ISSN: 1692-7257 - Volumen 1 – Número 31 - 2018

Revista Colombiana de Tecnologías de Avanzada

DOI: https://doi.org/10.24054/16927257.v31.n31.2018.2780

Recibido: 23 de mayo de 2017
Aceptado: 22 de junio de 2017

ÁRBOL DE CAMINOS MÍNIMOS: ENRUTAMIENTO, ALGORITMOS APROXIMADOS Y COMPLEJIDAD

MINIMUM PATH TREE: ROUTING, APPROXIMATE ALGORITHMS AND COMPLEXITY

Msc. Efren Romero Riaño[*], Msc. Gabriel Mauricio Martínez Toro[**]
Msc. Dewar Rico-Bautista[***]

[*] **Universidad Manuela Beltrán1,** Grupo de Investigación GIGIA.
+57 3016427051. Bucaramanga, Santander, Colombia.
Efren.romero@docentes.umb.edu.co
[**]**Universidad Autónoma de Bucaramanga,** Facultad de Ciencias Económicas, Administrativas y Contables, Programa De Administración de Empresas Formación Dual Universitarias, Grupo GENIO.
Bucaramanga, Santander, Colombia.
Celular: 3005126223 E-mail: gmartinez714@unab.edu.co
[***] **Universidad Francisco de Paula Santander Ocaña,** Departamento Sistemas e Informática. Grupo de ingeniería en innovación, tecnología y emprendimiento (GRIITEM).
Ocaña, Norte de Santander, Colombia.
Celular: 3123973390. E-mail: dwricob@ufpso.edu.co.

Resumen: El documento aborda la temática relacionada con los algoritmos aproximados como método para solución de problemas computacionalmente complejos. El objetivo de este paper, es presentar un análisis comparativo entre tres enfoques clásicos de solución de algoritmos y el enfoque de solución mediante algoritmos aproximados. Este análisis incluye la descripción de los códigos y pseudocódigos, así como su análisis comparativo en términos de complejidad en tiempo y espacio. La metodología implementada fue de revisión basada en indicadores biblio métricos de las fuentes y posterior análisis de contenido de los documentos seleccionados. La complejidad computacional de los algoritmos analizados se puede dividir en complejidad de tiempo y de espacio, cada una de estas agrupan los problemas según sus características dentro de los grupos P, NP, PSPACE, NPSPACE, entre otros. A través de la reducción del Set Cover Problem en una versión del Minimun Spanning Tree, MST, se logró comprobar la complejidad de este problema de conectividad. La eficiencia de los algoritmos se determinó con base en los recursos que son utilizados en el momento en que son ejecutados. Se presenta un paralelo entre algoritmos aproximados y algoritmos alternativos a la luz del ejemplo de la reducción del set cover en un árbol de expansión mínimo, logrando evidenciar la complejidad en algoritmos de conectividad. La complejidad computacional se mide a la luz del uso de los recursos, bien sea el uso de los recursos de tiempo o espacio, generando clasificaciones de problemas según sus características particulares medidas por estos dos parámetros.

Palabras clave: Complejidad computacional; MST; NP-HARD; NPSPACE; PSPACE.

Abstract: This paper approaches the topic related to approximation algorithms as a method to solve problems with computational complexity. The main objective of this paper is to present a comparative analysis of three classic approaches to solve algorithms against the approximation algorithm approach. The analysis includes the description of the codes and pseudocodes, also the comparative analysis related to complexity in time

and space. The methodology used is the revision based on a bibliometric index from various sources and afterward an analysis of the selected documents. Computational complexity from the analyzed algorithms can be divided into time complexity and space complexity, each of them are used to put together problems with similar characteristics into groups called P, NP, PSPACE, NPSPACE, and so on. By the reduction of the set cover problem into a Minimum spanning tree (MST), it was achieved to prove the connectivity complexity. The efficiency of the algorithms was determined by the usage of resources at the time to be executed.

Keywords: Computational complexity; MST; NP-HARD; NPSPACE; PSPACE.

1. INTRODUCCIÓN

Los algoritmos aproximados y las heurísticas surgen por la necesidad de resolver problemas que son ineficientes computacionalmente al encontrarse dentro de la clasificación de NP, NP-Hard. Una aproximación para buscar la solución a estos problemas surge de buscar alternativas que suavicen el requerimiento de soluciones en tiempo polinomial, a través de una búsqueda exhaustiva de soluciones optimas, explorando entre todas las posibles soluciones, con algoritmos en tiempo razonable (Williamson & Shmoys, 2011).

Los algoritmos aproximados y las heurísticas se utilizan especialmente en problemas de optimización, problemas que se caracterizan por tener varias posibles soluciones candidatas, entre las cuales se pueda hacer una comparación de forma tal que se logre el mejor resultado (Frederickson, Hecht, & Kim, 1976). Los problemas de optimización pueden clasificarse en dos tipos, los que tienen una solución en valores enteros y los que tienen soluciones que se encuentran en un conjunto de opción finita de posibilidades (problemas combinatorios), tales como permutaciones o estructura de un grafo. (Duarte Muñoz, Abraham, 2007). La principal distinción que se hace entre una heurística y un algoritmo aproximado es la limitante que las heurísticas tienen de escapar de los óptimos locales, ya que estos algoritmos generalmente no usan mecanismos que puedan continuar la búsqueda cuando se encuentran estos valores, a lo cual se crean metaheuristicas, como método aproximado de algoritmos, que guían las heurísticas a evitar el problema en mención, mejorando así la calidad de la solución (Duarte Muñoz, 2007).

El estudio de los algoritmos aproximados aún está en etapa de desarrollo, de tal forma que la aproximación solo refleja el rendimiento del algoritmo utilizado, dando pie a que esta sea vista más bien como una motivación de profundizar en técnicas que provean una exploración más profunda de la estructura combinatoria de los problemas (Patiño, Moreno, & Toro, 2013). Problemas tales como el de la Ruta más corta (*Shortest path*), *mínimum spanning tree*, *matching*, *set cover*, *mínimum Steiner tree*, son algunos de los problemas que se han tratado de resolver por medio de algoritmos exactos en el caso de los tres primeros de la lista, y de algoritmos aproximados en el caso de los dos últimos (Vazirani, 2003)

Este documento puntualiza algoritmos aproximados en problemas de optimización de Arboles Generadores, en problemas representados en grafos con el objetivo de Maximizar o minimizar. Los grafos se encuentran parametrizados por G=(V, E, w). (Rodriguéz Gálvez, 2009)

El artículo está dividido en tres partes, inicialmente se hace una contextualización de los algoritmos aproximados y su aplicación. La segunda parte corresponde a exponer un algoritmo aproximado y su complejidad. Finalmente, se hace un paralelo entre algoritmos aproximados y algoritmos alternativos a la luz de un ejemplo.

2. ÁRBOLES GENERADORES CON COSTE DE RUTEO MÍNIMO – ENFOQUE APROXIMADO

Se considera el siguiente problema en el diseño de redes: Dado un grafo no dirigido con retardos no negativos sobre las aristas, el objetivo es encontrar un árbol generador tal que el promedio de retardo de comunicación entre cualquier par de vértices del árbol se minimice (Parra & Herrera, 2013; Medina & Rico, 2009). El retardo entre un par de vértices es la suma de los retardos de las aristas en el camino entre ellos en el árbol. Minimizar el retardo promedio es equivalente a

minimizar el retardo total entre todos los pares de vértices usando el árbol. (Gupta & K??nemann, 2011)

En general, cuando el coste sobre una arista representa un precio para el ruteo de mensajes entre sus puntos extremos, el coste de ruteo para un par de vértices en un árbol generador dado se define como la suma de los costes de las aristas en el único camino en el árbol entre ellos. El coste de ruteo del árbol es la suma de todos los costes de ruteo de todos los pares de vértices del árbol. Por ejemplo, $C(T) = \Sigma u, v\ dT\ (u, v)$, donde $dT\ (u, v)$ es la distancia entre u y v en T. Para un grafo no dirigido, el coste de ruteo mínimo de un árbol generador (MRCT) es el que tiene menor coste de ruteo entre todos los posibles arboles generadores.

Se asume que G es un grafo simple y no dirigido y que los pesos de las aristas son no negativos. Encontrar un MRCT en un grafo no dirigido y aristas con pesos es un problema NP-completo.

A. Definición de c-aproximación

Ya que todas las aristas son de longitud no negativa, la distancia entre dos vértices no disminuye al eliminar aristas del grafo. Obviamente, $dT\ (u, v) \geq dG\ (u, v)$ para cualquier árbol generador T de G. Se puede obtener un límite inferior trivial del coste de ruteo.

Un algoritmo aproximado para un problema de optimización es un algoritmo que alcanza una solución del problema pero que no garantiza la solución óptima. ¿Cómo medir cuanto nos acercamos a ella?(Garroppo, Giordano, & Tavanti, 2010)

Si S* es la solución óptima con coste c(S*), un algoritmo δ -aproximado es un algoritmo que devuelve una solución S tal que:

$c(S) \leq \delta\ c(S^*)$ (para un problema de minimización)
$c(S^*) \leq \delta\ c(S)$ (para un problema de maximización)

B. Algoritmo 2-Aproximación basado en la mediana

La mediana de un grafo se puede encontrar fácilmente una vez que se conocen las distancias entre todos los pares de vértices. Por el teorema (El árbol de camino mínimo con origen en la mediana de un grafo es una 2-aproximación de un MRCT del grafo.), se puede tener un algoritmo con una 2-aproximación y el tiempo de complejidad viene determinado por encontrar las longitudes del camino mínimo entre todos los pares del grafo de entrada. (Rodriguéz Gálvez, 2009)

Propiedad

$C($**¡Error! No se encuentra el origen de la referencia.**$) \geq \Sigma u,v \epsilon V\ d_G\ (u, v)$.

Sea **r** la **mediana** del grafo G = (V, E, w), es decir, el vértice con mínima distancia total a todos los vértices. En otras palabras, r minimiza la función $f(v) = \Sigma_{u \epsilon V}\ d_G\ (u,v)$. Se puede demostrar que el árbol de camino mínimo con origen en r es una 2-aproximacion de un MRCT.

Demostración

Sea r la mediana de un grafo G = (V, E, w) y sea Y cualquier árbol de caminos mínimos que empieza en **r**. Nótese que la desigualdad del triángulo mantenida por la distancia entre vértices de cualquier grafo sin aristas de peso negativos. Por la desigualdad triangular se tiene: $d_Y\ (u,v) \leq d_Y\ (u, r) + d_Y\ (v, r)$ para cualquier vértice u y v. En resumen, para todos los pares de vértices se obtiene:

$$C(Y) = \Sigma_u \Sigma_v\ d_Y\ (u, v) \leq n\ \Sigma_u\ d_Y\ (u, r) + n\ \Sigma_v\ d_Y\ (v, r) = 2n\ \Sigma_v\ d_Y\ (v, r)$$

Ya que **r** es la mediana, $\Sigma_v\ d_G\ (r, v) \leq \Sigma_v\ d_G\ (u, v)$ para cualquier vértice **u**, por lo tanto:

$$\Sigma_v\ d_G\ (r, v) \leq 1/n\ \Sigma_{u,v}\ d_G\ (u, v)$$

En un árbol de caminos mínimos el camino desde la raíz a cualquier vértice es el camino mínimo en el grafo original. Por tanto, $d_Y\ (r, v) = d_G\ (r, v)$ para cada vértice v, y consecuentemente,

$$C(_Y) \leq 2n\ \Sigma_v\ d_G\ (r, v) \leq 2\ \Sigma_{u,v}\ d_G\ (u, v)$$

Por la propiedad, Y es una 2-aproximacion de un MRCT.

Corolario

Un MRCT de un grafo puede ser aproximado con factor 2 de aproximación en tiempo $O(n^2 \log n + mn)$.

El factor de aproximación en el teorema 1 es rigurosa en el sentido de que existe un caso extremo para la desigualdad asintótica. Se considera un grafo completo con longitud unitaria sobre cada arista. Cualquier vértice es una mediana del grafo y el árbol de camino mínimo que empiece en la mediana es una estrella. El costo de ruteo de la estrella se puede calcular mediante la siguiente formula:

$$2(n-1)(n-2)+2(n-1) = 2(n-1)^2$$

Ya que la distancia total sobre el grafo es: n(n-1). El radio es: 2 – (2/n).

Se puede verificar que, para el caso anterior, la estrella es además una solución óptima. El problema es que podríamos comparar la solución aproximada con la óptima, pero no con el límite inferior trivial. Ahora introducimos otra prueba del factor de aproximación del árbol de camino mínimo. La técnica de análisis que se utiliza se llama descomposición de la solución, la cual es ampliamente utilizada en el diseño de algoritmos, especialmente para la aproximación de algoritmos (Santos & Flórez, 2013).

Para diseñar un algoritmo de aproximación para un problema de optimización, primeramente, se supone que X es una solución óptima. Luego se descompone X y se construye otra solución factible Y. Para nuestro propósito, Y es diseñada para ser una buena aproximación de X y pertenece a algún subconjunto restringido de soluciones factibles, de la cual, la mejor solución puede ser encontrada eficientemente. El algoritmo es diseñado para encontrar la solución óptima del problema restringido, y el factor de aproximación se determina utilizando Y. Y juega un papel solamente en el análisis del factor de aproximación, pero no en el diseño del algoritmo. (Rodriguéz Gálvez, 2009)

Para cualquier árbol, siempre se puede cortar por un nodo r tal que cada rama contiene como mucho la mitad de los nodos. Tal nodo se le suele llamar el centroide del árbol. Por ejemplo, en la figura 1(a), el vértice r es un centroide del árbol. El árbol tiene nueve vértices. Eliminando r del árbol se obtienen tres subárboles y cada subárbol no tiene más que cuatro vértices. Se puede verificar que r es el único centroide del árbol de la figura 1(a). Sin embargo, para el árbol 1(b), r1 y r2 son los centroides del árbol.

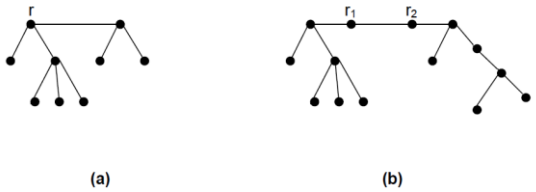

Fig 1. Centroide de un árbol. Fuente: (Rodriguéz Gálvez, 2009)

Suponiendo que r es el centroide del MRCT **¡Error! No se encuentra el origen de la referencia.**. Si se construye arbol de caminos mínimos Y tomando como raíz el centroide r, el coste de la ruta será como mucho dos veces el de **¡Error! No se encuentra el origen de la referencia.**. Se puede demostrar continuación.

Primeramente, si u y v son dos nodos en distintas ramas, **¡Error! No se encuentra el origen de la referencia.** (u,v) = **¡Error! No se encuentra el origen de la referencia.**(u,r) + **¡Error! No se encuentra el origen de la referencia.**(v,r). Se considera la distancia total de todos los pares de nodos de **¡Error! No se encuentra el origen de la referencia.**. Para cualquier nodo v, ya que cada rama no contiene más que la mitad de los nodos, el termino **¡Error! No se encuentra el origen de la referencia.**(u,r) se contara en la distancia total al menos n veces, n/2 veces desde v al resto y n/2 veces del resto a v. Se tiene que C(**¡Error! No se encuentra el origen de la referencia.**) $\geq n \Sigma_v$ **¡Error! No se encuentra el origen de la referencia.**(v,r). Ya que como en la prueba del teorema, $C(Y) \leq 2n \Sigma_v d_G$ (v,r), por consiguiente $C(Y) \leq 2$ C(**¡Error! No se encuentra el origen de la referencia.**). Se ha descompuesto la solución optima **¡Error! No se encuentra el origen de la referencia.** y construido una 2-aproximacion Y.

3. ÁRBOLES DE EXPANSIÓN MÍNIMA – ENFOQUE TRADICIONAL

Este apartado corresponde a la presentación de la solución de problemas mediante algoritmos clásicos. A continuación, se aborda, la descripción del problema del árbol de expansión mínimo, MST, desde la óptica de tres algoritmos de complejidad de tiempo Polinomial. Los nombres de estos algoritmos clásicos son: Kruskal, Prims y Baruvka. A continuación, se describen el problema y los algoritmos.

A. Descripción de entrada del problema:

Un grafo G = (V, E) con bordes ponderados.

Descripción del problema: El subconjunto de peso mínimo de los bordes E´ ⊂ E que forman un árbol en V.

El árbol de expansión mínimo (MST) de un grafo, define el subconjunto más barato de aristas que mantiene el grafo conectado en un componente. Este problema es de especial interés para compañías de telefonía interesadas en identificar el conjunto de ubicaciones que define el esquema de cableado que conecta los sitios, usando el menor cable posible. MST presenta el enfoque seminal de todos los problemas de diseño de red. (Skiena, 1982).

Las principales características del MST son: i) se pueden calcular rápida y fácilmente, ii) proporcionan una manera de identificar clusters en conjuntos de puntos, iii) pueden ser utilizados para dar soluciones aproximadas a problemas duros como el árbol Steiner y el vendedor ambulante, iv) proporcionan evidencia gráfica de que los algoritmos avariciosos, pueden proporcionar soluciones óptimas.

B. El algoritmo de Kruskal -

Este plantea que cada vértice comienza como un árbol separado y estos árboles se fusionan agregando repetidamente el borde de menor costo que abarca dos subárboles distintos (es decir, no crea un ciclo). A continuación, se presenta el algoritmo, donde n se refiere al número de vértices en un grafo, mientras que m es el número de aristas.

```
Kruskal(G)
Sort the edges in order of increasing weight
count = 0
while (count < n − 1) do
 get next edge (v,w)
if (component (v) ≠component(w))
add to T
component (v) = component(w)
Componente (v) = componente (w)
```

C. El algoritmo de Prim

Este comienza con un vértice v arbitrario y "crece" un árbol a partir de él, encontrando repetidamente el borde de menor costo que enlaza algún nuevo vértice en este árbol. Durante la ejecución, etiquetamos cada vértice como en el árbol, en la franja (es decir, existe un borde desde un vértice del árbol), o invisible (es decir, el vértice se encuentra a más de un borde del árbol). A continuación, se presenta el algoritmo.

```
Prim (G)
Select an arbitrary vertex to start
While (there are fringe vertices)
select minimum-weight edge between
tree and fringe
add the selected edge and vertex to the
tree
update the cost to all affected fringe
vertices
```

D. Algoritmo de Boruvka

Boruvka se basa en la observación de que el borde de menor peso incidente en cada vértice debe estar en el árbol de expansión mínimo. La unión de estos bordes resultará en un bosque que abarca un máximo de n / 2 árboles. Ahora, para cada uno de estos árboles T, seleccione el borde (x, y) de menor peso tal que x ∈ T y y ∈ T. Cada uno de estos bordes debe estar de nuevo en un MST, y la unión de nuevo resulta en un bosque de extensión con máximo la mitad de árboles que antes. A continuación se presenta el algoritmo.

```
Boruvka(G)
```

Initialize spanning forest F to n single-vertex trees
While (F has more than one tree)
for each T in F, find the smallest edge from T to G − T
add all selected edges to F, thus merging pairs of trees

La mayoría de problemas, se pueden plantear en términos de algoritmos basados en grafos (p.e. minimización del ancho de banda o el mejoramiento de autómatas de estado finito). El problema y los tres algoritmos presentados se incluyen en razón a que existen algoritmos eficientes para su solución, su tiempo de ejecución crece de forma polinomial con el tamaño del grafo.

La descripción de los tiempos de los algoritmos es en su orden: kruskal, obtiene un algoritmo de tipo O (m lg m); el algoritmo de Prim, con base a datos sencillos obtiene una implementación en tiempo (O n2); y el algoritmo de Boruvka, después de como máximo, log n iteraciones, cada una de las cuales toma tiempo lineal, obtiene un tiempo O (m log n).

4. ÁRBOLES DE CAMINO MÍNIMO – ENFOQUE TRADICIONAL

Llegar a destino, en tiempo y forma, puede requerir que el algoritmo de enrutamiento, que es el encargado de escoger las rutas y las estructuras de datos, cumpla con ciertas propiedades que aseguren la eficiencia de su trabajo.(Siachalou & Georgiadis, 2003)

Estas propiedades son: corrección, estabilidad, robustez, equitatividad, sencillez y optimalidad. La corrección y la sencillez casi no requieren comentarios; no así la necesidad de robustez, la cual se refiere a que el algoritmo debe ser diseñado para que funcione dentro de la red por años, sin fallas generales. El algoritmo deberá estar preparado para manejar cambios de topología y tráfico sin requerir el aborto de las actividades o el reinicio de la red.(Levin, Kowalski, & Segal, 2015)

La equitatividad y la optimalidad resultan con frecuencia contradictorias, ya que muchas veces se requiere una concesión entre la eficacia global (optimización) y la equitatividad; es decir, antes de intentar encontrar un justo medio entre estas dos, se debe decidir qué es lo que se busca optimizar. Minimizar el retardo de los paquetes (disminuyendo escalas y ancho de banda) y maximizar el rendimiento total de la red sería la combinación más apropiada para un algoritmo de ruteo.

Este postulado de optimización establece que, si el enrutador J está en la trayectoria óptima del enrutador I al enrutador K, entonces la trayectoria óptima de J a K también está en la misma ruta. Haciendo referencia a la figura 2, llamemos r1 a la parte de la ruta de I a J, y r2 al resto de la ruta. Si existiera una ruta mejor que r2 entre J y K, podría concatenarse con r1 para mejorar la ruta entre I y K, contradiciendo nuestra aseveración de que r 1 y r2 es óptima.

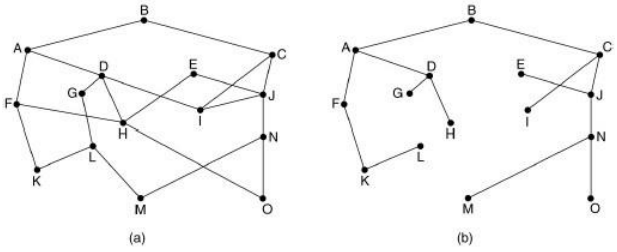

Fig 2. (a) Subred. (b) Árbol de Descenso para el enrutador B. Fuente: (Laporte & Osman, 1995)

Como consecuencia directa del principio de optimalidad, podemos ver que el grupo de trayectorias óptimas de todas las de orígenes a un destino dado forma un árbol con raíz en el destino. Ese árbol que se forma, se llama árbol de descenso, donde la métrica de distancia es el número de escalas. El árbol de descenso puede no ser único, pueden existir otros árboles con las mismas longitudes de trayectoria.

Dado que un árbol de descenso ciertamente es un árbol, no contiene ciclos, por lo que cada paquete será entregado con un número de escalas infinito y limitado. En la práctica, no siempre sucede esto, los enlaces y los enrutadores pueden caerse y reactivarse durante la operación, por lo que diferentes enrutadores pueden tener ideas distintas sobre la topología actual de la subred.(Fan & Shi, 2010)

E. Algoritmo Dijkstra base del Protocolo OSPF

Este protocolo utiliza el algoritmo de la mejor ruta es la de menor costo. Se basa en el algoritmo que fue desarrollado por Edsger Dijkstra, un especialista holandés en informática quien lo describió por primera vez en 1959. (Bollobás & Riordan, 1993)

El algoritmo considera la red como un conjunto de nodos conectados con enlaces punto a punto. Cada enlace tiene un costo. Cada nodo tiene un nombre. Cada nodo cuenta con una base de datos completa de todos los enlaces y por lo tanto se conoce la información sobre la topología física en su totalidad. Todas las bases de datos del estado de enlace, dentro de un área determinada, son idénticas. (Méndez, Rodriguez-Colina, & Medina, 2014)

El algoritmo de la ruta más corta calcula una topología sin bucles con el nodo como punto de partida y examinando a su vez la información que posee sobre nodos adyacentes (Rojas & Sánchez, 2012; Medina & Rico, 2008; Medina, Rico, & Areniz, 2016). El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de costo negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).(Yin & Wang, 2010)

- **Algoritmo**

Teniendo un grafo dirigido ponderado de N nodos no aislados, sea x el nodo inicial, un vector D de tamaño N guardará al final del algoritmo las distancias desde x al resto de los nodos.(Yin & Wang, 2010) (Bollobás & Riordan, 1993)

1. Inicializar todas las distancias en D con un valor infinito relativo ya que son desconocidas al principio, exceptuando la de x que se debe colocar en 0 debido a que la distancia de x a x sería 0.
2. Sea a = x (tomamos a como nodo actual).
3. Recorremos todos los nodos adyacentes de a, excepto los nodos marcados, llamaremos a estos nodos no marcados v_i.
4. Si la distancia desde x hasta vi guardada en D es mayor que la distancia desde x hasta a, sumada a la distancia desde a hasta v_i; esta se sustituye con la segunda nombrada, esto es: si ($D_i > D_a + d(a, v_i)$) entonces $D_i = D_a + d(a, v_i)$
5. Marcamos como completo el nodo a.
6. Tomamos como próximo nodo actual el de menor valor en D (puede hacerse almacenando los valores en una cola de prioridad) y volvemos al paso 3 mientras existan nodos no marcados.

Una vez terminado al algoritmo, D estará completamente lleno.

- **Pseudocódigo**

• Estructura de datos auxiliar: Q = Estructura de datos Cola de prioridad

```
DIJKSTRA (Grafo G, nodo_fuente s)
    para u ∈ V[G] hacer
        distancia[u] = INFINITO
        padre[u] = NULL
    distancia[s] = 0
    adicionar (cola, (s,distancia[s]))
    mientras que cola no es vacía hacer
        u = extraer_minimo(cola)
        para todos v ∈ adyacencia[u] hacer
            si distancia[v] > distancia[u] + peso (u,
            v) hacer
                distancia[v] = distancia[u] + peso (u,
                v)
                padre[v] = u
                adicionar(cola,(v,distancia[v]))
```

- **Complejidad**

Orden de complejidad del algoritmo: $O(|V|^2+|E|) = O(|V|^2)$ sin utilizar cola de prioridad, $O((|E|+|V|) \log |V|)$ utilizando cola de prioridad. (Barbehenn, 1998)

Se puede estimar la complejidad computacional del algoritmo de Dijkstra (en términos de sumas y comparaciones). El algoritmo realiza a lo más n-1 iteraciones, ya que en cada iteración se añade un vértice al conjunto distinguido. Para estimar el número total de operaciones basta estimar el número de operaciones que se llevan a cabo en cada iteración.

Se puede identificar el vértice con la menor etiqueta entre los que no están en S_k realizando n-1 comparaciones o menos. Después se hace una suma y una comparación para actualizar la etiqueta de cada uno de los vértices que no están en S_k. Por

tanto, en cada iteración se realizan a lo sumo 2(n-1) operaciones, ya que no puede haber más de n-1 etiquetas por actualizar en cada iteración. Como no se realizan más de n-1 iteraciones, cada una de las cuales supone a lo más 2(n-1) operaciones, llegamos al siguiente teorema.

- **TEOREMA:**

El Algoritmo de Dijkstra realiza $O(n^2)$ operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo y no dirigido con n vértices.

El algoritmo de Dijkstra se puede considerar formado por la secuenciación de una inicialización.

INIC: A:= V\{f};
for ¡Error! No se encuentra el origen de la referencia. rof;

y de un ciclo:

CICLO:
do ¡Error! No se encuentra el origen de la referencia.
A:= A\{w};
for ¡Error! No se encuentra el origen de la referencia.
$df[v] := \min\{df[v], df[w] + c[w,v]\}$
rof
od

El tamaño del problema es **n**, el número de vértices del grafo. Por otra parte, sea e = |E| el número de arcos del grafo. Obsérvese que **¡Error! No se encuentra el origen de la referencia.**, de modo que **¡Error! No se encuentra el origen de la referencia.**.

Como operaciones básicas, considérense:
- # : añadir un elemento de un conjunto
- \ : eliminar un elemento de un conjunto
- Escoja_min : escoger el elemento de un conjunto que alcanza el mínimo de una medida
- min : calcular el mínimo de dos números.

Así:

TD(n) = T(INIC) + T(CICLO)

T(INIC) = O(n) T(#)
T(CICLO) = O(n) [T(Escoja_min) + T(\)] + O(e) T(min)

Es decir:

TD(n) = O(n) [T(#) + T(Escoja_min) + T(\)] + O(e) T(min)

La complejidad definitiva depende de las estructuras de datos que se utilicen (Araque, Díaz, & Gualdrón, 2013). En otras palabras, ya que es concebible representar tanto el grafo *G(V,E,c)* como el conjunto de abiertos A con diversas estructuras de datos, el uso de recursos dependerá de la conveniencia de estas estructuras para realizar el algoritmo.

A continuación, se muestran tres variantes de representación que arrojan complejidades diferentes para las ejecuciones del algoritmo de Dijkstra:

Variante 1:

- G **¡Error! No se encuentra el origen de la referencia.**matriz de distancias directas (adyacencias etiquetadas con costos): $O(n^2)$.
- A **¡Error! No se encuentra el origen de la referencia.**arreglo booleano: O(n).

Como costo de las operaciones básicas, se supone:
- T(#) = O(1)
- T(\) = O(1)
- T(Escoja_min) = O(n)
- T(min) = O(1)

Entonces:

TD(n) = O(n) [O(1) + O(n) + O(1)] + O(e) O(1)
= $O(n^2 + e)$
= $O(n^2)$.

Variante 2:

- G **¡Error! No se encuentra el origen de la referencia.**lista de distancias directas: $O(n^2)$.

- A **¡Error! No se encuentra el origen de la referencia.** arreglo booleano: O(n).

Como costo de las operaciones básicas, se supone:

- T(#) = O(1)
- T(\) = O(1)
- T(Escoja_min) = O(n)
- T(min) = O(1)

El análisis es idéntico al del caso anterior:

TD(n) = $O(n^2)$

Variante 3:

- G **¡Error! No se encuentra el origen de la referencia.**lista de distancias directas: $O(n^2)$.
- A**¡Error! No se encuentra el origen de la referencia.** montón (inglés: heap) : O(n).

Como costo de las operaciones básicas, se supone:

- T(#) = O(log n)
- T(\) = O(log n)
- T(Escoja_min) = O(1)
- T(min) = O(1)

TD(n) = O(n) [O(log n) + O(1) + O(log n)] + O(e) O(1)
= O(n log n + e)

Ahora, si el grafo G(V,E,c) no tiene "demasiados arcos", por ejemplo, si **¡Error! No se encuentra el origen de la referencia.** log n, se tendrá que:

TD(n) = O(n log n).

5. CONCLUSIONES

Cuando la suma de los pesos de las aristas, por ejemplo, representado en velocidad de enlaces es mínima, se considera entonces que es un árbol generador mínimo (MST). En otras palabras, un árbol generador mínimo es un árbol constituido por un subconjunto de aristas de un grafo no dirigido.

Los algoritmos aproximados son una herramienta útil para enfrentar problemas de alta complejidad y buscan brindar soluciones de una calidad probable, mientras que otros algoritmos ofrecen soluciones subóptimas dentro de las posibilidades de tiempo de computación limitado.

Para el caso de los árboles de expansión mínima, los algoritmos aproximados constituyen una opción para el mejoramiento de la eficiencia. Esto se evidencia en la disminución del consumo de recursos representado en tiempo de computación, que se identifica al calcular la diferencia entre los tiempos de computación del algoritmo clásico y el aproximado.

REFERENCIAS

Araque, J., Díaz, J., & Gualdrón, O. (2013). Optimización del thd en un convertidor multinivel monofásico usando algoritmos genéticos. Revista Colombiana Tecnologías de Avanzada , 1(21), 60-66. doi:https://doi.org/10.24054/16927257.v21.n21.2013.297

Ariganello, E. (2014). *Redes Cisco. Guía de estudio para la certificación. CCNA Routing y switching* (Primera ed.). México D.F., México: Alfaomega. Recuperado el 15 de Mayo de 2017

Barbehenn, M. (1998). A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. *IEEE Transactions on Computers*, *47*(2), 263. https://doi.org/10.1109/12.663776

Bollobás, B., & Riordan, O. (1993). Dijkstra â€™s Algorithm. *Network*, *69*(1959), 36114. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/21282851

Duarte Muñoz, A. (2007). *Metaheuristicas.* Madrid: Dykinson.

Fan, D., & Shi, P. (2010). Improvement of Dijkstra's algorithm and its application in route planning. *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery*. https://doi.org/10.1109/FSKD.2010.5569452

Frederickson, G. N., Hecht, M. S., & Kim, C. E. (1976). Approximation agorithms for some routing problems. *17th Annual Symposium*

on Foundations of Computer Science (Sfcs 1976), *7*(2), 178–193. https://doi.org/10.1109/SFCS.1976.6

Garroppo, R. G., Giordano, S., & Tavanti, L. (2010). A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, *54*(17), 3081–3107. https://doi.org/10.1016/j.comnet.2010.05.017

Gupta, A., & Könemann, J. (2011). Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, *16*(1), 3–20. https://doi.org/10.1016/j.sorms.2010.06.001

Klinkowski, M., & Walkowiak, K. (2016). On the complexity of routing and spectrum allocation in survivable elastic optical network with unicast and anycast traffic. *International Workshop on Resilient Networks Design and Modeling (RNDM).* Halmstad: IEEE. doi:10.1109/RNDM.2016.7608283

Laporte, G., & Osman, I. H. (1995). Routing problems: A bibliography. Annals of Operations Research, 61(1), 227–262. https://doi.org/10.1007/BF02098290

Levin, L., Kowalski, D. R., & Segal, M. (2015). Message and time efficient multi-broadcast schemes. *Theoretical Computer Science*, *569*, 13–23. https://doi.org/10.1016/j.tcs.2014.12.006

Medina Cárdenas, Y., & Rico Bautista, D. (2008). Modelo de gestión de servicios para la universidad de Pamplona: ITIL. Scientia Et Technica, XIV (39), 314-319.

Medina Cárdenas, Y., & Rico Bautista, D. (2009). Modelo de gestión basado en el ciclo de vida del servicio de la Biblioteca de Infraestructura de Tecnologías de Información (ITIL). Revista Virtual Universidad Católica del Norte, (27), 1-21.

Medina, Y., Rico, D., & Areniz, Y. (2016). Modelo estratégico para la gestión tecnológica en la organización: plan táctico de la calidad (ITIL & ISO 20000). (I. T. Metropolitano, Ed.) Ocaña: Fondo Editorial ITM. doi:https://doi.org/10.22430/9789585414006

Méndez, L., Rodriguez-Colina, E., & Medina, C. (2014). Toma de Decisiones Basadas en el Algoritmo de DIJKSTRA. *Redes de Ingeniería*, *4*(2), 2013. Retrieved from http://revistas.udistrital.edu.co/ojs/index.php/REDES/article/view/6357/7872

Parra, C., & Herrera, J. (2013). Aplicación de los sistemas de detección de intrusos y la tecnología de agentes en el monitoreo inteligente de redes de datos. Revista Colombiana de Tecnologías de Avanzada, 106-110. doi:https://doi.org/10.24054/16927257.v22.n22.2013.417

Patiño, N., Moreno, M., & Toro, E. (2013). Modelamiento matemático del problema de ocupación de quirófanos. Revista Colombiana Tecnologías de Avanzada, 2(20), 43-49. doi:https://doi.org/10.24054/16927257.v20.n20.2012.187

Rojas, M., & Sánchez, M. (2012). Arquitectura de software para el servicio de soporte de tecnología de información basada en servicios web. Revista Colombiana Tecnologías de Avanzada, 2(20), 144-150. doi:https://doi.org/10.24054/16927257.v20.n20.2012.201

Rodriguéz Gálvez, M. D. (Enero de 2009). Problemas de Optimozación en Arboles Generadores. *Trabajo de fin de carrera*. Madrid.

Santos, L., & Flórez, A. (2013). Metodología para el análisis forense en linux. Revista Colombiana Tecnologías de Avanzada, 2(20), 90-96. doi:https://doi.org/10.24054/16927257.v20.n20.2012.194

Siachalou, S., & Georgiadis, L. (2003). Efficient QoS routing. *Computer Networks*, *43*(3), 351–367. https://doi.org/10.1016/S1389-1286(03)00286-X

Skiena, S. (1982). The Algorithm Design Manual Second Edition. Department of Computer Science State University of New York at Stony Brook New York, USA.

Universidad de los Andes. (2016). *Diseño y Análisis de algoritmos. DISC – Departamento de Ingeniería de Sistemas y Computación.* Recuperado el 15 de Mayo de 2017, de https://cursos.virtual.uniandes.edu.co/isis1105/

Vazirani, V. V. (2003). *Approximation Algorithms.* Berlin: Springer.

Williamson, D. P., & Shmoys, D. B. (2 de 2 de 2011). The Design of Approximation

Algoritms. Cambridge: Cambridge University Press.

Yin, C., & Wang, H. (2010). Developed Dijkstra shortest path search algorithm and simulation. In *2010 International Conference on Computer Design and Applications, ICCDA 2010* (Vol. 1). https://doi.org/10.1109/ICCDA.2010.5541129