\})$ es un grafo, donde $v_i \in$ V.
- $G_j = (V_j, E_j)$ es un grafo.
- ψ_{in} y ψ_{out} son dos conjuntos, cuyos elementos son cuádruplos de la forma (v_m, c_1, c_2, v_n), donde $c_1, c_2 \in \mathrm{R}^+$; $v_m \in V_j$; $v_n \in (V–\{v_i\})$.

Para que un cuádruplo (v_m, c_1, c_2, v_n) pertenezca a ψ_{in} se debe cumplir que exista la arista (v_n, v_i) en el grafo G (sobre el cual se define la regla de reescritura); además, el costo de dicha arista debe ser c_1. Luego de aplicar la regla de reescritura, se obtiene el grafo $G_1 = (V_1, E_1, fc_1)$ y se cumple que la arista (v_n, v_m) pertenece al nuevo grafo, y el costo de la misma es c_2. Análogamente se define ψ_{out}, con la única diferencia de la orientación de las aristas.

Definición 2. Un grafo reducido es un cuádruplo $G_r = (V_r, E_r, f, R)$, donde:

- V_r es un conjunto de vértices.
- E_r es un multiconjunto de aristas, donde cada arista es un par ordenado de vértices (opcionalmente puede tener una etiqueta que la identifique y una lista de atributos).
- f: $V_r \times V_r \times V_r \rightarrow \mathbb{R}^+$ es una función que para cada trío de vértices (v_i, v_j, v_k) retorna el costo de ir desde v_i hasta v_k a través de v_j, siendo v_k adyacente a v_j y v_j adyacente a v_i. De forma abreviada se hará referencia a esta función como la función de cruce.
- R es un conjunto de reglas de reescritura sobre (V_r, E_r). Si el conjunto R de un grafo reducido es vacío, la función de cruce puede ser calculada a partir de la función de costo del grafo ponderado como $f(v_i, v_j, v_k) = fc(v_i, v_j) + fc(v_j, v_k)$, siendo v_k adyacente a v_j y v_j adyacente a v_i.

Definición 3. Sean G y G_r dos grafos y $R = \{r_1, r_2, \ldots, r_n\}$ un conjunto de reglas de reescritura de grafos; se puede afirmar que G_r es un grafo reducido a partir de G, si al aplicar las reglas de reescritura especificadas en R al grafo G_r se obtiene el grafo G.

Definición 4. Sea un grafo $G = (V, E)$ y una partición P sobre V; un vértice $v_i \in V$ es interior si $\forall v_j \in V$, tal que v_i y v_j son adyacentes, se cumple que v_i y v_j pertenecen a la misma clase de P. Un vértice v_i es exterior si $\exists v \in V$, tal que $((v, v_i) \in$ E o $(v_i, v) \in$ E) y v_i y v_j no pertenecen a la misma clase de P.

Cuando se menciona el grafo original se hace referencia al grafo que es entrada de la iteración del algoritmo que se esté ejecutando, que puede ser un grafo distinto al existente antes de ejecutar el algoritmo de reducción por primera vez.

La propuesta tiene como entrada un grafo y una partición sobre los vértices del grafo de entrada. Sin embargo, puede ser necesario refinar esta partición; ya que para obtener un camino en el grafo reducido, de igual costo que el que se obtiene con el algoritmo de Dijkstra en el grafo original, es necesario que en dicho grafo reducido no existan pares de vértices que sean adyacentes y a su vez reducidos. Debido a esto, el primer paso del algoritmo de reducción consiste en refinar la partición que es entrada de dicho algoritmo (haciendo uso del Algoritmo 4 teniendo en cuenta la definición de vértice interior y exterior). Para ello se sigue la siguiente estrategia: dos vértices pertenecen a una clase de la nueva partición si y solo si están en la misma clase de la partición de entrada y son vértices interiores.

En la Figura 2 se ilustra con un ejemplo en qué consiste lo explicado anteriormente. En la misma se puede apreciar la modificación realizada a la partición inicial haciendo uso de la definición de vértice interior, con el objetivo de garantizar la obtención de caminos óptimos de igual costo en el grafo sin reducir.

Por cada clase de la partición refinada se crea un vértice en el grafo reducido. Si la cardinalidad de la clase es mayor que uno, el vértice que se crea se considera reducido; en otro caso se considera no reducido. Además, se propone el uso de una función (en este caso *ObtenerNombre*) que a partir de una clase de la partición, devuelva un identificador que

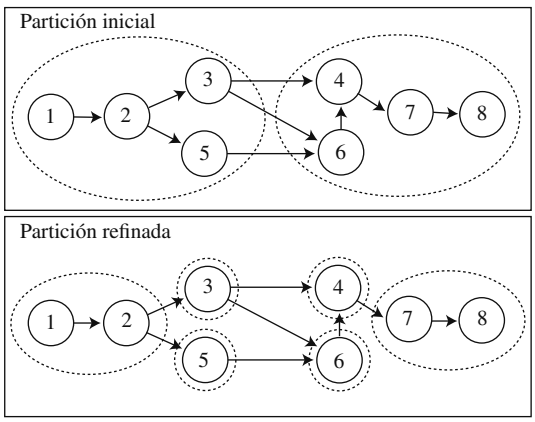

Figura 2. Ejemplo de refinamiento de la partición. En el grafo de la parte superior se puede apreciar que los vértices 3, 4, 5 y 6 son exteriores; por cada uno de dichos vértices se creará una clase en la nueva partición.

será asociado a dicho vértice. En caso de que la clase tenga cardinalidad mayor que uno, la función retornará el valor del atributo utilizado para crear la clase; si la cardinalidad es uno, se retorna el identificador del vértice correspondiente a la clase en el grafo original.

Las aristas del grafo reducido se crean a partir de las clases de la partición refinada y del grafo original. Para ello se analiza cada par de clases de la partición, y si existe una arista entre dos vértices que pertenecen a clases distintas, se adiciona al grafo reducido. A medida que se van adicionando las aristas al grafo reducido se debe ir actualizando la función de costo.

Para construir las reglas de reescritura se parte de la Definición 1. Se crea una regla por cada vértice reducido. Este es un paso esencial en el algoritmo de reducción, es el que permite que no haya pérdida de información en la reducción del grafo y además garantiza que se pueda obtener el grafo original a partir del grafo reducido.

En primer lugar, para cada vértice reducido v_i se crea el grafo G_i; el mismo está formado por un vértice que representa la clase correspondiente de la partición refinada. Luego se crea el grafo G_j, formado por todos los vértices que pertenecen a la clase en cuestión y las aristas existentes entre dichos vértices en el grafo original. Luego, por cada arista del grafo original que incide en vértices que pertenecen a la clase de la partición se adiciona un cuádruplo al conjunto ψ_{in}, y por cada arista del grafo original que sale de un vértice de la clase hacia un vértice que no pertenece a dicha clase se adiciona un cuádruplo en ψ_{out}. De esta forma se garantiza que cuando se apliquen las reglas de reescritura al grafo reducido, se obtiene un nuevo grafo con los mismos vértices y aristas del grafo original; por lo que se garantiza que el proceso de reducción es reversible y no presenta pérdida de información.

El último paso consiste en calcular la función de cruce del grafo reducido. Esta función almacena para cada trío de vértices (v_i, v_j, v_k), con v_k adyacente a v_j y v_j adyacente a v_i, el costo de ir desde v_i hasta v_k a través v_j. La función de cruce también puede ser vista como la forma de almacenar el costo de pasar por un vértice reducido. Para calcular esta función se crea un grafo auxiliar por cada vértice reducido a partir del grafo G_j de la regla de reescritura correspondiente. El objetivo de este grafo auxiliar es tener un grafo sobre el que se pueda realizar modificaciones para utilizarlo en el cálculo de la función antes mencionada.

Luego de adicionar los vértices y aristas del grafo G_j al grafo auxiliar, se adicionan los vértices adyacentes a cada vértice que pertenezca al grafo G_j. Estos vértices adyacentes se almacenan en la variable *verticesAdyacentes* para ser utilizados posteriormente.

Sobre el grafo auxiliar creado se aplica el Algoritmo 13 (MDijkstra), tomando como vértice de origen (v_o) cada uno de los vértices almacenados en la variable *verticesAdyacentes*; cada vez que se invoca este algoritmo se debe ir actualizando la función de cruce.

La función calculada almacena, además del costo de ir de un vértice a otro pasando por uno reducido, el propio camino; o sea, la secuencia de vértices a seguir de forma tal que con este valor precalculado se reduce el tiempo necesario para mostrar el camino óptimo.

Para calcular el camino antes mencionado se utiliza la función *CamiNº* La implementación de esta función parte del hecho que *Pr* es un vector que representa los predecesores de cada vértice en el

camino óptimo desde el vértice de origen (v_o) hasta el resto de los vértices del grafo. El camino desde v_o hasta v_d se obtiene recorriendo el vector Pr; en este recorrido se calcula cuál es el predecesor v_k del vértice v_d ($v_k = Pr[v_d]$); después se calcula el predecesor de v_k. Este proceso se repite hasta que el predecesor de algún vértice del camino coincida con el vértice de origen. La concatenación de los vértices, en orden inverso al encontrado, es el camino óptimo desde v_o hasta v_d.

Finalmente, se crea un grafo reducido utilizando los conjuntos de vértices, aristas, reglas de reescritura y la función de cruce.

El Algoritmo 3 muestra el pseudo-código del algoritmo de reducción. Además se presentan nuevos algoritmos para la implementación de dichos pasos.

Algoritmo 3: *ReducirGrafo*
Entrada: Un grafo ponderado reducido $G = (V, E, f, R)$, donde R es un conjunto de reglas de reescritura posiblemente vacío. Una partición P en V.
Salida: Un grafo ponderado reducido

1. P = *RefinarParticion*(P, G)
2. V_r = *ConstruirVerticesReducidos*(P)
3. E_r = *ConstruirAristas*(P, G)
4. R_r = *ConstruirRr*(P, G)
5. $f_r = \phi$
6. ***Para todo*** $A_i \in P, |A_i|>1$
7. *Calcularf*(G, $R_r[A_i]$. G_i, $R_r[A_i]$. G_j, f_r) {Calcular el costo de pasar por el vértice reducido correspondiente a la clase A_i de P (ver Algoritmo 8). La variable f_r es un parámetro pasado por dirección.}
8. ***Fin Para***
9. Crear el grafo reducido $G_r = (V_r, E_r, f_r, R_r)$
10. ***Retornar*** G_r

Algoritmo 4: *RefinarParticion*
Entrada: Una partición P y un grafo $G = (V, E)$
Salida: Una partición Pa

1. $Pa = \{\}$
2. **Para todo** $v_i \in V$ **hacer**
3. $Int_v = \{\}$
4. **Para todo** $v_j \in V$, $i \neq j$ **hacer**
5. **Si** v_i y v_j pertenecen a la misma clase de P **y** *EsInterior*(G, P, v_j) **entonces** *Adicionar*(Int_v, v_j)
6. **Sino Si** v_i y v_j pertenecen a la misma clase de P **entonces** *Adicionar*(Pa,{v_j})
7. **Fin Si**
8. **Fin Para**
9. **Si** $Int_v \neq \phi$ **entonces**
10. *Adicionar*(Pa, Int_v)
11. **Fin Si**
12. **Fin Para**
13. **Retornar** Pa

Algoritmo 5: *ConstruirVerticesReducidos*
Entrada: Una partición $P = \{A_1, A_2, \ldots, A_s\}$.
Salida: El conjunto de vértices V_r formado por s vértices.

1. $V_r = \{\}$
2. **Para todo** $A_i \in P$
3. *Adicionar*(V_r, *ObtenerNombre*(A_i))
4. **Fin Para**
5. **Retornar** V_r

Algoritmo 6: *ConstruirAristas*
Entrada: Una partición $P = \{A_1, A_2, \ldots, A_s\}$ y un grafo reducido $G = (V, E, f, R)$
Salida: El conjunto de aristas E_r

1. **Para todo** $e_m \in A_i$, $e_n \in A_j$, $i \neq j$; $i, j = 1..s$ **hacer**
2. v_i = *ObtenerNombre*(A_i)
3. v_j = *ObtenerNombre*(A_j)
4. **Si** $(e_m, e_n) \in E$ **entonces**
5. *Adicionar*(E_r, v_i, v_j)
6. $f(v_i, v_i, v_j) = f(e_m, e_m, e_n)$
7. **Fin Si**
8. **Fin Para**
9. **Retornar** E_r

Algoritmo 7: *ConstruirRr*
Entrada: Una partición $P = (A_1, A_2, \ldots, A_s)$ y un grafo reducido G = (V, E, f, R)
Salida: El conjunto de reglas de reescritura R

1. $R = \{\}$
2. **Para todo** $A_i \in P$, $i = 1..s$, tal que $|A_i| >1$ **hacer**
3. $G_i = (\{$*ObtenerNombre*(A_i)$\}$, $\{\}$), $V_j = A_i$, $E_j = \{\}$
4. **Para todo** v_m, $v_n \in V_j$, $m \neq n$ **hacer**
5. **Si** $(v_m, v_n) \in E$ **entonces**
6. *Adicionar*(E_j, *ObtenerArista*(G, v_m, v_n))
7. $f_j(v_m, v_m, v_n) = f(v_m, v_m, v_n)$
8. **Fin Si**
9. **Si** v_n es reducido en G **entonces**
10. **Para todo** $v_k \in$ *Adyacentes*(G, v_n) **hacer**
11. $f_j(v_m, v_n, v_k) = f(v_m, v_n, v_k)$
12. **Fin Para**
13. **Fin Si**
14. **Fin Para**
15. R_j = *ObtenerR*(V_j, R) {Obtiene las reglas de reescritura asociadas a V_j.}
16. $G_j = (V_j, E_j, f_j, R_j)$
17. **Para todo** $v_k \in V_j$ **hacer**
18. *ady* = *AristasQueEntran*(v_k, G, $[v_k]$–V_j)
19. **Para todo** $v \in$ *ady* **hacer**
20. *Adicionar*(ψ_{in}, $(v, f(v, v, v_k), f(v, v, v_k), v_k)$)
21. **Fin Para**
22. *ady* = *AristasQueSalen*(v_k, G, $[v_k]$–V_j)
23. **Para todo** $v \in$ *ady* **hacer**
24. *Adicionar*(ψ_{out},$(v, f(v_k, v_k, v), f(v_k, v_k, v), v_k)$)
25. **Fin Para**

26. **Fin Para**
27. Adicionar(R, (G_i, G_j, ψ_{in}, ψ_{out}))
28. **Fin Para**
29. **Retornar** R

Algoritmo 8: *Calcularf*
Entrada: Un grafo ponderado y reducido $G = (V, E, f_r, R)$, los subgrafos G_i y G_j de la regla de reescritura asociada a una clase A_i de P y la función de cruce f (este último parámetro es pasado por dirección)
Salida: La función de cruce f para el vértice reducido correspondiente

1. $G_{aux} = (A_{aux}, E_{aux}, f_{aux}) = G_j = (A_i, E_i, f_c)$
2. *verticesAdyacentes* = { }
3. **Para todo** v_i, v_j, $v_i \in A_{aux}$, $v_j \in Adyacentes(v_i, G)$ **hacer**
4. **Si** $v_j \notin A_i$ **entonces**
5. *AdicionarArista*(G_{aux}, *ObtenerArista*(v_i, v_j, G))
6. $f_{aux}(v_i, v_i, v_j) = f_r(v_i, v_i, v_j)$
7. $f_{aux}(v_j, v_j, v_i) = f_r(v_j, v_j, v_i)$
8. *Adicionar*(*verticesAdyacentes*, v_j)
9. **Fin Si**
10. **Fin Para**
11. **Para todo** $v_o \in$ *verticesAdyacentes* **hacer**
12. *MDijkstra*(v_o, G_{aux}) (ver Algoritmo 13)
13. **Para todo** $v_d \in$ *verticesAdyacentes*, $v_o \neq v_d$ **hacer**
14. $f(v_o, v_i, v_d) = (D[v_d], camino(v_o, v_d, P_r))$
15. **Fin Para**
16. **Fin Para**

El algoritmo de reducción de grafos propuesto tiene particular importancia en el trabajo con mapas debido a que los mismos siempre se muestran al usuario a una determinada escala; contar con un grafo reducido con el algoritmo anterior permitiría realizar análisis de redes en función de una determinada escala.

Por ejemplo, se puede relacionar cada iteración del algoritmo con una escala del mapa. En otras palabras, si se reduce agrupando todos los vértices que están en una misma provincia, el grafo reducido resultante representa la capa vectorial que pertenece al mapa a escala de provincia. Esto trae como consecuencia simplicidad en el análisis que se realice, ya que a una escala determinada pudieran perder importancia algunos objetos geográficos debido a que a esa escala, los mismos no son visibles o no son de interés para el análisis que se realiza.

Además, el Algoritmo 3 permite reducir un grafo sin que exista pérdida de información en el proceso, lo que contribuye a realizar análisis sobre el grafo reducido y obtener los mismos resultados que se obtienen en el grafo sin reducir.

Calculando la complejidad temporal de cada paso de los algoritmos presentados en este epígrafe se obtiene que la complejidad computacional del algoritmo de reducción propuesto: $O(n^2\sqrt{n}\ \log(n-\sqrt{n}))$, siendo n la cantidad de vértices del grafo original. Es válido aclarar que esta complejidad no afecta la búsqueda de caminos óptimos, ya que el proceso de reducción es un preprocesamiento de los datos, o sea, el grafo se reduce solamente una vez y luego de la reducción, se pueden realizar varias búsquedas de caminos óptimos.

BÚSQUEDA DE CAMINOS ÓPTIMOS

El análisis de redes en el presente modelo se realiza sobre un grafo reducido según el Algoritmo 3. En particular se describe cómo se realiza la búsqueda de caminos óptimos, para ello se debe tener en cuenta que se puede realizar el análisis según dos enfoques:

1. A escala, es decir, donde no se tengan en cuenta todos los objetos que existen en la capa vectorial utilizada como entrada del modelo; por ejemplo, cuál es el camino óptimo entre el municipio *x* y el *y*. Este tipo de pregunta debe estar acorde con la reducción realizada; para ello, se debe haber reducido el grafo según una partición que agrupe en la misma clase todos los vértices que pertenecen a un mismo municipio.
2. Teniendo en cuenta todos los objetos presentes en la capa vectorial que forma parte de la entrada del algoritmo de reducción.

La diferencia principal entre estos dos enfoques radica en la forma de obtener el grafo sobre el cual se va a realizar el análisis.

En el primer enfoque se debe utilizar un grafo en el que todos sus vértices estén a un mismo nivel de reducción; es decir, todos hayan sido reducidos utilizando la misma partición y por tanto están a la misma escala.

En el segundo enfoque deben existir vértices con distintos niveles de reducción. Esto se debe a que cuando se hace una búsqueda de camino óptimo en grafos grandes, el mecanismo que provee el modelo para lograr eficiencia es tener varios vértices reducidos; los vértices de origen y de destino no pueden estar reducidos en aras de obtener resultados exactos.

Para hacer búsqueda de caminos óptimos según los dos enfoques planteados se propone el uso de la estrategia que se describe a continuación.

En primer lugar, se debe modificar el grafo reducido en dependencia del enfoque que se desee seguir. Si el enfoque es el primero, es necesario tener todos los vértices del grafo a un mismo nivel, según la escala a la que se desee realizar el análisis. Para obtener este grafo se puede recorrer la lista de vértices y expandir cada vértice reducido haciendo uso del Algoritmo 11, de esta forma se obtendría el nivel de reducción anterior del grafo; si es más de un nivel bastaría con volver a aplicar lo descrito anteriormente.

Si el enfoque es el segundo habría que expandir vértices reducidos hasta que los vértices de origen (v_o) y de destino (v_d) estén en el grafo.

Se deben realizar dos tipos de expansiones, en uno de ellos se expande un vértice reducido (aplicando la regla de reescritura correspondiente) y en el otro se deben realizar expansiones de vértices reducidos hasta que aparezca en el grafo un vértice dado.

En el primer caso (Algoritmo 11) se parte de un vértice del grafo reducido y se aplica la regla de reescritura correspondiente. Para ello, se debe adicionar el grafo G_j al grafo reducido, conectar los vértices de G_j con los vértices del nuevo grafo, según ψ_{in} y ψ_{out} y, por último, eliminar el vértice reducido que se está expandiendo.

En el segundo caso (Algoritmo 12) se tiene en cuenta el carácter jerárquico de las particiones; por ejemplo, si el grafo se redujo por municipio y luego por provincia, para obtener a través de expansiones de vértices reducidos un vértice que se encuentra situado geográficamente en un determinado municipio, se debe expandir en primer lugar el vértice que se corresponde con la provincia y luego el que se corresponde con el municipio.

Para obtener el vértice reducido que contiene a un vértice dado (paso 3 del Algoritmo 12), se realiza un recorrido sobre el conjunto de vértices del grafo reducido y se comprueba con cuál vértice reducido está relacionado el vértice pasado como parámetro. Para esto se hace uso de las particiones utilizadas como entrada del Algoritmo 3 para reducir el grafo.

Cuando se encuentra el vértice reducido, se expande aplicando la regla de reescritura correspondiente. El pseudo-código correspondiente se muestra en el Algoritmo 10.

Una vez que se tiene un grafo reducido en el cual los vértices de origen y destino son vértices no reducidos, se aplica el Algoritmo 13 (MDijkstra). El mismo está basado en el algoritmo de Dijkstra. El algoritmo MDijkstra es uno de los principales aportes del presente trabajo, ya que garantiza la obtención de un camino óptimo en el grafo reducido, de igual costo que el camino óptimo encontrado por el algoritmo de Dijkstra en el grafo original (sin reducir). La modificación realizada sigue el siguiente principio: cada vez que se actualiza la distancia desde el pivote (ver línea 7, el vértice w_n se denomina pivote) hasta un vértice adyacente que sea reducido (línea 13), se actualizan las distancias desde el pivote hasta todos los vértices adyacentes a dicho vértice reducido, haciendo uso de la función de cruce. Este algoritmo retorna un vector D con las distancias mínimas desde el vértice de origen al resto de los vértices, así como el vector P que contiene los predecesores de cada vértice en el camino óptimo desde el vértice de origen.

La función de cruce, calculada durante el proceso de reducción, es necesaria, ya que para obtener un camino óptimo se requiere conocer el costo de pasar por un vértice reducido. Este valor no queda asociado directamente a las aristas porque, en el caso de los grafos reducidos, el costo de una arista depende del camino en el cual se encuentren los vértices que conforman la misma.

Como se puede apreciar en la Figura 3, el efecto que tiene el costo de ir desde el vértice 2 al 13 en el cálculo de un determinado camino, depende si se llega al vértice 2 desde el vértice 7 o desde el 6. Cada vértice reducido representa un subgrafo del grafo original por lo que el costo de recorrer dicho subgrafo dependerá del vértice desde el cual se llega al mismo.

Para obtener el camino óptimo a partir del vector P de predecesores se calcula el antecesor del vértice v_d en el camino desde v_o, a este vértice le llamaremos $v_{aux} = P[v_d]$. Luego, el antecesor del vértice encontrado (v_{aux}) se calcula de la misma forma $v_{aux} = P[v_{aux}]$. Este proceso se repite hasta que v_{aux} sea igual a v_o. El camino que se retorna es la concatenación de todos los vértices encontrados, en orden inverso.

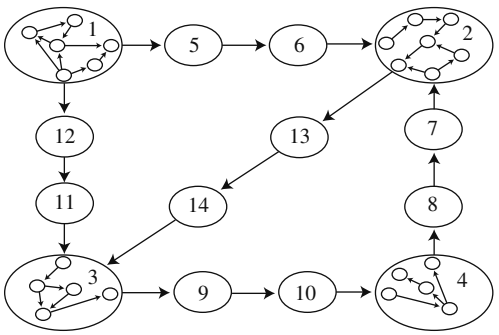

Figura 3. Ejemplo de grafo reducido.

La complejidad temporal del algoritmo *BuscarCaminoOptimo* es $O(max[n_{vr} * a_i, n_{vr} * \log n_{vr}])$ siendo n_{vr} la cantidad de vértices del grafo reducido y $a_i = max[|A_i|, A_i \in Pa]$. Teniendo en cuenta que en un grafo suficientemente grande $a_i > n_{vr}$, entonces $a_i > \log n_{vr}$. Finalmente la complejidad sería $O(n_{vr} * a_i)$.

En un SIG, los vértices de origen y de destino para una petición de búsqueda de camino óptimo se seleccionan generalmente haciendo uso del mapa; es decir, un usuario selecciona estos puntos haciendo clic en el mapa que muestra el sistema. En este sentido, se asume que cuando un usuario selecciona un vértice de origen o destino, el SIG realiza una expansión de un vértice reducido teniendo en cuenta la porción del mapa que se está mostrando y el punto seleccionado.

Si un sistema para realizar búsquedas de caminos óptimos es implementado de esta forma, el tiempo necesitado para expandir un vértice reducido sería irrelevante para la búsqueda de caminos óptimos; considerando que la expansión de un vértice reducido tiene complejidad lineal $O(a_i)$, donde $a_i = max[|A_i|, A_i \in Pa]$. De esta forma, el tiempo de respuesta ante una petición de búsqueda de camino estaría dado por la complejidad del Algoritmo MDijkstra.

Entre los algoritmos utilizados para búsqueda de caminos óptimos, el de menor complejidad temporal es el algoritmo A* cuando se utiliza una heurística óptima (la complejidad es $O(n)$ donde n es el número de vértices del grafo). Por otra parte, la complejidad temporal del Algoritmo MDijkstra es $O(n_1 \log n_1)$, menor que ${n_1}^2$, donde n_1 es la cantidad de vértices del grafo reducido. Por tanto, si en el proceso de reducción se obtiene un grafo $G_r = (V_r, E_r)$ a partir de un grafo $G = (V, E)$, tal que $n_1 < \sqrt{n}$, $n_1 = |V_r|$, $n = |V|$ la complejidad temporal del Algoritmo MDijkstra sería, en teoría, menor que la del Algoritmo A*.

Generalmente existe una relación inversa entre eficiencia y exactitud en los algoritmos que tienen como entrada un volumen elevado de datos. El principal resultado que se muestra, en cuanto a la búsqueda de caminos óptimos, es una mejora de la eficiencia en la búsqueda de caminos sin afectar la exactitud de la respuesta.

Para referirse a un grafo (G_i o G_j) de la regla de reescritura asociada a un vértice reducido v_r se utilizará la notación $v_r{\cdot}G_i$ y $v_r{\cdot}G_j$. Análogamente se utilizará la notación $v_r{\cdot}\psi_{in}$ y $v_r{\cdot}\psi_{out}$ para referirse a los conjuntos ψ_{in} y ψ_{out} de la regla de reescritura asociada al vértice v_r.

Algoritmo 9: *BuscarCaminoOptimo*
Entrada: Grafo reducido G, vértice de origen v_o y destino v_d, *enfoque y nivel*
Salida: Camino óptimo desde v_o hasta v_d

1. **Si** *enfoque* = 1
2. *ObtenerGrafoNivel*(G, nivel)
3. **Sino**
4. *ExpandirE*(G, v_o)
5. *ExpandirE*(G, v_d)
6. **Fin Si**
7. *MDijkstra*(G, v_o, v_d) (ver Algoritmo 13)
8. **Retornar** *Camino*(v_o, v_d, P_{rn})

Algoritmo 10: *ObtenerVerticeReducido*
Entrada: Grafo reducido G y vértice v a buscar
Salida: Vértice reducido v_r que contiene al vértice v

1. **Para todo** $v_i \in V_r$ **hacer**
2. **Si** (v_i no es reducido y $v_i = v$) **o** (v_i es reducido **y** v_i y v pertenecen a la misma clase **entonces**
3. **Retornar** v_i
4. **Fin Si**
5. **Fin Para**
6. **Retornar** *null*

Algoritmo 11: *ExpandirVerticeReducido*
Entrada: Vértice v_r a expandir y el grafo reducido G al que pertenece
Salida: Un grafo reducido

1. **Para todo** $e \in v_r{\cdot}G_j{\cdot}E_j$ **hacer**
2. *AdicionarArista*(G, e)
3. **Fin Para**
4. **Para todo** $(u_1, c_1, c_2, u_2) \in v_r{\cdot}\psi_{in}$ **hacer**
5. *AdicionarArista*(G, (u_2, u_1, *costo* = c_2))
6. **Fin Para**
7. **Para todo** $(u_1, c_1, c_2, u_2) \in v_r{\cdot}\psi_{out}$ **hacer**

8. *AdicionarArista*(G, (u_1, u_2, *costo* = c_2))
9. **Fin Para**

Algoritmo 12: *ExpandirE*
Entrada: Grafo reducido G y vértice v_e a expandir
Salida: Grafo reducido $G_r = (V_r, E_r, R)$, $v_e \in V_r$

1. Inicializar la variable *nivel* en la cantidad de niveles de reducción del grafo reducido
2. **Repetir**
3. v = *ObtenerVerticeReducido*(G, v_e)
4. **Si** v no es vértice reducido *Terminar*
5. *ExpandirVerticeReducido*(v, G)
6. *nivel* = *nivel*–1
7. **Hasta que** *nivel* = 0

Algoritmo 13 *MDijkstra*
Entrada: Un grafo ponderado reducido $G = (V, E, f, R)$ y un vértice de origen v_o
Salida: Vector D_n de distancias mínimas y vector P_{rn} de predecesores

1. $C_n = \{\}$, *cola* = ϕ{La variable *cola* representa una cola con prioridad}
2. **Para todo** $v \in V$ **hacer**
3. $D_n[v] = f(v_o, v_o, v)$, $P_{rn}[v] = v_o$
4. *Adicionar*(*cola*, v, $D_n[v]$) {Se adiciona el vértice v tomando como prioridad la distancia hasta el mismo.}
5. **Fin Para**
6. **Mientras** !*Vacia*(*cola*) **hacer**
7. w_n = *Extraer*(*cola*), $C_n = C_n \, U\{w_n\}$
8. **Para todo** $v \in$ *Adyacentes*(w_n) **hacer**
9. **Si** $D_n[v] > D_n[P_{rn}[w_n]] + f(P_{rn}[w_n], w_n, v)$ **entonces**
10. $D_n[v] = D_n[P_{rn}[w_n]] + f(P_{rn}[w_n], w_n, v)$
11. $P_{rn}[v] = w_n$
12. *DecrementarLlave*(*cola*, v, $D_n[v]$) {Si $D_n[v]$ es menor que la prioridad de v en la cola se modifica la prioridad.}
13. **Fin Si**
14. **Si** v es reducido entonces
15. Actualizar distancia y predecesor de cada vértice adyacente a v utilizando la función de cruce f
16. **Fin Si**
17. **Fin Para**
18. **Fin Mientras**

RESULTADOS EXPERIMENTALES

Para la validación del modelo se realizó la demostración de corrección de todos los algoritmos diseñados. Estas demostraciones no se incluyen por motivos de espacio. No obstante, se presentan algunos resultados experimentales que muestran la escalabilidad y eficiencia en la búsqueda de caminos mínimos haciendo uso de la propuesta. Los experimentos muestran además que el algoritmo de reducción propuesto garantiza que no existe pérdida de información.

Para la realización de los experimentos se implementó una biblioteca de clases basada en el modelo propuesto. Se utilizó el Lenguaje de Programación Python, la biblioteca de clases NetworkX [25], entre otras bibliotecas auxiliares. La biblioteca de clases NetworkX brinda una implementación de los algoritmos de Dijkstra y A*, entre otros. Esto permitió comparar el tiempo de ejecución del Algoritmo MDijkstra con el tiempo de ejecución de los dos antes mencionados, haciendo uso de la misma tecnología.

Para las pruebas experimentales se utilizaron tres grafos: el primero (G_1), obtenido a partir de una capa vectorial (streets_wake.shp) del mapa del estado Carolina del Norte[2], tiene 41810 vértices. Este grafo fue reducido dos veces (utilizando particiones basadas en el código postal) obteniéndose un grafo de 250 vértices ($G_{r1.1}$) y otro de 1826 vértices ($G_{r1.2}$). El segundo grafo (G_2) se obtuvo a partir de la capa vectorial (grnf012r05a_e.shp) que representa la red de viales de Nova Scotia[3] (Canadá) y tiene 63509 vértices. Este grafo fue reducido dos veces (utilizando dos particiones de vértices de forma tal que todos los vértices de una clase de la partición estuvieran conectados a través de un camino) obteniéndose un grafo de 517 vértices ($G_{r2.1}$) y otro de 936 vértices ($G_{r2.2}$). El tercer grafo representa la red de viales de la ciudad de San Francisco[4], cuenta con 174956 vértices (G_3) y también fue reducido dos veces (utilizando dos particiones de vértices de forma tal que todos los vértices de una clase de la partición estuvieran conectados a través de un camino), obteniéndose un grafo de 769 vértices ($G_{r3.1}$) y el otro de 2617 vértices ($G_{r3.2}$).

Para la obtención de los grafos G_1 y G_2 a partir de archivos en formato *shape*, se utilizó la función *read_shp* de la biblioteca NetworkX. El grafo G_3 se creó a partir de dos archivos en texto plano; uno con los vértices y el otro con las aristas.

[2] http://grass.osgeo.org/sampledata/north_carolina/
[3] http://geodepot.statcan.ca/diss/2006dissemination/data/frr_rnf_e.cfm
[4] http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm

A partir de los datos anteriores, se han definido los siguientes elementos presentes en el diseño de los experimentos:

- Unidad básica de análisis: Grafos.
- Población: Grafos obtenidos a partir de los mapas mencionados anteriormente. La característica distintiva que tienen todos los miembros de la población es que deben ser grafos conexos.
- Muestra:
  - Grafo G_1: Grafo con 41810 vértices.
  - Grafo $G_{r1.1}$: Grafo con 250 vértices.
  - Grafo $G_{r1.2}$: Grafo con 1826 vértices.
  - Grafo G_2: Grafo con 63509 vértices.
  - Grafo $G_{r2.1}$: Grafo con 517 vértices.
  - Grafo $G_{r2.2}$: Grafo con 936 vértices.
  - Grafo G_3: Grafo con 174956 vértices.
  - Grafo $G_{r3.1}$: Grafo con 769 vértices.
  - Grafo $G_{r3.2}$: Grafo con 2617 vértices.

Se hará referencia a los grafos G_1, G_2 y G_3 como los grafos originales y a los restantes como los grafos reducidos.

Los experimentos fueron ejecutados en una Computadora Personal (PC) con un procesador Intel Pentium 4 de 3.20 GHz [512 Kb de cache] con 1.5 Gb RAM, sobre el sistema operativo Kubuntu 11.10. A continuación se enumeran los experimentos realizados:

- Experimento 1: Aplicar los algoritmos de Dijkstra y A* a los grafos originales.
- Experimento 2: Aplicar el Algoritmo MDijkstra a los grafos reducidos.

Cada algoritmo se ejecutó 10 veces registrándose el tiempo de ejecución, descartándose el mayor y menor valor en cada caso. Finalmente, se calculó el promedio de los restantes 8 valores. Los resultados de los experimentos 1 y 2 se resumen en la Tabla 1. La última columna indica si el algoritmo en cuestión obtuvo el valor óptimo.

Como se puede apreciar en la Tabla 1, el algoritmo MDijkstra, propuesto en este trabajo, obtiene un camino óptimo de igual costo al obtenido por el algoritmo de Dijkstra en el grafo original; lo que evidencia que el modelo presentado representa una red de forma adecuada.

Tabla 1. Resultados experimentales

Grafo	**Algoritmo**	**Cant. vértices**	**Tiempo (s)**	**Ópt.**
G_1	Dijkstra	41810	0,16	sí
	A* (h=0)		0,494	sí
	A*(h=dist. euclidiana)		0,020	no
$G_{r1.1}$	MDijkstra	250	0,004	sí
$G_{r1.2}$	MDijkstra	1826	0,026	sí
G_2	Dijkstra	63509	1,011	sí
	A* (h=0)		0,145	sí
	A*(h=dist. euclidiana)		0,285	sí
$G_{r2.1}$	MDijkstra	517	0,016	sí
$G_{r2.2}$	MDijkstra	936	0,034	sí
G_3	Dijkstra	174956	3,025	sí
	A* (h=0)		2,211	sí
	A*(h=dist. euclidiana)		0,101	no
$G_{r3.1}$	MDijkstra	765	0,019	sí
$G_{r3.2}$	MDijkstra	2617	0,072	sí

También se puede afirmar que el modelo de representación propuesto garantiza escalabilidad en la búsqueda de caminos óptimos. Esta afirmación se basa en que dado un grafo conexo se puede encontrar una partición que genere un grafo lo suficientemente reducido para que el tiempo de respuesta ante una petición de búsqueda de camino óptimo sea del orden de milisegundos; tal como se muestra en los resultados experimentales obtenidos.

Varios algoritmos heurísticos se han diseñado para reducir el tiempo de respuesta de la búsqueda de caminos óptimos en SIG reduciendo el espacio de búsqueda de la solución, para ello se asume que un bajo porcentaje de error es admisible en esta área. Como se puede apreciar en la Tabla 1, el algoritmo A* retorna una respuesta en tiempos menores que el algoritmo de Dijkstra, pero en algunos casos no alcanza el óptimo. Sin embargo, con el modelo propuesto es posible obtener un camino óptimo en un tiempo similar al que presentan algunos algoritmos heurísticos, e incluso menor, como se muestra en la tabla antes mencionada. Además, el Algoritmo MDijkstra es escalable en la muestra seleccionada,

el tiempo de respuesta varía entre 0,004 y 0,072 segundos (una diferencia de 6,8 milisegundos) para un incremento de 107955 vértices del grafo original.

Adicionalmente, en [26] se presenta una implementación del modelo propuesto en esta investigación, mostrando la viabilidad de implementación como parte de un motor de persistencia de grafos que realiza el análisis en memoria externa. Lo cual sería conveniente si se cuenta con una PC que no posee recursos de *hardware* suficientes para realizar el análisis en RAM.

CONCLUSIONES

En el presente artículo se ha descrito un modelo para la representación de redes en SIG basado en grafos reducidos. Como parte del mismo se han diseñado varios algoritmos. También se describe un algoritmo para realizar búsquedas de caminos óptimos en grafos reducidos garantizando escalabilidad y eficiencia en el análisis.

Finalmente se concluye lo siguiente:

- Al obtener un grafo a partir de una red representada en una capa vectorial de un mapa, se cuenta con un modelo robusto y a la vez flexible para la realización de análisis de redes. Este es un modelo ampliamente estudiado y cuenta con una gran variedad de aplicaciones.
- El uso de la reescritura de grafos propuesta permite reducir un grafo sin que exista pérdida de información.
- El uso de un grafo reducido para realizar búsqueda de caminos óptimos permite dar respuesta de forma eficiente cuando las redes son grandes.
- El uso de grafos reducidos para la representación de redes en SIG facilita la realización de análisis de redes a diferentes escalas.
- La generalidad del modelo propuesto radica en la posibilidad de su aplicación en distintos tipos de redes debido a que se garantiza que no existe pérdida de información.

REFERENCIAS

[1] S. Aronoff. "Geographical Information Systems: A management perspective". WDL Publications. Ottawa, Canadá, pp. 294. 1989.

[2] ESRI Shapefile Technical Description. pp. 1-5. 1998. Fecha de consulta: 10 de junio de 2011. Disponible en: www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[3] L. Daniel, P. Loree and A. Whitener. "Inside MapInfo Professional". OnWord Press. Third ed. New York, USA, pp. 576. 2001. ISBN: 0766834727.

[4] M. Haklay and P. Weber. "OpenStreetMap: User-Generated Street Maps". IEEE Pervasive Computing. Vol. 7 Nº 4, pp. 12-18. Octubre 2008. ISSN: 1536-1268.

[5] G. Korte. "The Gis Book". OnWord Press. fifth ed. New York, USA, pp. 387. 2001. ISBN: 9780766828209.

[6] pgRouting Project. 2011. Fecha de consulta: 16 de marzo de 2011. Disponible en: http://www.pgrouting.org/

[7] F. Campos Gutiérrez, J.P. Castro Fernández and R. García Martín. "IDELabRoute: Librería para la gestión de grafos escalable". IV Jornadas SIG Libre. Girona, España. Marzo 2010.

[8] pgRouting. Routing from point to point. 2011. Fecha de consulta: 16 de marzo de 2011. Disponible en: http://pgrouting.postlbs.org/discussion/topic/353

[9] R.R. Puente and R.S. Castro. "Enfoque de hipergrafos en sistemas de información geográfica para la modelación de redes". VII Congreso Internacional Geomática 2011. La Habana, Cuba. Febrero 2011.

[10] R. Angles and C. Gutiérrez. "Survey of graph database models". ACM Computing Surveys. Vol. 40 Nº 1, pp. 1-39. Febrero 2008. ISSN: 0360-0300. DOI:10.1145/1322432.1322433.

[11] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen and D. Wilkins. "A comparison of a graph database and a relational database: a data provenance perspective". 48th Annual Southeast Regional Conference. Oxford, Mississippi, USA. Abril 2010.

[12] E.W. Dijkstra. "A Note on Two Problems in Connection with Graphs". Numerische Mathematik. Vol. 1, pp. 269-271. Diciembre 1959. ISSN: 0029-599X. DOI:10.1007/BF01386390.

[13] P. Hart, N. Nilsson and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions

on Systems Science and Cybernetics. Vol. 4 Nº 2, pp. 100-107. Julio 1968. ISSN: 0536-1567. DOI:10.1109/TSSC.1968.300136.

[14] GRASS Programmer's Manual: Directed Graph Library. 2008. Fecha de consulta: 16 de marzo de 2011. Disponible en: http://grass.osgeo.org/programming6/dglib.html

[15] R. Blazek, M. Neteler and R. Micarelli. "The new GRASS 5.1 vector architecture". Open source GIS-GRASS users conference. University of Trento, Italy. Septiembre 2002.

[16] S. Wasserman and K. Faust. "Social Network Analysis. Methods and Applications". Cambridge University Press. Cambridge, UK, p. 857. 1995. ISBN: 0521387078.

[17] R. Sedgewick and K. Wayne. "Algorithms". Addison-Wesley Professional. Fourth ed. Boston, USA, p. 976. 2011. ISBN: 978-0321573513.

[18] S.P. Borgatti and D.S. Halgin. "On Network Theory". Organization Science. Vol. 22. Nº 5, pp. 1168-1181. Abril 2011. ISSN: 1526-5455. DOI:10.1287/orsc.1100.0641

[19] P. Sanders and D. Schultes. "Highway hierarchies hasten exact shortest path queries". 13th annual European conference on Algorithms. Palma de Mallorca, España. Octubre 2005.

[20] Z. Wang, O. Che, L. Chen and A. Lim. "An efficient shortest path computation system for real road networks". 19th international conference on Advances in Applied Artificial Intelligence: industrial, Engineering and Other Applications of Applied Intelligent Systems. Annecy, France. Junio 2006.

[21] G. Jagadeesh and T. Srikanthan. "Route computation in large road networks: a hierarchical approach". Intelligent Transport Systems. Vol. 2 Nº 3, pp. 219-227. Septiembre 2008. ISSN: 1751-956X. DOI:10.1049/iet-its:20080012.

[22] Q. Song and X. Wang. "Efficient Routing on Large Road Networks Using Hierarchical Communities". IEEE Transactions on Intelligent Transportation Systems. Vol. 12 Nº 1, pp. 132-140. Marzo 2011. ISSN: 1524-9050. DOI:10.1109/TITS.2010.2072503.

[23] F.E. Cellier and J. Greifeneder. "Continuous System Modeling". Springer-Verlag. NJ, USA, p. 755. 1991. ISBN 978-0-387-97502-3.

[24] M. de Berg, O. Cheong, M. van Krefeld and M. Overmars. "Computational Geometry: Algorithms and Applications". Springer-Verlag. Third ed. New York, USA, p. 386. 2008.

[25] A.A. Hagberg, D.A. Schult and P.J. Swart. "Exploring network structure, dynamics, and function using NetworkX". 7th Python in Science Conference. Pasadena, CA, USA. Agosto 2008.

[26] J.A. Inda Herrera, R. Rodríguez Puente y M. Lazo Cortés. "Sistema para la búsqueda de caminos mínimos en grafos grandes". I Taller de Geoninformática. La Habana, Cuba. Febrero 2012.