Algoritmo de reducción de grafos sin pérdida de información

Rafael Rodríguez-Puente y Manuel S. Lazo-Cortés

Universidad de las Ciencias Informáticas, La Habana, Cuba

rafaelrp@uci.cu, manuellazo55@gmail.com

Resumen. Los algoritmos relacionados con la teoría de grafos han sido ampliamente estudiados. Varios estudios se enfocan en la disminución de la complejidad temporal de estos algoritmos. Las técnicas utilizadas en este sentido, generalmente se basan en reducir un grafo o el espacio de búsqueda de solución, eliminando información redundante para el problema específico que se desea resolver. El proceso de reducción de un grafo consiste en obtener grafos más pequeños (con menos vértices) que tengan las características principales o relevantes del grafo original. En el caso de la búsqueda de caminos óptimos, los algoritmos que hacen uso de la reducción de grafos o del espacio de búsqueda de solución, no garantizan la obtención del óptimo en todos los casos. Lo mismo ocurre en otros tipos de problemas tales como la reducción de grafos en redes de *workflow*, de computadoras, etc. En este trabajo se propone un algoritmo de reducción de grafos sin pérdida de información. La propuesta tiene una forma flexible de especificar la manera en que se quiere reducir el grafo; por consiguiente, puede ser utilizada en la solución de varios tipos de problemas, contribuyendo a la obtención de respuestas óptimas en tiempos menores.

Palabras clave. Reducción de grafos, reescritura de grafos, búsqueda de caminos óptimos.

Graph Reduction Algorithm without Loss of Information

Abstract. Algorithms related to graph theory have been studied widely. Several studies deal with the reduction of temporal complexity of these algorithms. The techniques used in this sense are generally based on reducing the graph or the search space of solution. These approaches remove redundant information for a specific kind of problem. The process of reducing a graph is based on obtaining smaller graphs (with fewer vertices) with major or relevant characteristics of the original graph. Algorithms that make use of the graph reduction approach (or reduction of solution search space) for the shortest path search do not guarantee obtaining an optimal path in all cases. The same applies to other types of problems such as graph reduction in workflow networks, computer networks, etc. In this paper we propose a graph reduction algorithm without loss of information. Our method is characterized by a flexible way to specify in what manner in it desirable to reduce a given graph. Therefore, our proposal can be used in solving various types of problems and obtaining optimal responses in less time.

Keywords. Graph reduction, graph rewriting, shortest path search.

1. Introducción

La teoría de grafos ha sido ampliamente estudiada en los últimos años, con énfasis en los algoritmos sobre grafos. Varios de estos algoritmos tienen un elevado costo computacional y algunos son diseñados para resolver problemas NP o NP-Duros. Por consiguiente, varios estudios están relacionados con la reducción de la complejidad computacional de estos algoritmos.

Una de las estrategias que se ha utilizado consiste en reducir el grafo teniendo en cuenta el problema que se está solucionando. Esto se sustenta en el hecho de que el tiempo de ejecución de estos algoritmos, generalmente, depende del número de vértices del grafo en cuestión.

El proceso de reducción de un grafo consiste en obtener grafos más pequeños (con menos vértices) que tengan las características principales o relevantes del grafo original.

En la revisión bibliográfica realizada sobre este tema, se aprecia que varios algoritmos de reducción están enfocados en mantener las características relevantes de grafos que representan redes de carreteras. Esto se realiza

con el objetivo de disminuir los tiempos en la búsqueda de caminos óptimos.

En este sentido se pueden mencionar varios trabajos, por ejemplo, en [7] Gutman propone un algoritmo en el cual se define un atributo de los vértices llamado *reach*, este atributo es una medida de la relevancia del vértice. El atributo *reach* se calcula utilizando el grafo, el mismo contribuye a reducir el tiempo de ejecución en la búsqueda de caminos óptimos reduciendo el espacio de búsqueda de solución.

Una propuesta relevante de algoritmo de reducción, en grafos que representan redes de carreteras, utiliza la jerarquía de la red como criterio de reducción. Varias estrategias utilizan este hecho, por ejemplo, en [18] se propone un algoritmo para construir jerarquías de redes, así como ejecutar consultas sobre la misma. Los autores alcanzan tiempos de ejecución menores y muestran la factibilidad de la propuesta. Otra propuesta introduce un enfoque similar al propuesto por Gutman [1]. Los autores usan nodos relevantes (*transit nodes*) para viajes de larga distancia, pre-calculando los caminos óptimos entre cada par de nodos relevantes y desde cada origen y destino potencial hasta estos nodos relevantes.

En [6] se utiliza la jerarquía de la red de carreteras para particionar la red en áreas y pre-calcular los caminos óptimos en estas áreas. Este enfoque utiliza el hecho de que algunas calles son más transitadas que otras y algunos choferes prefieren viajar por las calles más largas.

En [5] se propone un enfoque que utiliza solo las aristas que relacionan nodos importantes. También se han diseñado algoritmos que imitan el comportamiento humano utilizando la jerarquía de la red de carreteras [14]. Este acercamiento se basa en la idea de que para calcular rutas largas (en redes grandes), solo se necesitan las calles de mayor nivel (autopistas, avenidas, etc.). Esta consideración contribuye a reducir el tiempo de respuesta del algoritmo, sin embargo, no garantiza la obtención del óptimo en todos los casos.

Estos mecanismos pueden ser vistos como algoritmos de reducción de grafos para búsqueda de caminos óptimos y, consecuentemente, para obtener tiempos de respuesta menores.

Por otra parte, la reducción de grafos se ha aplicado en otras áreas, tales como:

- Redes de *workflow*. Un método utilizado para eliminar conflictos estructurales se basa en la reducción de grafos. Este método consiste en identificar ciertas estructuras (e.g., ciclos) y simplificarlas como se muestra en [12, 13, 17]. En [11] se propone un conjunto de reglas de reescritura que permite reducir una red de *workflow*.
- Redes de computadoras. En [2] se introduce un procedimiento denominado *Network Graph Reduction*, en el cual el proceso de reducción se realiza eliminando las aristas congestionadas. En [3, 19, 20] se pueden apreciar varios modelos para reducir grafos. Sin embargo, en [10] se muestra que estos modelos aún carecen del rendimiento que se necesita para una respuesta rápida. Esos autores proponen un algoritmo que elimina aristas que no son *necesarias* para solucionar este problema.

Los mecanismos antes mencionados, eliminan o no consideran información que no es *necesaria* para resolver un problema específico, pero que puede ser indispensable para resolver otros problemas.

En el caso de la búsqueda de caminos óptimos, los algoritmos que hacen uso de la reducción de los grafos o del espacio de búsqueda de solución, no garantizan la obtención del óptimo en todos los casos.

Debido a esta situación, en el presente trabajo se propone un algoritmo de reducción de grafos sin pérdida de información. La propuesta tiene una forma flexible de especificar la forma en que se quiere reducir el grafo; por consiguiente, puede ser utilizada en la solución de varios tipos de problemas, contribuyendo a la obtención de respuestas óptimas en tiempos menores.

El artículo se organiza como sigue: primero se presentan los fundamentos teóricos utilizados en la propuesta de algoritmo de reducción, luego se describe el algoritmo de reducción y se presenta el pseudocódigo del mismo, a continuación se realiza la demostración de la validez del algoritmo de reducción y por último se enuncian las conclusiones.

2. Fundamentos teóricos

Para el desarrollo de este trabajo se utiliza el siguiente marco conceptual.

Definición 1. Un grafo, o grafo no dirigido $G = (V, E)$ se define como un conjunto V finito y no vacío de vértices y un multiconjunto E de aristas, donde cada arista (v_i, v_j) es un par no ordenado de vértices ($v_i, v_j \in V$). En este caso se dice que v_i y v_j son adyacentes. Opcionalmente una arista puede tener asociados un valor que la identifique y una lista de atributos. Cuando los elementos de E tienen multiplicidad uno, el grafo se denomina grafo simple.

Definición 2. Se define un grafo ponderado como una estructura $G = (V, E, f_c)$, donde:

- V representa el conjunto de vértices del grafo.
- E representa el multiconjunto de aristas del grafo.
- La función $f_c : E \rightarrow \mathbb{R}^+$ le hace corresponder a cada arista (v_i, v_j) un valor real positivo denominado costo, cuya interpretación es el costo de ir desde el vértice v_i al vértice v_j.

Definición 3. Una regla de reescritura de grafos sobre un grafo $G = (V, E, f_c)$ es una tupla de la forma $r = (G_i, G_j, \psi_{in}, \psi_{out})$, donde:

- $\mathrm{G_i} = (\{\mathrm{v_i}\}, \{\})$ es un grafo donde $\mathrm{v_i} \in \mathrm{V}$.
- $\mathrm{G_j} = (\mathrm{V_j}, \mathrm{E_j})$, es un grafo.
- ψ_{in} y ψ_{out} son dos conjuntos de información de empotrado, de la forma $(\mathrm{v_m}, \mathrm{c_1}, \mathrm{c_2}, \mathrm{v_n})$, donde: $\mathrm{c_1}, \mathrm{c_2} \in \mathbb{R}^+, \mathrm{v_m} \in \mathrm{V_j}, \mathrm{v_n} \in (\mathrm{V} - \mathrm{V_j})$. En el caso de ψ_{in}, $\exists(\mathrm{v_n}, \mathrm{v_i}) \in \mathrm{E}$ tal que $\mathrm{f_c}(\mathrm{v_n}, \mathrm{v_i}) = \mathrm{c_1}$, luego de aplicar la regla de reescritura, se obtiene el grafo $\mathrm{G_1} = (\mathrm{V_1}, \mathrm{E_1}, \mathrm{f_{c1}})$ y se cumple que $\exists(\mathrm{v_n}, \mathrm{v_m}) \in \mathrm{E_1}$ tal que $\mathrm{f_{c1}}(\mathrm{v_n}, \mathrm{v_m}) = \mathrm{c_2}$. 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 post-empotradas.

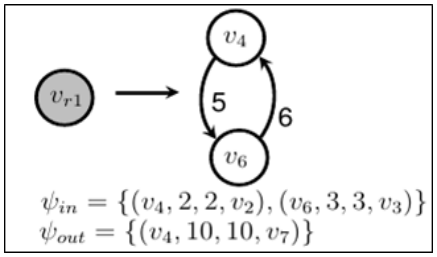


Fig. 1. Ejemplo de regla de reescritura. En la parte izquierda se muestra el grafo $G_i = (\{v_{r1}\}, \{\})$, en la parte derecha se muestra el grafo G_j y en la parte inferior se muestra la información de empotrado ψ_{in} y ψ_{out}.

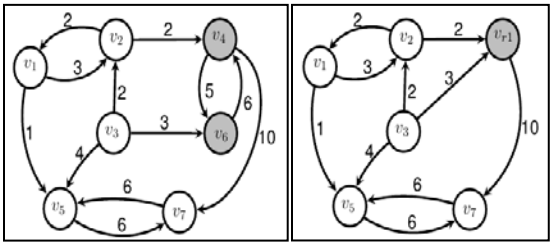
El conjunto ψ_{in} permite transformar el conjunto de aristas preempotradas que inciden en el vértice v_i en aristas postempotradas que inciden en uno o más vértices $v_j \in V_j$, de forma similar ψ_{out} permite transformar las aristas pre-empotradas que salen de v_i en aristas postempotradas que salen 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. Se puede comprobar, que luego de aplicar dicha regla al grafo de la Figura 2(b) según el mecanismo NCE [9], se obtiene el grafo de la Figura 2(a).

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.

Definición 4. Un grafo reducido es una tupla $G = (V_r, E_r, f, R)$, donde:

- *V_r es un conjunto de vértices.*
- *E_r es un conjunto de aristas.*
- *$f: V_r \times V_r \times V_r \rightarrow \mathbb{R}^+ \cup \{0, \infty\}$, es una función que para cada (v_i, v_j, v_k) retorna el costo de ir desde v_i hasta v_k pasando por v_j, siendo v_k adyacente a v_j y v_j adyacente a v_i. La función f obviamente se define para los*



a) Ejemplo de grafo b) Grafo reducido

Fig. 2. Ejemplos de grafo

casos en que $v_i = v_j$ y/o $v_j = v_k$. En el caso trivial $f(v,v,v) = 0$.

- *R es un conjunto de reglas de reescritura sobre (V_r, E_r, f_c), donde f_c se define como $f_c(v,w) = f(v,v,w)$.*

Definición 5. Sea $G_r = (V_r, E_r, f, R)$ un grafo reducido. Se dice que G_r es un grafo reducido a partir del grafo G si al aplicar las reglas de reescritura especificadas en R al grafo G_r se obtiene el grafo G.

Definición 6. 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. En otro caso se dice que v_i es exterior.

El término grande es difuso por naturaleza, no obstante, y debido al uso de este adjetivo en varias situaciones, a continuación se muestra un esbozo de una definición formal de este término, que puede ser utilizado en la definición de algoritmos sobre grafos.

Sean:

- RNF_t, variable que representa el requisito no funcional asociado al tiempo de respuesta en la solución de un problema P.
- t, variable que representa el tiempo empírico promedio de respuesta de un algoritmo A sobre un grafo G que resuelve el problema P.
- hardware, variable que representa las características del hardware del que se dispone.
- software, variable que representa el algoritmo A sobre un grafo G para la solución del problema P.

Se puede definir una función T, que a la entrada $(hardware, software, RNF_t)$ retorna el valor t definido como tiempo empírico promedio de respuesta, o sea,

$T(hardware, software, RNF_t) = t.$

Para variar este tiempo y modificar la característica de grafo grande, se puede modificar el hardware o el software. También se puede modificar la variable RNF_t en dependencia del problema que se desee resolver.

3. Algoritmo de reducción de grafos

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

El algoritmo de reducción diseñado reduce 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 original. Además, el hecho de que no exista pérdida de información hace posible su uso en la reducción de varios tipos de grafos.

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 realizar determinados análisis, por ejemplo el de búsqueda de caminos óptimos, y obtener resultados iguales a los obtenidos en el grafo original, es necesario que en el grafo reducido no existan pares de vértices que sean adyacentes y a su vez reducidos. Para contribuir a garantizar tal propósito, se introdujo la definición 6.

El primer paso del algoritmo de reducción consiste en refinar la partición que es entrada de dicho algoritmo. Para ello se sigue la siguiente estrategia:

- Dos vértices están en la misma clase de la partición refinada si y solo si:
  - Están en la misma clase de P.
  - Son vértices interiores.

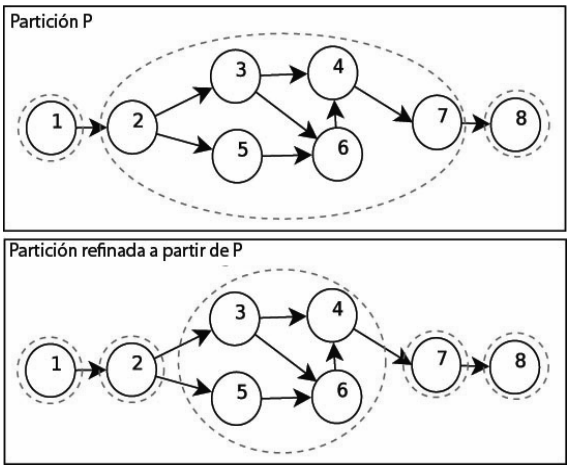


Fig. 3. Ejemplo de refinamiento de partición utilizando los conceptos vértice interior y exterior. En el grafo de la parte superior se puede apreciar que los vértices 2 y 7 son exteriores; por cada uno de dichos vértices se crea una clase en la nueva partición

- Si un vértice es exterior, se crea una nueva clase que contiene solo a dicho vértice.

En la Figura 3 se ilustra con un ejemplo el uso de la definición de vértice interior y exterior para refinar la partición que es entrada del algoritmo.

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.

Algoritmo 1: $ConstruirVerticesReducidos$
Entrada: Una partición refinada $P = \{A_1, A_2, \dots, A_s\}$.
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

Además, se propone el uso de la función ObtenerNombre (ver paso 3 del Algoritmo 1) que a partir de una clase de la partición, devuelve un identificador que 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. El pseudocódigo para la creación de los vértices del grafo reducido se muestra en el Algoritmo 1.

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; 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 f_r del grafo reducido G_r que se devolverá como salida del algoritmo de reducción. El pseudocódigo para la creación de las aristas del grafo reducido se muestra en el Algoritmo 2.

Algoritmo 2: $ConstruirAristas$
Entrada: Una partición $P = \{A_1, A_2, \dots, A_s\}$ 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

Para construir las reglas de reescritura del grafo reducido se parte de la definición 3. Se crea una regla por cada vértice reducido (o por cada clase de la partición que tenga cardinalidad mayor que uno).

En primer lugar, 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. Para crear la información de empotrado, 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} con la información de la misma (ver Definición 3) 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} que representa la arista en cuestión.

Algoritmo 3: $ConstruirRr$

Entrada: Una partición $P = (A_1, A_2, \dots, A_s)$ 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. $Adicionar(\psi_{in}, (v, f(v, v, v_k), f(v, v, v_k), v_k))$
21. **Fin Para**
22. $ady = AristasQueSalen(v_k, G, [v_k] - V_j)$
23. **Para todo** $v \in ady$ **hacer**
24. $Adicionar(\psi_{out}, (v, f(v_k, v_k, v), f(v_k, v_k, v), v_k))$
25. **Fin Para**
26. **Fin Para**
27. $Adicionar(R, (G_i, G_j, \psi_{in}, \psi_{out}))$
28. **Fin Para**
29. **Retornar** R

La construcción de las reglas de reescritura es un paso esencial en el algoritmo de reducción, es lo 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. El pseudocódigo para este paso se muestra en el Algoritmo 3.

El último paso consiste en calcular la función f 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 de v_j. Además, dicha función puede ser vista como el mecanismo para almacenar el costo de pasar por un vértice reducido.

Para calcular la función f, 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 puedan 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 en pasos posteriores.

Sobre el grafo auxiliar creado se aplica el algoritmo de Dijkstra [4]**,** 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 f.

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 pre-calculado se reduce el tiempo necesario para mostrar el camino óptimo.

En caso de ser necesario, esta función se puede modificar de forma tal que almacene los datos que sean necesarios para la solución de un problema particular. El pseudocódigo para calcular la función f se muestra en el Algoritmo 4.

Algoritmo 4: Cálculo de la función f

Entrada: Un grafo 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 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. $verticesAdyacentes = \{\}$
3. **Para todo** $v_i, v_j \in A_{aux}, v_j \in Ayacentes(v_i, G)$ **hacer**
4. **Si** $v_j \notin A_i$, **entonces**
5. $AdicionarVertice(G_{aux}, v_j)$
6. $AdicionarArista(G_{aux}, vi, v_j)$
7. $f_{aux}(v_j, v_j, v_i) = f_r(v_j, v_j, v_i)$
8. $f_{aux}(v_i, v_i, v_j) = f_r(v_i, v_i, v_j)$
9. **Fin Para**
10. **Para todo** $v_o \in verticesAdyacentes$ **hacer**
11. $dijkstra(v_o, G_{aux})$
12. **Para todo** $v_d \in verticesAdyacentes, v_o \neq v_d$ **hacer**
13. $f(v_o, v_d, v_i) = (D[v_d], camino(v_o, v_d, P_r))$
14. **Fin Para**
15. **Fin Para**
16. **Retornar** f

Finalmente, se crea un grafo reducido utilizando los conjuntos de vértices, aristas, reglas de reescritura y la función f. El pseudocódigo para reducir un grafo se muestra en el Algoritmo 5.

Algoritmo 5: $ReducirGrafo$

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. $Calcular f(G, R_r[A_i].G_i, R_r[A_i].G_j, f_r)$ {Calcular el costo mínimo de pasar por el vértice reducido correspondiente a la clase A_i de P (ver Algoritmo 4). 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

El algoritmo de reducción de grafos enunciado en este trabajo 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 propuesto permite realizar análisis de redes en función de una determinada escala. Por ejemplo, si el grafo que representa la red de carreteras de un país, provincia o estado, se reduce agrupando todos los vértices que están en un mismo municipio o distrito, el grafo reducido resultante representa el mapa a escala de municipio o distrito; 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.

En la Figura 4 se muestra un ejemplo de grafo reducido. Note que entre cada par de vértices reducidos existen al menos dos vértices que no son considerados reducidos.

Una versión anterior del algoritmo de reducción presentado aquí se aplicó en la ejecución del Método de los Grafos Dicromáticos [16], utilizado en el ámbito del Diseño Racional en la Ingeniería Mecánica; además se utilizó para dividir un modelo en sub-modelos de forma tal que se facilite el análisis de modelos complejos a los ingenieros mecánicos.

También se puede afirmar que el algoritmo de reducción propuesto es relevante para la búsqueda de caminos óptimos cuando los grafos son grandes [15].

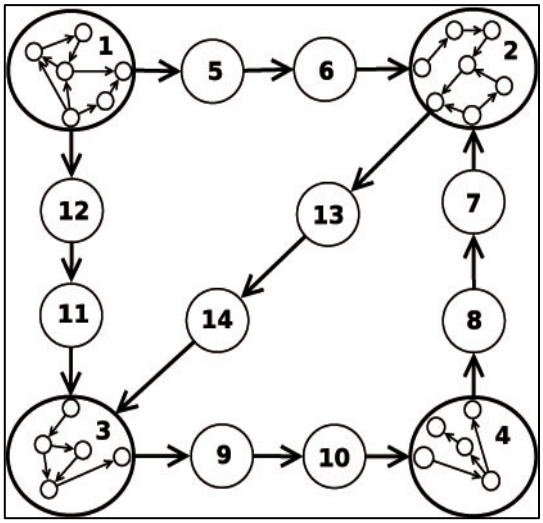


Fig. 4. Ejemplo de grafo reducido

3.1. Demostración de la validez del algoritmo de reducción de grafos propuesto

A continuación se demuestra que el algoritmo de reducción no presenta pérdida de información, o lo que es lo mismo, que el proceso de reducción es reversible, por lo que a partir del grafo reducido se puede obtener el grafo original a partir del cual se redujo.

Para realizar la demostración haciendo uso de la tripleta de Hoare [8], se definen las siguientes precondiciones e invariantes de ciclo. Los pasos fundamentales en el proceso de reducción para garantizar que no exista pérdida de información, son los relacionados con la creación de las reglas de reescritura; por lo que se demostrarán las invariantes de ciclo asociadas a la creación de dichas reglas.

Precondiciones:

- $G = (V, E, f, \{\})$ es un grafo.
- $P = \{A_1, A_2, \dots, A_s\}$ es una partición sobre el conjunto de vértices V.
- $G_i = (\{v_i\}, \{\})$ es el grafo de la parte izquierda de la regla de reescritura r_i asociada a la clase $A_i \in P, |A_i| > 1$.
- $G_j = (V_j, E_j, f_j, R_j)$ grafo de la parte derecha de la regla de reescritura r_i asociada a la clase $A_i \in P, |A_i| > 1$.

Invariantes de ciclo:

1. Sea $v_j \in V_j,\ \forall v_k \in ([v_j] - V_j)$ se cumple que $\exists(v_k, v_j) \in E \rightarrow (v_k, f_c(v_k, v_j), f_c(v_k, v_j), v_j) \in \psi_{in}$. Para todos los vértices interiores (V_j), se debe crear información de empotrado que contenga las aristas que inciden en los mismos desde los vértices exteriores que pertenecen a la misma clase de equivalencia ($[v_j] - V_j$). Note que $[v_j]$ denota la clase en P que contiene a v_j.
2. Sea $v_j \in V_j,\ \forall v_k \in ([v_j] - V_j)$ se cumple que $\exists(v_j, v_k) \in E \rightarrow (v_k, f_c(v_k, v_j), f_c(v_k, v_j), v_j) \in \psi_{out}$. Para todos los vértices interiores (V_j), se debe crear información de empotrado que contenga las aristas que inciden en los vértices exteriores de la misma clase de equivalencia ($[v_j] - V_j$) y salen de los vértices del grafo G_j.

La poscondición a comprobar, es que G_r es un grafo reducido a partir de G. Dicho de otra forma, que a partir del grafo reducido G_r, se puede obtener el grafo original G.

Lema 1. Sea $v_j \in V_j,\ \forall v_k \in ([v_j] - V_j)$, tal que $\exists(v_k, v_j) \in E$, se tiene que
$(v_k, f(v_k, v_k, v_j), f(v_k, v_k, v_j), v_j) \in \psi_{in_k}$.

Demostración: (Por inducción sobre k)
Caso base $k = 0, \psi_{in_0} = \{\}$
Para $k + 1$:
$\psi_{in_{k+1}} =$
$\psi_{in_k} \cup \{(v_{k+1}, f(v_{k+1}, v_{k+1}, v_j), f(v_{k+1}, v_{k+1}, v_j), v_j)\}$
(ver paso 20 del Algoritmo 3) ■

Lema 2. Sea $v_j \in V_j,\ \forall v_k \in ([v_j] - V_j)$, tal que $\exists(v_j, v_k) \in E)$, se tiene que
$(v_k, f_c(v_k, v_j), f_c(v_k, v_j), v_j) \in \psi_{out}$.
La demostración de este lema es similar a la del Lema 1.

Teorema 1. Sea $G_r = (V_r, E_r, f_r, R_r)$ un grafo reducido a partir del grafo $G = (V, E, f, R)$ y la partición P, si se aplica la regla de reescritura $r_i = (\{\{v_i\}, \phi), (V_j, E_j, f_j, R_j), \psi_{in}, \psi_{out}), r_i \in R_r$, asociada al vértice $v \in V_r$ sobre el grafo G_r, se obtiene un grafo $G_r' = (V_r', E_r', f_r', R_r')$ y se cumple lo siguiente:
Sea $v_i \in V_r,\ \forall v_k \in V_j$, tal que v_i y v_k pertenecen a la misma clase de P, $(v_i, v_k) \in E \rightarrow (v_i, v_k) \in E_r'$.

Demostración: Sea $(v_i, v_k) \in E$, tal que $v_i, v_k \in [v]$, supongamos que $(v_i, v_k) \notin E_r'$ luego de aplicar la regla r_i asociada al vértice reducido v. Note que si existe una arista entre un vértice de un grafo G_j y uno del grafo G_r, dichos vértices pertenecen a la misma clase de equivalencia.
Cuando se aplica una regla r_i se sustituye el grafo $G_i = (\{v_i\}, \{\})$ por el grafo G_j y se conecta G_j con $(G - G_i)$ según se especifica en ψ_{in} y ψ_{out}.
Sea $v_j \in V_j, \forall v_k \in ([v_j] - V_j)$, note que $[v_k \in ([v_j] - V_j)] \rightarrow v_k \in (V_r - \{v_i\})$:

- si $\exists(v_k, v_j) \in E$, entonces
$(v_j, f(v_k, v_k, v_j), f(v_k, v_k, v_j), v_k) \in \psi_{in}$, por el Lema 1.

- si $\exists(v_j, v_k) \in E$, entonces $(v_j, f(v_j, v_j, v_k), f(v_j, v_j, v_k), v_k) \in \psi_{out}$, por el Lema 2.

Lo cual contradice nuestra suposición, por lo que se tiene que $(v_i, v_k) \in E_r'$.■

Corolario 1. Sea $G_r = (V_r, E_r, f_r, R_r)$ un grafo reducido a partir del grafo $G = (V, E, f, R)$ y la partición P, si se aplica el conjunto de reglas de reescritura R_r, asociadas a los vértices reducidos del grafo G_r, se obtiene el grafo $G = (V, E, f, R)$ a partir del cual se redujo G_r.

De esta forma queda demostrado, que el algoritmo de reducción de grafos propuesto garantiza que no exista pérdida de información.

4. Conclusiones

El algoritmo de reducción de grafos aquí presentado garantiza que no se pierde información en el proceso de reducción, para ello se hace uso de un mecanismo de reescritura de grafos. Además, se puede utilizar para resolver el problema de la búsqueda de caminos óptimos.

Haciendo uso de grafos reducidos se puede disminuir considerablemente el tiempo de respuesta de los algoritmos sobre grafos, garantizando así la eficiencia. Además, con la reducción, se garantiza escalabilidad respecto al tamaño del grafo sobre el que se realiza el análisis.

La generalidad del algoritmo de reducción propuesto radica en la posibilidad de su aplicación en distintos tipos de redes, así como en el Diseño Racional en Ingeniería Mecánica.

Referencias

1. **Bast, H., Funke, S., Sanders, P., & Schultes, D. (2007).** Fast Routing in Road Networks with Transit Nodes. *Science,* 316(5824), 566–566.
2. **Casetti, C., Lo Cigno, R., Mellia, M., Munafo, M., & Zsóka, Z. (2003).** A new class of QoS routing strategies based on network graph reduction. *Computer Networks*, 41(4), 475–487.
3. **Cong, J., Fang, J., & Khoo, K.Y. (1999).** An implicit connection graph maze routing algorithm for ECO routing. *1999 IEEE/ACM International Conference on Computer-Aided Design,* San Jose, CA, USA, 163–167.
4. **Dijkstra, E.W. (1959).** A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1), 269–271.
5. **Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008).** Contraction hierarchies: Faster and simpler hierarchical routing in road networks. *7th International Conference on Experimental Algorithms (WEA'08),* Massachusetts, USA, 319–333.
6. **Gonzalez, H., Han, J., Li, X., Myslinska, M., & Sondag, J.P. (2007).** Adaptive fastest path computation on a road network: a traffic mining approach. *33rd International Conference on Very Large Data Bases (VLDB'07),* Vienna, Austria, 794–805.
7. **Gutman, R.J. (2004).** Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics,* New Orleans, LA, USA*,* 100–111.
8. **Hoare, C.A.R. (1969).** An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 576–580.
9. **Janssens, D. & Rozenberg, G. (1982).** Graph grammars with neighbourhood-controlled embedding. *Theoretical Computer Science*, 21(1), 55–74.
10. **Li, Y.L., Li, J.Y., & Chen, W.B. (2007).** An efficient tile-based ECO router using routing graph reduction and enhanced global routing flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2), 345–358.
11. **Lin, H., Zhao, Z., Li, H., & Chen, Z. (2002).** A novel graph reduction algorithm to identify structural conflicts. *35th Annual Hawaii International Conference on System Sciences,* Big Island, Hawaii, 1–10.
12. **Liu, Q., Cao, B., & Zhao, Y. (2010).** An improved verification method for workflow model based on Petri net reduction. *2nd IEEE International Conference on Information Management and Engineering (ICIME),* Chengdu, China, 252–256.
13. **Lu, K. & Liu, Q. (2007).** An algorithm combining graph-reduction and graph-search for workflow graphs verification. *11th International Conference on Computer Supported Cooperative Work in Design,* Melbourne, Australia*,* 772–776.
14. **Pfoser, D., Efentakis, A., Voisard, A., & Wenk, C. (2009).** *Exploiting road network properties in*

efficient shortest path computation (TR-09-007). Berkeley, CA, USA: International Computer Science Institute.

15. **Rodríguez-Puente, R. & Lazo-Cortés, M.S. (2013).** Algorithm for shortest path search in Geographic Information Systems by using reduced graphs. SpringerPlus, 2, Art. No.291.
16. **Rodríguez-Puente, R., Marrero-Osorio, S.A., & Lazo-Cortés, M.S. (2012).** Aplicación de un algoritmo de reducción de grafos al Método de los Grafos Dicromáticos. *Ingeniería Mecánica,* 15(2),158–169.
17. **Sadiq, W. & Orlowska, M.E. (2000).** Analyzing process models using graph reduction techniques. *Information systems*, 25(2), 117–134.
18. **Sanders, P. & Schultes, D. (2005).** Highway hierarchies hasten exact shortest path queries. *Algorithms–Esa 2005, Lecture Notes in Computer Science,* 568–579.
19. **Xing, Z. & Kao, R. (2002).** Shortest path search using tiles and piecewise linear cost propagation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2), 145–158.
20. **Zheng, S.Q., Lim, J.S., & Iyengar, S.S. (1996).** Finding obstacle-avoiding shortest paths using implicit connection graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 15(1), 103–110.

Rafael Rodríguez Puente es Doctor en Ciencias Técnicas por la Universidad de las Ciencias Informáticas (UCI, 2012), actualmente es Decano de la Facultad 3 de la UCI. Ha impartido diversas asignaturas entre las que se encuentran Programación, Estructuras de Datos, Técnicas de Compilación, Patrones de diseño, Máquinas Computadoras, Bases de datos y Metodología de la Investigación Científica. Sus áreas de interés son: algoritmos sobre grafos, Sistemas de Información Geográfica, bases de datos de grafos e hipergrafos.

Manuel S. Lazo Cortés es Doctor en Ciencias Matemáticas por la UCLV (1994). Sus áreas de interés son el reconocimiento de patrones (RP) con enfoque lógico combinatorio, selección de rasgos y clasificación no supervisada. Es autor de varios artículos publicados en revistas especializadas, editor y revisor de memorias de congresos internacionales en el área de RP. Es miembro del Consejo Asesor de Ciencia e Innovación Tecnológica del Ministerio de la Informática y las Comunicaciones de la República de Cuba.

Artículo recibido el 13/04/2013, aceptado el 17/07/2013.