Ingeniería Industrial/ISSN 1815-5936/Vol. XXXVIII/No. 1/enero-abril/2017/p. 32-42

ARTÍCULO ORIGINAL
INVESTIGACIÓN DE OPERACIONES

Búsquedas de caminos mínimos haciendo uso de grafos reducidos

Shortest path search using reduced graphs

Rafael Rodríguez-Puente, Manuel Lazo-Cortés

Universidad de las Ciencias Informáticas, Facultad 3. La Habana, Cuba
E-mail: rafaelrp@uci.cu, manuelslc@uci.cu

Recibido: 22 de octubre de 2013
Aprobado: 21 de octubre de 2016

RESUMEN
Uno de los problemas de optimización ampliamente estudiados consiste en la búsqueda de caminos mínimos desde un origen a un destino. Los algoritmos clásicos que solucionan este problema no son escalables. Se han publicado varias propuestas utilizando heurísticas que disminuyen el tiempo de respuesta; pero las mismas introducen un error en el resultado. El presente trabajo propone el uso de un algoritmo de reducción de grafos sin pérdida de información, el cual contribuye a reducir el tiempo de respuesta en la búsqueda de caminos. Además, se propone una modificación al algoritmo de Dijkstra para ser utilizado sobre grafos reducidos, garantizando la obtención del óptimo en todos los casos. También se realiza una comparación entre el tiempo de ejecución de la propuesta y los algoritmos de Dijkstra y A*, la cual muestra que es posible obtener un camino óptimo en tiempos similares, e incluso inferiores, a los obtenidos por algoritmos heurísticos.

Palabras clave: Algoritmo de Dijkstra, búsqueda de camino mínimo, reducción de grafos.

ABSTRACT
One of the widely studied ptimization problems consistsin finding shortest paths from a source to a destination. Classical algorithms for solving this problemare not scalable. Several approaches have been proposed, using heuristics, to reduce the run time. These approaches introduce anerrorand, consequently, they do not guaranteean optimum value inall cases. In this paper we propose agraph reduction algorithm without loss of information, which helps to reduce the run time of algorithms. In addition, we propose a modification to the Dijkstraalgorithmto be used on reduced graphs, guaranteeing an optimal result in all cases. Besides, a comparison between the run time of the proposed algorithms and Dijkstraand A* algorithms is presented. This comparison shows that it ispossible to obtain an optimal pathin a similar time, and even in less time, than those obtained by heuristic algorithms.

Key words: Dijkstra algorithm, shortest path search, graph reduction.

I. INTRODUCCIÓN

La búsqueda de caminos mínimos ha sido ampliamente estudiada, se puede encontrar aplicaciones en varias ramas de la ciencia.

Uno de los algoritmos clásicos y más utilizados para el cálculo del camino mínimo desde un origen a un destino es el algoritmo de Dijkstra, el mismo fue enunciado por primera vez por Edger Dijkstra en el año 1959 [1] y es uno de los algoritmos más utilizados y discutidos en la literatura de grafos, la complejidad temporal es $O(n^2)$. Sin embargo, este algoritmo no es eficiente para realizar búsquedas de camino mínimo en grafos grandes [2]. Se debe aclarar que el término grande es relativo y depende del *hardware* que se disponga.

Varios autores definen heurísticas para disminuir el tiempo de respuesta [3,4], sin embargo, estas heurísticas introducen un margen de error, el cual no permite obtener el óptimo en todos los casos. A continuación, se describen algunos de los algoritmos heurísticos más relevantes.

Uno de los algoritmos heurísticos más utilizados es el algoritmo A* (A-star) [5]. El objetivo principal del mismo es reducir el tiempo de ejecución mediante la reducción del espacio de búsqueda, teniendo en cuenta solo los vértices que tienen mejores posibilidades de aparecer en el camino mínimo. Si la heurística seleccionada es óptima, la complejidad del algoritmo se reduce a $O(n)$. Por esta razón el algoritmo A* es ampliamente utilizado para búsquedas de caminos mínimos.

Noto y Sato (2000) proponen una extensión del algoritmo de Dijkstra basado en la búsqueda bidireccional [6]. Este algoritmo es capaz de dar respuesta en un tiempo dado. Si el tiempo de búsqueda especificado se alcanza por el algoritmo, se utiliza otro enfoque para el resto de la búsqueda del camino mínimo utilizando algoritmos genéticos. Con este enfoque se logra un tiempo de respuesta bajo, pero este algoritmo está basado en el algoritmo de Dijkstra por lo que no es adecuado para grafos grandes.

Otro enfoque estudiado para la búsqueda de caminos mínimos en grafos grandes está relacionado con el uso de determinadas propiedades de la red que representa el grafo para la reducción del espacio de búsqueda. En este sentido se pueden mencionar varios trabajos; por ejemplo, Gutman (2004) propone un algoritmo en el cual se define un atributo formal llamado *reach* [7]. Este atributo es una medida de la relevancia del vértice, es precalculado utilizando el grafo y contribuye a la reducción del tiempo de ejecución de la búsqueda de caminos mínimos.

Goldberg y Harrelson (2005) definen el algoritmo Landmark-A* [8], el aporte en este caso es una técnica basada en preprocesamiento para realizar el cómputo de los puntos de referencia. Una vez elegidos dichos puntos, se calcula y almacena el camino mínimo desde cada vértice del grafo a cada punto de referencia y viceversa. Esta información luego es utilizada para reducir el espacio de búsqueda. Este algoritmo es similar al propuesto por Gutman, los dos basan su estrategia en determinar puntos "importantes" para la búsqueda de caminos mínimos atendiendo a distintos criterios. Como se puede apreciar, el tiempo de preprocesamiento crece considerablemente con el crecimiento de la cantidad de vértices del grafo y la cantidad de puntos de referencia.

Otro enfoque relevante que utiliza las propiedades de la red sobre la que se realiza el análisis está relacionado con el uso de la jerarquía presente en una red de viales. Varias estrategias utilizan este enfoque; por ejemplo, Sanders y Shultes (2005) proponen un algoritmo para la construcción y consulta de redes jerárquicas [9], los autores logran obtener un tiempo de ejecución bajo mostrando la viabilidad de este método. Gonzalez et al. (2007) utilizan la jerarquía de las calles para particionar la red de viales en áreas y precalcular caminos mínimos en estas áreas [10]. Este enfoque utiliza el hecho de que algunas calles son más transitadas que otras y los choferes usualmente utilizan las calles más largas y anchas.

Geisberger et al. (2008)proponen un enfoque que utiliza solamente las aristas que están relacionadas con vértices "importantes" [11]. Efentakis et al. presentan un algoritmo de búsqueda de camino mínimo que imita el comportamiento de los conductores humanos explotando la jerarquía de la red de viales [12].

Delling et al. (2009) muestran una revisión de algoritmos de cálculo de rutas [13], todas las estrategias muestran importantes avances en el cálculo de caminos mínimos y hacen posible un bajo tiempo de respuesta en grafos grandes a través del uso de heurísticas.

Sanders y Shultes (2007) definen un método que es similar al propuesto por Gutman [14]. Los autores tienen en cuenta vértices relevantes (*transitnodes*) para precalcular caminos mínimos para viajes de larga distancia. Para ello, realizan el cálculo del camino mínimo entre cada par de vértices relevantes y el camino mínimo desde cada origen o destino potencial hasta los vértices relevantes más cercanos. Este enfoque necesita una noción efectiva de viaje de larga distancia, por otra parte, el óptimo se garantiza dependiendo del filtro seleccionado.

Los enfoques descritos anteriormente se basan en la idea de que para el cálculo de caminos largos (en redes grandes), solo se necesitan las calles de los niveles más altos (autopistas, calles más transitadas, etc.) de la red jerárquica. Esta consideración reduce el tiempo de ejecución de la búsqueda de caminos mínimos pero no garantiza la obtención del camino óptimo.

Los sistemas comerciales utilizan algoritmos heurísticos para el cálculo de caminos mínimos con el objetivo de reducir el tiempo de respuesta a este tipo de petición. Varios autores han diseñado algoritmos para este propósito, a continuación se describen dos de ellos.

Liu y Yang (2009) proponen un algoritmo heurístico, en el cual se define una función de restricción espacial para disminuir el costo de la búsqueda de caminos mínimos [15]. Los autores prueban, en los resultados experimentales, que el tiempo de respuesta de este algoritmo está cercano al del algoritmo A*.

Fei et al. (2010) describen un sistema que puede mostrar información espacial y atributos de varias secciones de una calle, analiza información del estado del tráfico y de accidentes de tráfico y es capaz de mostrarla en tiempo real [16]. Este sistema utiliza el formato TAB de MapInfo para almacenar la cartografía y la información adicional (datos socio-económicos) la almacena en SQL Server 2000. Se utilizan completamente sistemas propietarios que en este caso se descartan por las implicaciones que tiene en la economía del país y en la soberanía tecnológica.

Los algoritmos heurísticos son muy importantes en la solución de problemas de alto costo computacional; sin embargo, introducen un margen de error y por tanto no garantizan la obtención del camino óptimo en todos los casos. La complejidad de aplicación está ligada en gran medida a la selección de una heurística óptima

Por otra parte, existen varios algoritmos que permiten reducir un grafo [17-19], al aplicar cualquier algoritmo sobre el grafo reducido evidentemente se logrará un menor tiempo de respuesta, pero en los mismos la reducción trae como consecuencia pérdida de información, por lo que no se puede garantizar que se obtenga el óptimo.

En el presente artículo se propone un mecanismo para disminuir el tiempo de respuesta de las búsquedas de caminos mínimos haciendo uso de un algoritmo de reducción de grafos sin pérdida de información. Además, se expone una modificación al algoritmo de Dijkstra para realizar búsquedas de caminos mínimos en grafos reducidos. Los resultados experimentales muestran que los costos obtenidos por los algoritmos de Dijkstra y A* son superiores a los obtenidos por el Dijkstra modificado sobre los grafos reducidos.

II. MÉTODOS

El problema de la búsqueda de caminos mínimos en un grafo ponderado $G=(V,E,fc)$ desde un vértice v_i hasta otro v_j consiste en buscar un camino desde v_i a v_j, tal que el valor $\sum_{k=1}^{j} fc(k,k+1)$ sea mínimo.

En varios problemas que son resueltos haciendo uso de la teoría de grafos, es común la necesidad de transformar un grafo. En particular, una transformación puede tratarse como reducción si el nuevo grafo obtenido tiene una menor cantidad de vértices.

A continuación se muestran notaciones y definiciones necesarias para la comprensión del presente trabajo.

Las gramáticas de grafo se pueden clasificar según varios criterios, atendiendo a la forma en que se transforman los grafos y siguiendo el enfoque clásico de Janssens y Rozenberg se pueden clasificar en gramáticas NCE [20], NLC [21], HR y NR [22], etc., la especificación de cómo se debe transformar un grafo, se denomina mecanismo de empotrado.

Debido al mecanismo de empotrado que utilizan los tipos de gramáticas antes mencionadas, se selecciona para utilizar en este trabajo el tipo de gramáticas NCE. Para profundizar sobre las diferencias entre los distintos mecanismos de empotrado, el lector puede remitirse a [23].

Definición. Una gramática de grafo NCE es un sistema $Gg = (\Sigma, \Delta, \mathrm{Prod}, S)$ donde:

- Σ es un conjunto finito y no vacío denominado alfabeto.
- Δ es un subconjunto de Σ denominado alfabeto de los símbolos terminales.
- Prod es un conjunto finito de producciones o reglas de reescritura de la forma (α, β, ψ), donde α es un grafo conexo, β es un grafo y $\psi : V_\alpha \times V_\beta \times \Sigma \to (0,1)$, ψ se denomina función de empotrado y está determinada por el mecanismo de empotrado que se utilice.
- S es el símbolo distinguido o axioma.

La aplicación de una producción o regla de reescritura consiste en eliminar el subgrafo α del grafo al que se le va a aplicar la regla, adicionar el grafo β a dicho grafo y conectarlo según se especifique en ψ.

La formalización de gramática NCE utiliza como base la definición de grafos no dirigidos y etiquetados por lo que cuando el grafo es dirigido se deben tener en cuenta consideraciones adicionales. Para dar soporte a grafos dirigidos se puede seguir el mismo enfoque que en [21], sustituyendo la función ψ por dos funciones: ψ_{in} y ψ_{out} para las aristas que entran o salen de un vértice respectivamente.
En correspondencia con las definiciones y enfoques analizados, se enuncian las siguientes definiciones de regla de reescritura y grafo reducido.

Definición. Una regla de reescritura de grafos sobre un grafo $G = (V, E)$ 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 de información de empotrado, de la forma (v_m, c_1, c_2, v_n), donde: $c_1, c_2 \in \Re^+, v_m \in V_j, v_n \in (V - V_j)$. En el caso de ψ_{in}, $\exists (v_n, v_i) \in E, f_c(v_n, v_i) = c_1$, luego de aplicar la regla de reescritura, se obtiene el grafo $G_1 = (V_1, E_1, f_{c1})$ y se cumple que $\exists (v_n, v_m) \in E_1, f_{c1}(v_n, v_m) = c_2$. Análogamente a ψ_{in}, se define ψ_{out}, con la única diferencia de la orientación de las aristas.

Las aristas que unen el vértice v_i y los vértices del grafo $G - G_i$ se denominan aristas preempotradas. Después de aplicar una regla de reescritura, las aristas que unen los vértices del grafo G_j con los vértices del grafo $G - G_i$ se denominan aristas postempotradas.

El conjunto ψ_{in} permite transformar el conjunto de aristas preempotradas que inciden en el vértice V_i en aristas postempotradas que inciden en uno o más vértices $v_j \in V_j$, de forma similar ψ_{out} permite transformar las aristas preempotradas que salen de v_i en aristas postempotradas que salen de uno o más vértices $v_j \in V_j$.

Teniendo en cuenta la definición de regla de reescritura enunciada anteriormente, en la Figura 1 se muestra un ejemplo de regla de reescritura. En la parte izquierda se muestra el grafo $G_i = (\{v_i\}, \{\})$, en la parte derecha se muestra el grafo G_j y en la parte inferior se muestra la información de empotrado

ψ_{in} y ψ_{out}. Se puede comprobar, que luego de aplicar dicha regla al grafo de la Figura 2 b según el mecanismo seleccionado, se obtiene el grafo de la Figura 2a.

Para referirse a los grafos G_i y G_j de la regla de reescritura asociada a un vértice reducido v_r se utilizará la notación $v_r.G_i$ y $v_r.G_j$ respectivamente. Análogamente, se utilizará la notación $v_r.\psi_{in}$ y $v_r.\psi_{out}$ para referirse a las funciones ψ_{in} y ψ_{out} de la regla de reescritura asociada al vértice v_r.

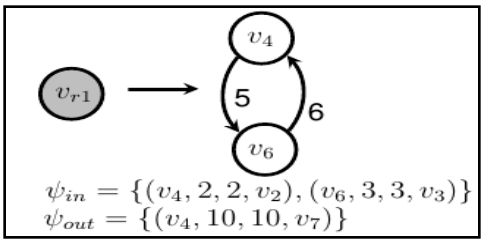
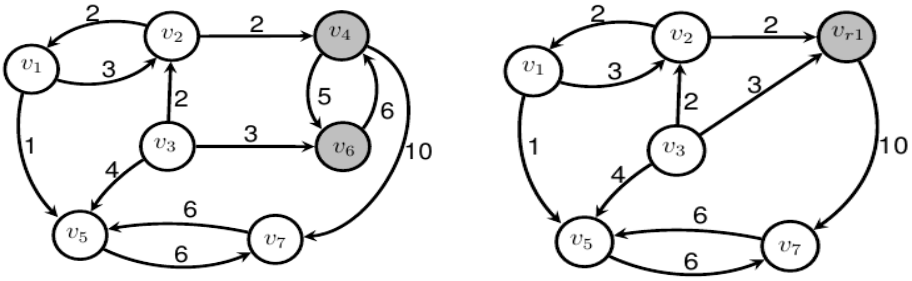


Fig. 1. Ejemplo de regla de reescritura



a) Ejemplo de grafo b) Grafo reducido

Fig. 2. Ejemplos de grafo

Definición. Un grafo reducido es una tuplaG= (V_r, E_r,f, R), donde:

- $V_r$es un conjunto de vértices.
- $E_r$es un conjunto de aristas.
- $f : Vr \times Vr \times V_r \rightarrow (\Re^+ \cup \infty)$, es una función que para cada (V_i,V_j, V_k) retorna el costo de ir desde $V_i$hasta V_ka travésV$_j$
- R es un conjunto de reglas de reescritura sobre (V_r, E_r).

Definición. Sean G y G_r dos grafos, $R = \{r_1, r_2, .., r_n\}$ 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.

A partir de lo anterior, se enuncia un algoritmo de reducción de grafos sin pérdida de información relevante para la búsqueda de caminos mínimos. Posteriormente, se presenta una modificación al algoritmo de Dijkstra para realizar búsqueda de caminos mínimos en grafos reducidos con el algoritmo de reducción propuesto.

Reducción de grafos

La reducción de grafos se realiza utilizando el Algoritmo 1, en este epígrafe se presenta dicho algoritmo y se expone una breve explicación del mismo.

En el caso del algoritmo de reducción que se propone, se debe garantizar que no existan vértices reducidos[1] que sean adyacentes con el objetivo de poder obtener un camino óptimo de igual costo al obtenido en una búsqueda en el grafo original, para contribuir al logro de este objetivo, se enuncia la siguiente definición:

Definición. Sea un grafo $G=(V,E)$ y una relación de equivalencia RE 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 $RE(v_i,v_j)$ (o sea, $[v_i]=[v_j]$). Un vértice $v_i \in V$ es exterior si $\exists(v_i,v_j)\in E$ o [$\exists(v_j,v_i)\in E$ y $\neg RE(v_i,v_j)$].

Si la entrada del algoritmo es una relación de equivalencia, lo primero que se hace es obtener una partición a partir de dicha relación. Para crear esta partición se puede seguir una estrategia voraz que divida el conjunto de vértices V en subconjuntos disjuntos según RE. La estrategia consistiría en comparar cada elemento del conjunto V con los restantes elementos para determinar cuáles están relacionados (según RE) y a partir de estos crear las clases que forman la partición. Si una clase está formada por un solo elemento, este será un vértice exterior. Si la clase tiene más de un elemento, entonces todos estos serán vértices interiores.

Para construir el conjunto de vértices del grafo reducido, se crea un vértice por cada clase de P.
Para adicionar una arista al grafo reducido, los dos vértices que la forman deben pertenecer a clases distintas. En caso de que exista más de una arista, en el grafo de entrada del algoritmo, entre los vértices que pertenecen a dos clases distintas, se adiciona la de menor costo.

Algoritmo 1: Reducción de grafos.
Entrada: Un grafo ponderado y posiblemente reducido $G=(V_r,E_r,f,R)$ que representa un mapa, donde R es un conjunto de reglas de reescritura, posiblemente vacío. Una relación de equivalencia RE que permite crear una partición P en V o una partición P en V.
Salida: Un grafo ponderado reducido que representa el mapa

1. Si la entrada es una relación de equivalencia
2. Obtener la partición $P=(V/RE)=A_1,A_2,...,A_s$, donde $A_i=[a_i], a_i \in V, i=1..s$
3. Fin Si
4. Construir el conjunto de vértices V_r del grafo reducido G_r a partir de las clases de la partición P
5. Construir el conjunto de aristas E_r, donde $(v_{rm},v_{rk})\in E_r$ si y sólo si $v_{rm}\in A_i, v_{rk}\in A_j, i\neq j$ y $\exists(v_m,v_k)\in E$ (v_m,v_k) con $v_m \in A_i, v_k \in A_j$ tal que $f(v_m,v_k)$ sea mínimo. Hacer $f_r(v_{rm},v_{rk})=f(v_m,v_k)$
6. Para todo $A_i \in P$
7. Si $|A_i|>1$, escribir una regla de reescritura que permita sustituir el vértice reducido v_{ri} por el subgrafo que representa dicho vértice.
8. Fin Para
9. Crear el grafo reducido $G_r=(V_r,E_r,f_r,R_r)$
10. Para todo $A_i \in P$
11. Si $|A_i|>1$, $Actualizarf_r(G,R_r[a_i].G_i,R_r[A_i].G_j,f_r)$
12. Fin para
13. Retornar G_r

1 Si un vértice de un grafo reducido es creado a partir de una clase de equivalencia Ai , tal que |Ai | > 1, el vértice se considera reducido. Si |Ai | = 1 el vértice se considera no reducido.

Por cada clase de equivalencia A_i, tal que en la misma existan varios vértices, se crea una regla de reescritura. Este paso es esencial en el algoritmo de reducción, es lo que garantiza que no haya pérdida de información en el proceso de reducción. La regla de reescritura especifica la forma en que estaban conectados los vértices que pertenecen a A_i con el resto de los vértices en el grafo de entrada del algoritmo. Esta información es vital para que el proceso de reducción sea reversible.

El paso 11 es el encargado de actualizar la función de costo del grafo reducido. Esta función, almacena el costo de ir desde un vértice hasta otro pasando por uno reducido.

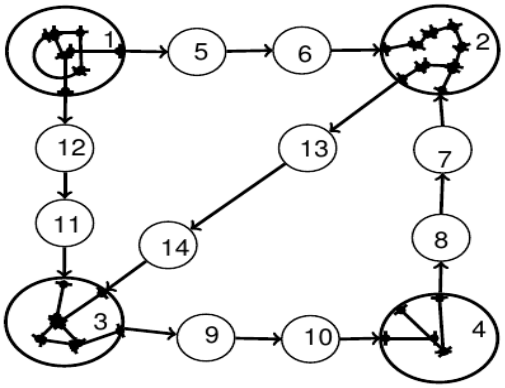


Fig. 3. Ejemplo de grafo reducido

El Algoritmo 2 muestra una forma de calcular esta función. En el caso de ir desde un vértice V_ia otro v_j que no están reducidos, se utilizaría la misma función $f(v_i, v_i, v_j)$. Note que por definición $f_c(v,v) = 0$ y por tanto $f(v_i, v_i, v_j) = f_c(v_i, v_j)$.

Búsqueda de caminos mínimos en grafos reducidos

La modificación que se propone realizar al algoritmo de Dijkstra para realizar búsqueda de caminos mínimos sobre un grafo reducido es sustituir:

- $D[v] > D[w] + f(w,v)$ por $D[v] > D[P[w]] + f(P[w], w, v)$
- $D[v] = D[w] + f(w,v)$ por $D[v] = D[P[w]] + f(P[w], w, v)$

Este cambio es necesario porque para obtener un camino mínimo se necesita 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 vértice por el que se desea pasar, del vértice precedente en el camino que se está calculando y del vértice al cual se va a llegar.

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 6 o desde el 7. Debido a que cada vértice reducido representa un subgrafo del grafo original el costo de recorrer dicho subgrafo dependerá, en cierta medida, del vértice desde el cual se llega al mismo.

Algoritmo 2: Cálculo de la función f

Entrada: Un grafo ponderado y reducido $G=(V,E,f,R)$, los subgrafos $G_i=(\{v_i\},\{\})$ y $G_j=(V_j,E_j)$ de la regla de reescritura asociada a una clase de equivalencia A_i de la partición P.

Salida: La función f para el vértice reducido correspondiente a A_i.

1. Crear un grafo auxiliar $G_{aux}=(V_{aux},E_{aux},f_{aux})=G_j=(A_i,E_i,f_c)$
2. Vértices Adyacentes={}
3. Para todo $v \in A_{aux}$
4. tmp={}
5. Para todo $b \in Adyacentes(v)$
6. Si $b \notin A_i$
7. G_{aux}. adicionar Vertice (b)
8. G_{aux}. adicionar Arista (v,b)
9. $f_{caux}(v,b=f_c(v,b))$
10. Vértices Adyacentes. Adicionar(b)
11. Fin Si
12. Fin Para
13. Fin Para
14. Para todo $v_o \in verticesAdyacentes$
15. Dijkstra (v_o,G_{aux})
16. Para todo $v_d \in verticesAdyacentes$, $v_o \neq v_d$
17. $f(v_o,v_d,v_i)=(D[v_d],camino(v_o,v_d,P_r))$ (D y P_r forman parte del resultado del paso 15)
18. Fin Para
19. Fin Para
20. Retornar f

Experimentos

Para las pruebas experimentales se utilizaron dos grafos. El primero, obtenido a partir de la cartografía del estado Carolina del Norte[2], tiene 41810 vértices (G_1). Este grafo fue reducido dos veces (utilizando una relación de equivalencia basada en el código postal) obteniéndose un grafo de 250 vértices ($G_{r1.1}$)y otro de 1826 vértices ($G_{r1.2}$). El segundo grafo, representa la red de viales de la ciudad de San Francisco[3], cuenta con 149756 vértices (G_2) y también fue reducido dos veces, obteniéndose un grafo de 769 vértices ($G_{r2.1}$) y otro de 2617 vértices (G_{r22}).

Los algoritmos propuestos fueron implementados utilizando el Lenguaje de Programación Python, la biblioteca de clases NetworkX [24], entre otras bibliotecas auxiliares. La biblioteca NetworkX, brinda una implementación de los algoritmos de Dijkstra, A*, entre otros; esto permitió comparar el tiempo de ejecución del algoritmo de búsqueda de camino mínimo propuesto (algoritmo de Dijkstra modificado) con el tiempo de ejecución de los dos antes mencionados haciendo uso de la misma tecnología.

Los experimentos fueron ejecutados en una computadora con un procesador Intel(R) Pentium(R) 4 de 3.20GHz [512 Kb de cache] con 1.5Gb 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 dos grafos originales.
- Experimento 2: Aplicar el algoritmo de Dijkstra modificado a los cuatro grafos reducidos.

2 Disponible en http://grass.osgeo.org/sampledata/north_carolina/
3 Disponible en http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm

Cada algoritmo se ejecutó 10 veces, descartándose el mayor y menor valor en cada caso. Finalmente, se calculó el promedio de los restantes 8 valores.

II. RESULTADOS Y DISCUSIÓN

Los resultados de los experimentos 1 y 2 se muestran en la Tabla 1.

Tabla 1. Tiempo de ejecución del algoritmo de Dijkstra

Grafo	Algoritmo	Número de vértices	Tiempo en segundos	Óptimo
G_1	Dijkstra	41810	0,6160	Si
	A* (h=0)		0,4938	No
	A* (h=Distancia euclidiana)		0,0200	No
$G_{r1.1}$	Algoritmo propuesto	250	0,0036	Si
$G_{r1.2}$		1826	0,0265	Si
G_2	Dijkstra	149756	3,0249	Si
	A* (h=0)		2,2108	No
	A* (h=Distancia euclidiana)		0,1011	No
$G_{r2.1}$	Algoritmo propuesto	765	0,0193	Si
$G_{r2.2}$		2617	0,0722	Si

A* en los dos grafos originales,así como del algoritmo propuesto sobre los grafos reducidos

La complejidad temporal del algoritmo A*, utilizando una heurística óptima, esO(n),donde n es el número de vértices del grafo sobre el cual se realiza el análisis. Por otra parte, la complejidad temporal del algoritmo de Dijkstra modificado que se propone es $O(n_1^2)$, donde n_1 es la cantidad de vértices del grafo reducido. Por tanto, si en el proceso de reducción se obtiene un grafo $Gr = (V_r, E_r)$ a partir de un grafo G = (V,E), tal que $n_1 = |Vr|, n = |V|, n_1 \leq \sqrt{n}$, la complejidad temporal del algoritmo propuesto sería, en teoría, similar y en ocasiones menor, a la complejidad del algoritmo A* utilizando una heurística óptima. Los resultados experimentales obtenidos están en correspondencia con este análisis.

El análisis anterior se corresponde con los resultados mostrados en la Tabla 1, solo se obtuvo un tiempo superior al algoritmo A* en el grafo$G_{r1.2}$, el cual tiene una cantidad de vértices superior, en más de 1500 unidades, a la raíz cuadrada de la cantidad de vértices del grafo G_1(el grafo a partir del cual se obtuvo $G_{r1.2}$). En el caso del grafo G_{r22} pasa algo similar con relación a la cantidad de vértices, sin embargo se obtiene un tiempo inferior al algoritmo A* debido a la diferencia que hay entre la cantidad de vértices de los grafos G_2y G_{r22}.

Adicionalmente se puede observar que el algoritmo propuesto obtiene el óptimo en todos los casos, mientras que el algoritmo A* no obtiene el óptimo en ninguno de los casos, para los experimentos realizados.

Es importante señalar que la complejidad del algoritmo de Dijkstra puede ser menor ($O(n \lg n)$), si se utiliza una cola con prioridad, en particular si se utiliza la implementación que hace uso de la estructura de datos *heap*de *Fibonacci*. En este trabajo no se utilizó la implementación antes mencionada para evitar tiempos de respuesta inferiores debido al uso de estructuras de datos diferentes en los algoritmos utilizados.

Generalmente existe una relación inversamente proporcional entre la eficiencia y la exactitud de un algoritmo que tiene como entrada un volumen elevado de datos. El principal resultado que se muestra en este trabajo es la mejora en la eficiencia en la búsqueda de camino mínimo sin afectar la exactitud del resultado.

La propuesta brinda la posibilidad de realizar búsqueda de caminos mínimos entre cada par de vértices del grafo sin reducir. Esto puede ser llevado a cabo aplicando la regla de reescritura asociada a un vértice escogido convenientemente. Sin embargo, esto trae consigo costo adicional. No obstante, si el sistema que se utiliza para realizar la búsqueda de caminos mínimos, cada vez que se seleccione un origen o destino, se realiza la expansión del vértice reducido correspondiente (en caso de que el origen o el destino estén como parte de un vértice reducido), cuando se ejecute la acción de buscar el camino mínimo la expansión de un vértice reducido no tendría efecto alguno sobre el tiempo de respuesta del algoritmo.

Adicionalmente, Inda et al. presentan una implementación de la propuesta realizada en esta investigación, mostrando la viabilidad de implementación como parte de un motor de persistencia de grafos que realiza el análisis en memoria externa [25]; lo cual sería conveniente si se cuenta con una computadora que no tiene recursos de *hardware* suficientes para realizar el análisis en Memoria de Acceso Aleatorio.

En este caso, la implementación relacionada con análisis de redes se realizó como parte del propio motor de persistencia, lo que garantiza la eficiencia en cuanto al acceso a los datos, sin necesidad de utilizar memoria adicional para los mismos.

IV. CONCLUSIONES

Como resultado de la presente investigación se obtuvo un algoritmo de reducción de grafos sin pérdida de información, el mismo permite obtener tiempos menores en la búsqueda de caminos mínimos. Además se realizó una modificación al algoritmo de Dijkstra para ser utilizado en grafos reducidos, lo cual contribuye a disminuir el tiempo de respuesta ante una petición de búsqueda de camino mínimo. En base a los resultados obtenidos se ha arribado a las siguientes conclusiones:

- La búsqueda de caminos mínimos en grafos reducidos con el algoritmo propuesto permite obtener un camino óptimo de igual costo al obtenido en una búsqueda realizada en el grafo sin reducir haciendo uso de algoritmos clásicos (Dijkstra, etc.).
- Con la propuesta realizada se garantiza escalabilidad respecto al tamaño del grafo sobre el que se realiza el análisis.
- Haciendo uso de grafos reducidos se disminuye considerablemente el tiempo de respuesta en la búsqueda de caminos mínimos por lo que se garantiza la eficiencia, lo cual queda evidenciado en los resultados experimentales obtenidos.

V. REFERENCIAS

1. Dijkstra EW. A Note on Two Problems in Connection with Graphs. Numerische Mathematik. 1959;1:269-71. ISSN 0945-3245.
2. Fuhao Z, Jiping L. An Algorithm of Shortest Path Based on Dijkstra for Huge Data. En: Fourth International Conference on Fuzzy Systems and Knowledge Discovery. p. 244-7. ISBN 978-0-7695-3735-1. DOI http://doi.ieeecomputersociety.org/10.1109/FSKD.2009.848.
3. Nazari S, Meybodi MR, Salehigh MA, et al. An Advanced Algorithm for Finding Shortest Path in Car Navigation System. En: First International Conference on Intelligent Networks and Intelligent Systems. p. 671-4. DOI 10.1109/icinis.2008.147.
4. Shao F, Deng W, Zhang B. Traffic Information Management and Promulgating System Based on GIS. En: Optoelectronics and Image Processing (ICOIP), 2010 International Conference on. p. 676-9. DOI 10.1109/icoip.2010.243.
5. Hart PE, Nilsson NJ, Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. Systems Science and Cybernetics, IEEE Transactions on. 1968;4(2):100-7. ISSN 0536-1567. DOI 10.1109/tssc.1968.300136.
6. Noto M, Sato H. A method for the shortest path search by extended Dijkstra algorithm. IEEE International Conference on Systems, Man, and Cybernetics. 2000;2313:2316-20. DOI 10.1109/icsmc.2000.886462.
7. Gutman RJ. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. En: Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics. New Orleans, LA, USA. p. 100-11. Citado 20 de mayo de 2013 Disponible en: http://www.siam.org/meetings/alenex04/abstacts/rgutman1.pdf.

8. Goldberg AV, Harrelson C. Computing the shortest path: A search meets graph theory. In: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms; Vancouver, British Columbia: Society for Industrial and Applied Mathematics; 2005. p. 156-65. Citado 5 de febrero de 2013 Disponible en: http://dl.acm.org/citation.cfm?id=1070432.1070455.
9. Sanders P, Schultes D. Highway hierarchies hasten exact shortest path queries. In: Proceedings of the 13th annual European conference on Algorithms; Palma de Mallorca, Spain: Springer-Verlag; 2005. p. 568-79. ISBN DOI 10.1007/11561071_51.
10. Gonzalez H, Han J, Li X, et al. Adaptive fastest path computation on a road network: a traffic mining approach. In: Proceedings of the 33rd international conference on Very large data bases Vienna, Austria: VLDB Endowment; 2007. p. 794-805. Citado 25 de abril de 2013 Disponible en: http://dl.acm.org/citation.cfm?id=1325851.1325942.
11. Geisberger R, Sanders P, Schultes D, et al. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In: Proceedings of the 7th international conference on Experimental algorithms; Provincetown, MA, USA: Springer-Verlag; 2008. p. 319-33.[Citado 20 de febrero de 2013 Disponible en: http://dl.acm.org/citation.cfm?id=1788888.1788912
12. Efentakis A, Pfoser D, Voisard A. Efficient data management in support of shortest-path computation. In: Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science Chicago, Illinois: ACM; 2011. p. 28-33. DOI 10.1145/2068984.2068990.
13. Delling D, Sanders P, Schultes D, et al. Engineering Route Planning Algorithms. In: Algorithmics of Large and Complex Networks; Springer-Verlag; 2009. p. 117-39. DOI 10.1007/978-3-642-02094-0_7.
14. Sanders P, Schultes D. Engineering fast route planning algorithms. In: Proceedings of the 6th international conference on Experimental algorithms; Rome, Italy: Springer-Verlag; 2007. p. 23-36.
15. Liu YC, Yang DH. A Spatial Restricted Heuristic Algorithm of Shortest Path. In: International Conference on Artificial Intelligence and Computational Intelligence; 2009. p. 36-9. DOI 10.1109/aici.2009.160.
16. Fei S, Wei D, Bing Z. Traffic Information Management and Promulgating System Based on GIS. In: Proceedings of the 2010 International Conference on Optoelectronics and Image Processing; IEEE Computer Society; 2010. p. 676-9. ISBN DOI 10.1109/icoip.2010.243.
17. Qing xiu L, Baoxiang C, Yi wei Z. An improved verification method for workflow model based on Petri net reduction. In: The 2nd IEEE International Conference on Information Management and Engineering (ICIME); 2010. p. 252-6. DOI 10.1109/ICIME.2010.5477436.
18. Sadiq W, Orlowska ME. Analyzing process models using graph reduction techniques. In: Information Systems - The 11th international conference on advanced information systems engineering; 2000. p. 117-34. DOI 10.1016/s0306-4379(00)00012-0.
19. Kai L, Qiang L. An Algorithm Combining Graph-Reduction and Graph-Search for Workflow Graphs Verification. In: 11th International Conference on Computer Supported Cooperative Work in Design; 2007. p. 772-6. ISBN DOI 10.1109/cscwd.2007.4281534.
20. Janssens D, Rozenberg G. Graph grammars with neighbourhood-controlled embedding. Theoretical Computer Science. 1982;21(1):55-74. ISSN 0304-3975.
21. Janssens D, Rozenberg G. On the structure of node-label-controlled graph languages. Information Sciences. 1980;20(3):191-216. ISSN 0020-0255.
22. Rozenberg G, Salomaa A. Handbook of Formal Languages: Springer; 1997. ISBN 9783540614869.
23. Blostein D, Fahmy H, Grbavec A. Issues in the Practical Use of Graph Rewriting. Graph Grammars and Their Application to Computer Science. Berlin Heidelberg: Springer 1996. p. 38-55. DOI 10.1007/3-540-61228-9_78.
24. Hagberg A, Schult D, Swart P. Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th Python in Science Conference; 2008. p.11-5. [Citado 20 de mayo de 2013 Disponible en: http://conference.scipy.org/proceedings/SciPy2008/paper_2/
25. Inda Herrera JA, Rodríguez Puente R, Lazo Cortés M. Sistema para la búsqueda de caminos mínimos en grafos grandes. In: I Taller de Geoninformática; UCIENCIA; 2012.