Algoritmos de corte mínimo en grafos

Franss Cruz Ordoñez
Universidad Nacional de Ingeniería
fransscruz18@gmail.com

Davis García Fernandez
Universidad Nacional de Ingeniería
yums123@hotmail.com

Fernando Macuri Orellana
Universidad Nacional de Ingeniería
rfmo2014@hotmail.com

***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.**

***Index Terms*—Algoritmo de reducción,iteración,optimización, reducción de grafos,nodos.**

I. Introducción

De los más diversos usos de los grafos (análisis de circuitos, análisis y planificación de proyectos, genética, lingüística, ciencias sociales, robótica ...), una de ellas es la optimización de itinerarios. Así, una red de carreteras que une varias ciudades puede ser representada a través de un grafo. A través de la conjugación del uso de un grafo y del Algoritmo de corte mínimo es posible calcular la ruta más corta para realizar determinada ruta. De este modo, se pretende con este artículo describir el funcionamiento de dichos algoritmos, permitiendo incluso su aplicación a nivel informático, haciendo notar algunas ventajas y desventajas entre estos.

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 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.

Fundamento Teórico.

Para el presente trabajo usaremos las siguientes definiciones:

- **Definición 1:**Un grafo es un par ordenado (V, E), donde V es algún conjunto no vacío y E es un conjunto de subconjuntos de dos puntos de V.Los elementos del conjunto V se llaman vértices o **nodos** del grafo G y los elementos de E se llaman ramas de G.

- **Definición 2:**Un grafo no dirigido $G = (V, E)$ se define como un conjunto V no vacío de vértices y el conjunto E de aristas, donde cada arista (v_i, vj) 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 3:** Sea $G = (V, E)$ un grafo ponderado finito tal que $V = \{v_1, \ldots, v_n\}$. Llamaremos matriz de peso del grafo G a la siguiente matriz de orden n x n:

$$W = [a_{ij}]/a_{ij} = \begin{cases} w_{ij} & si \quad (V_i, V_j) \in A \\ \infty & si \quad (V_i, V_j) \notin A \end{cases}$$

  Es decir, que pondremos el valor del peso cuando lo tenga, y el símbolo infinito cuando no exista tal valor. En un grafo ponderado llamamos peso de un camino a la suma de los pesos de las aristas (o arcos) que lo forman. En un grafo ponderado llamamos camino más corto entre dos vértices dados al camino de peso mínimo entre dichos vértices.
- **Definición 4:** Un grafo reducido es una tupla (V_r, E_r, f, R), donde:
  - V_r es un conjunto de vértices.
  - E_r es un conjunto de aristas.
  - $f : V_r \times V_r \times V_r \to \mathbb{R}_+ \cup \{0, \infty\}$ 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 casos en que $V_i = V_J$ y/o $V_j = V_i$. 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 6:** 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:
  - $G_i = (v_i, \{\})$ es un grafo donde $v_i \in V$.
  - $G_j = (V_j, E_j)$, es un grafo.
  - $\psi_{in} \quad y \quad \psi_{out}$ son dos conjuntos de información

de empotrado, de la forma (v_m, c_1, c_2, v_n), donde: $c_1, c_2 \in \mathbb{R}^+, v_m \in v_j, v_n \in (V - V_j)$. En el caso de ψ_{in}, $\exists(v_n, v_1) \in E$ tal que $(v_n, v_i) = c_1$, luego de aplicar la regla de reescritura, se obtiene el grafo $G_1 = (v_1, E_1, f_{c_1})$ y se cumple que $\exists(v_n, v_m) \in E_1$ tal que $f_{c_1}(v_n, v_m) = c_2$.
Análogamente a ψ_{in} ,se define ψ_{out}, con la única diferencia de la orientación de las aristas.

- **Definición 7:** Una iteración significa repetir varias veces un proceso con la intención de alcanzar una meta deseada, objetivo o resultado. En programación, iteración es la repetición de un segmento de código dentro de un programa de computadora.
- **Definición 8:** Un algoritmo es una secuencia de pasos a seguir para resolver un problema en cuestión.
- **Definición 9:** El algoritmo de reducción se encarga de 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 original. Además, el hecho de que no exista pérdida de información hace posible su uso en la reducción de varios tipos de grafos.

Algoritmos a utilizar:

- **Algoritmo de Dijkstra:** Resuelve el problema de los caminos más cortos desde un único vértice origen hasta todos los otros vértices del grafo.
- **Algoritmo de Bellman - Ford:** Resuelve el problema de los caminos más cortos desde un origen si la ponderación de las aristas es negativa.
- **Algoritmo de Búsqueda A*:** Resuelve el problema de los caminos más cortos entre un par de vértices usando la heurística para intentar agilizar la búsqueda.
- **Algoritmo de Floyd - Warshall:** Resuelve el problema de los caminos más cortos entre todos los vértices.

Objetivo:

Utilizar el Algoritmo de Dijkstra y Bellman-Ford en las siguientes aplicaciones a la vida cotidiana:

1. Aplicaciones del algoritmo de Bellman-Ford
   - El Problema del Barco Mercante
   - El Horario del Operador Telefonico
2. Aplicaciones del Algoritmo de Dijkstra
   - En la Economía
   - En la Estadística

II. Estado del arte

1. Artículos científicos:

a) Dijkstra EW. A Note on Two Problems in Connection with Graphs. Numerische Mathematik.

b) 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.

c) 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.

d) Hart PE, Nilsson NJ, Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. Systems Science and Cybernetics, IEEE Transactions on.

e) Noto M, Sato H. A method for the shortest path search by extended Dijkstra algorithm. IEEE International Conference on Systems, Man, and Cybernetics.

f) 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.

g) 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.

h) 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.

2. Algunas menciones:

a) Este artículo presenta el algoritmo de Dijkstra clásico en detalle e ilustra el método de implementación del algoritmo y las desventajas del algoritmo: los nodos de red requieren memoria de clase cuadrada, por lo que es difícil cuantificar la ruta más corta de los nodos principales. Al mismo tiempo, describe el algoritmo de nodo adyacente que es un algoritmo de optimización basado en el algoritmo de Dijkstra.

b) El algoritmo aprovecha al máximo la relación de conexión de arcos en la información de topología de red, y evita el uso de una matriz de correlación que contiene un valor infinito sustancial, por lo que es un análisis más adecuado de la red para datos masivos. Está demostrado que el algoritmo puede ahorrar mucha memoria y es más adecuado para la red con enormes nodos.

c) Proponemos algoritmos de ruta más cortos que utilizan la búsqueda A * en combinación con una nueva técnica de límite inferior teórico de gráficos basada en puntos de referencia y la desigualdad de triángulos. Nuestros algoritmos calculan caminos óptimos más cortos y trabajan en cualquier gráfico dirigido. Proporcionamos resultados experimentales que demuestran que el algoritmo más eficiente supera los algoritmos previos, en particular la búsqueda A * con límites euclidianos, por un amplio margen en redes de carreteras y en algunas familias de problemas sintéticos.

d) Presentamos una técnica de planificación de ruta basada únicamente en el concepto de contracción del nodo. Los nodos primero se ordenan por 'importancia'. Luego se genera una jerarquía al contratar iterativamente el nodo menos importante. Contratar un nodo significa reemplazar los caminos más cortos que pasan por v por atajos. Obtenemos un algoritmo de consulta jerárquica utilizando búsqueda bidireccional de ruta más corta.

III. Diseño del Experimento

1. Algoritmo de Dijkstra:

Trataremos de explicar brevemente el algoritmo de Dijkstra y del uso en el lenguaje R:
Dado un grafo conexo, dirigido o no dirigido, con distancias no negativas; el algoritmo de Dijkstra empieza por marcar el nodo fuente e inicializar a 0 o a infinito las distancias acumuladas de cada nodo. Posteriormente itera, mientras queden nodos sin marcar, seleccionando en cada etapa el arco con la suma más pequeña de la distancia acumulada por uno de los nodos marcados más la distancia del arco que lo une a uno de los nodos no marcados, guardando este último nodo como marcado y actualizando su distancia acumulada con la suma antes calculada. Al final del proceso tendremos un árbol de expansión de camino más corto. La implementación del algoritmo tal y como está escrita en el **pseudocódigo** no es casual y facilita portarlo a R. Existe varias implementaciones del algoritmo al margen de la original de Dijkstra. De hecho, la que utilizaremos en el paquete **optrees** modifica un poco su funcionamiento para adaptarlo a nuestras necesidades. Por otra parte, es necesario, eso sí, indicar si el grafo que introducimos en la función es o no dirigido, pues en este segundo caso deberemos realizar el paso previo de duplicar los arcos. El resto de datos del grafo de entrada lo constituyen el conjunto de nodos y arcos, los cuales se introducen en R de la misma forma que hemos visto hasta ahora, por lo demás, las instrucciones del algoritmo de Dijkstra se pueden trasladar directamente manteniendo el uso de un único bucle, lo que permite evitar aumentar la complejidad del problema. La propuesta original de Dijkstra funciona en un tiempo computacional teórico de $O(|V|^2)$. Con el tiempo se han desarrollado mejores implementaciones que requieren una cola de prioridad y reducen la complejidad del algoritmo, pero complican portal el código a R. En nuestro caso, aunque con diferencias, mantenemos un formato muy similar al modelo original de Dijkstra, que es de por sí lo suficientemente rápido como para obtener resultados por debajo del segundo en grafos con cientos de nodos y miles de arcos.

1.1 Optrees Este paquete en R, encuentra árboles óptimos ponderados. En particular, este paquete proporciona herramienta de resolución de problemas de árbol de expansión de costo mínimo, problemas de arborescencia de costo mínimo, problemas de árbol de ruta más corto y problema de árbol de corte mínimo.

A continuación te mostramos el Pseudocódigo:

Algorithm 1 Dijkstra
Entrada: $G = (V, A); D, s \in V$
Salida: $G^T = (V^T, A^T); W$

$A^T \leftarrow \emptyset$
$V^T \leftarrow \{s\}$
$W_{1,|V|} \leftarrow 0$
while $|V^T| < |V|$ **do**
    seleccionar arcos $(i, j) \in A$ con $i \in V^T$ y $j \in VV^T$
    elegir un arco (i, j) de los anteriores tal que $d_{ij} + W_i$ sea mínimo
    $A^T \leftarrow A^T \cup \{(i, j)\}$
    $V^T \leftarrow V^T \cup \{j\}$
    $W_j \leftarrow d_{ij} + W_j$
end while

2. Algoritmo de Bellman-Ford:

Este segundo algoritmo, para el problema del árbol de camino más corto es el conocido como algoritmo de Bellman-Floyd. El algoritmo empieza inicializando a 0 la distancia asociada a la fuente y a infinito las distancias acumuladas del resto de nodos. Posteriormente, lleva a cabo el proceso de relajación iterando en todos los nodos distintos de la fuente, comprobando en cada arco que sale de él si la distancia hacia otro nodo es mayor que la suma de la distancia acumulada del primero más el peso del arco que los une, en caso afirmativo establece este resultado como la distancia acumulada por el nodo de llegada y guarda el nodo de partida como su predecesor. Al final termina comprobando si se han encontrado ciclos negativos, revisando arco por arco si se puede reducir la distancia acumulada del nodo de llegada. Si no se topa con uno, el resultado final es un árbol de expansión del camino más corto. El proceso de programar el algoritmo de Bellman-Floyd en R es equivalente al pseudocódigo aquí desarrollado, aunque con ligeros añadidos. En este caso, también es necesario introducir un parámetro de entrada que permita señalar si estamos ante un grafo dirigido o no, para duplicar los arcos ante eseta segunda situación, pero al final es necesario incorporar un bucle adicional que reconstruya el árbol del camino más corto a partir del conjunto de nodos y los predecesores que hemos ido almacenando durante el proceso de relajación. Este último bucle ralentiza un poco nuestra función, tiene una complejidad de $O(|V||A|)$. Es, por tanto, más lento que el de Dijkstra, siendo este el precio a pagar por permitir trabajar con pesos negativos.

A continuación te mostramos el Pseudocódigo:

Algorithm 2 Bellman-Ford
Entrada: $G = (V, A); D; s \in V$
Salida: $G^T = (V^T, A^T); W$

$A^T \leftarrow \emptyset$
$W_{1:|V|} \leftarrow \infty; W_s \leftarrow 0$
$P \leftarrow \emptyset$
for cada nodo $i \in V\{s\}$ **do**
    for cada arco $(i, j) \in A$ **do**
        if $W_j > W_i + d_{ij}$ **then**
            $W_j \leftarrow W_i + d_{ij}$
            $P_j \leftarrow i$
        end if
    end for
end for
for cada arco $(i, j) \in A$ **do**
    if $W_j > W_i + d_{ij}$ **then**
        No hay solución
    end if
end for
for cada nodo $j \in V\{s\}$ **do**
    $i \leftarrow P_j$
    seleccionar arco (i, j) de A
    $A^T \leftarrow A^T \cup \{(i, j)\}$
end for
$V^T \leftarrow V$

3. Algoritmo de Gusfield:

La principal ventaja del algoritmo de Gusfield es, precisamente, evitar el requisito de comprimir nodos. El algoritmo arranca construyendo un arbol con el primer nodo como nodo único, e itera posteriormente añadiendo cada vez uno nuevo de acuerdo al orden $2, 3, ..., |V|$. En cada iteración la duda radica en escoger a qué nodo del árbol se une cada nuevo nodo k cuando existe más de una posibilidad.

Para determinarlo repetimos el siguiente proceso:

- Buscamos el arco (i, j) de menor peso dentro del árbol. Este arco tendrá el valor del corte mínimo entre los nodos i y j en el grafo original.
- Borramos dicho arco del árbol formando dos componentes, una con el nodo $i(T_1)$ y otra con el nodo $j(T_2)$. Con ellos sabremos también los nodos que pertenecen a la misma componente que $i(S_1)$ y los que pertenecen a la musma componente que $j(S_2)$ en el corte $i - j$ del grafo original.
- Si el nodo k pertenece a la componente S_1 en el grafo original, entonces fijamos como nuevo árbol a comprobar la componente T_1 del árbol. En caso contrario fijamos como nuevo árbol a comprobar la componente T_2 del árbol.
- Repetimos los pasos anteriores hasta que nos quedemos conn un árbol con un único nodo.

Este será el nodo del árbol al que tenemos que unir el nuevo nodo k. El peso del arco que los una será igual al corte mínimo entre los dos nodos en el grafo original.

Todo este proceso se realiza hasta completar un árbol con todos los nodos del grafo y un nuevo conjunto de

arcos. Este será un árbol des corte minimal en el que cada arco (i, j) se corresponda con la capacidad del corte $i - j$ mínimo en el grafo original.

A continuación te mostramos el Pseudocódigo:

Algorithm 3 Gusfield
Entrada: $G = (V, A); Z$
Salida: $G^{T'} = (V^{T'}, A^{T'}); Z^T$

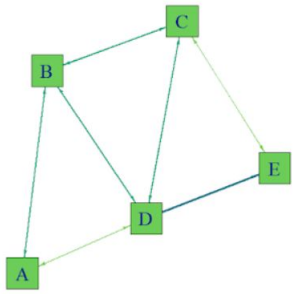
$V^T \leftarrow \{1\}$
$A^T \leftarrow \emptyset$
$Z^T_{|V|X|V|} \leftarrow \infty$
for cada nodo $i \in V1$ **do**
  $V^T_k \leftarrow V^T; A^T_k \leftarrow A^T$
  while $|V^T_k| > 1$ **do**
    borrar un arco $(i, j) \in A^T_k$ cuyo $a_{i,j}$ sea mínimo
    determinar componentes T_1 y T_2 de V^T_k y A^T_k tal que $i \in T_1$ y $j \in T_2$
    obtener corte mínimo entre i y j en el grafo $G = (V, A)$ original
    determinar componentes S_1 y S_2 de V^T y A^T tal que $i \in S_1$ y $j \in S_2$
    if nodo $i \in S_1$ **then**
      $V^T_k \leftarrow$ nodos de la componente T_1
      $A^T_k \leftarrow$ arcos de la componente T_1
    else
      $V^T_k \leftarrow$ nodos de la componente T_2
      $A^T_k \leftarrow$ arcos de la componente T_2
    end if
  end while
  añadir arco (i, k), con $k \in V^T_k$, al árbol A^T
  obtener corte mínimo entre i y k en el grafo $G = (V, A)$ original
  $Z^T_{ik} \leftarrow$ capacidad del corte mínimo $i - k$
  $V^T \leftarrow V^T \cup \{i\}$
end for

Experimentación con el algoritmo de Dijkstra:

Vamos a usar el algoritmo en código R,que se encuentra en el repositorio del grupo, en una matriz de pesos establecida y con lo cuál obtenemos los recorridos mínimos de nodo a nodo. Tenemos:
Matriz del grafo de 5 nodos
> mat

	A	B	C	D	E
A	0	4	0	2	0
B	4	0	5	6	0
C	0	5	0	7	1
D	2	6	7	0	8
E	0	0	1	8	0

#Creo grafo desde la matriz de adyecencia mat
g = graph.adjacency(mat, weightted = TRUE)
#PLOTEAR GRAFO
#{set.seed(68); plot(g)}



El grosor de los arcos es proporcional a su ”peso”. La solución del grafo tomando como nodo inicial la letra A (fuente=1) se produce con las siguientes líneas:
APLICACIÓN DEL ALGORITMO CON NODO FUENTE A (EQUIVALENTE A 1)
>Dij(Grafo=g, fuente=1)

#	Desde	Hasta	Distancia					
# 1	A	A	0					A
# 2	A	B	4			A	→	B
#3	A	C	9	A	→	D	→	C
# 4	A	D	2			A	→	D
# 5	A	E	10	A	→	D	→	E

Este resultado se puede verificar desde la matrix de adyacencia.

Referencias

[1] SMITH, David. Network Optimization Practice. John Wiley Sons. 1982
[2] TORANZOS, Fausto. Introducción a la Teoría de Grafos. OEA. 1976
[3] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” Numerische Mathematik, no. 1, pp. 269–271, 1959
[4] Optimal Trees: Programación en R de poblemas de búsqueda de árboles ótimos. Fontenla Cadavid, Manuel. Universidade de Santiago de Compostela. 2013 - 2014
[5] Matousek,J. y Nestril, J. Invitación to Discrete Mathematics, Claredon Press - Oxford, 1998
[6] Ross, Kenneth A., Discrete Mathematics, Princeton Hall, 1970