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,corte mínimo.**

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 del árbol de camino más corto 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.

II. Objetivos

1. El principal objetivo del presente artículo es obtener caminos de recorrido mínimo con aplicaciones de la vida cotidiana mediante el algoritmo de Dijkstra.
2. Comparar el algoritmo de Dijkstra con otros algoritmos de cortes mínimos en grafos.

III. Estado del arte

- **Redes Inalámbricas Malladas**
  Las redes inalámbricas malladas (Wireless Mesh Networks o **WMN**) son aquellas en las que existen uno o más nodos intermedios que envían y reciben paquetes para facilitar las comunicaciones entre nodos que no pueden hacerlo directamente. A diferencia de los sistemas más tradicionales en los que existe un enlace inalámbrico entre nodos, las MWN pueden ampliar la cobertura de una red, mejorar su conectividad, y permitir una mayor velocidad de transmisión de datos, lo que aumenta el rendimiento y proporciona un uso más eficiente del medio.
  A la hora de trabajar con este tipo de redes, suelen representarse habitualmente como un grafo en el que los vértices simbolizan los nodos inalámbricos y las aristas representan un enlace entre nodos. Cada nodo posee un rango de cobertura inalámbrica determinado, y para que exista un enlace entre dos nodos ambos deben estar dentro del radio de cobertura del otro, como se muestra en la figura 1. Este tipo de redes necesitan usar ciertas técnicas de enrutamiento que, al igual que sus homólogas cableadas, procuran elegir la mejor secuencia de nodos intermedios entre la fuente y el destino para retransmitir la información.

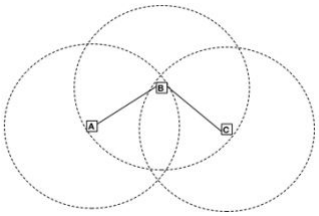

Figura 1: Representación de nodos inalámbricos en una MWN.

- **Algoritmos de exploración de grafos**
  Mencionaremos algunos de los algoritmos de búsqueda más importantes. Su uso como herramienta auxiliar ha resultado fundamental para muchas de las funciones implementadas, y frecuentemente ha sido necesaria su correspondencia adaptación con el objetivo de asegurar el correcto funcionamiento de los algoritmos desarrollados a lo largo del trabajo.
  - **BTS**(Breatch-First Search): Se trata de un conocido algoritmo de exploración de grafos, que comienza en un nodo $root$ o raíz para después recorrer uno por uno todos sus "hijos". Se puede decir que explora la red .[en] anchura"puesto que primero ha de revisar todos los nodos de un nivel antes de avanzar al siguiente nivel. En la figura que se mostrara se observa un ejemplo de exploración de grafo sencillo mediante **BFS**. Su eficiencia espacial usando matrices de adyacencia (como será el caso) es $\theta(|V|)$, mientras que la temporal es $\theta(|V| + |E|)$.

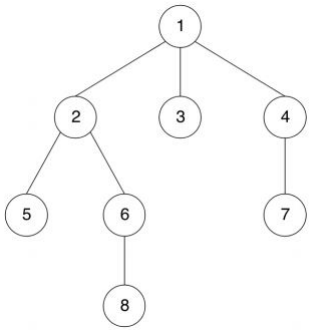

Figura 2: Recorrido en BFS.

- **DFS**(Depth-First Search): De forma opuesta a la de **BFS, DFS** explora el grafo .en profundidad". Despues de inspeccionar un vértice explora sus "hijos"de manera recursiva hasta llegar a uno que sea hoja (no tiene más descendientes), para después volver a ascender y seguir inspeccionandoel resto. en la siguiente figura muestra un ejemplo de este tipo de recorrido, con el orden de exploración representado por el número de cada vértice. Su complejidad espacial es $\theta(|V|)$ y la temporal $\theta(|E|)$ (exceptuando casos concretos).

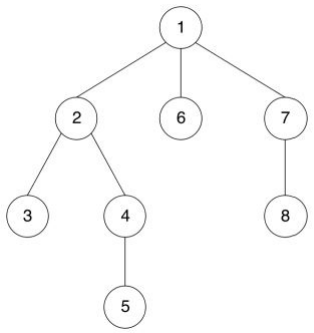

Figura 3: Recorrido en DFS.

- **Algoritmos para hallar el camino más corto**
  Mencionaremos una clase algorítmica que se empleará muy frecuentemente en este proyecto es la dedicada a encontrar el $path$ o camino más corto en un grafo dado. Esto es, la secuencia de nodos entre una fuente S y un destino D de menor número de saltos o de un coste global menor.
  - **Dijkstra:** Se trata de uno de los algoritmos más conocidos de la teoría de grafos, propuesto por Edger Dijkstra en 1959. Dado un grafo conexo y de distancias no negativas, encuentra el camino de coste mínimo entre un vértice dado y el resto de vértices del grafo. Su eficiencia computacional es $\theta(|V|^2)$, y $\theta((|E|+|V|)log(|V|))$ usando colas de prioridad.
  - **KSP** (K-Shortest Paths): Presentado por Jin Y. Yen en 1971, se trata de un algoritmo capaz de hallar los K caminos de menor coste entre dos nodos dados de un grafo, sin que estos contengan bucles. Su complejidad es de $\theta(N^2+KN)$ siendo N el número total de nodos del grafo y K la cantidad de caminos mínimos que se pretende hallar. El funcionamiento a grandes rasgos es el que se describe a continuación:
    1. Hallar el camino de menor coste (mediante Dijkstra por ejemplo) entre un nodo fuente s y su destino d.
    2. Aplicar una serie de modificaciones al camino previo para generar una ruta alternativa entre s y d, evitando crear ciclos internos. Almacenarla temporalmente.
    3. Repetir el paso 3 un número de veces equivalente a la longitud .en saltos"del último camino más corto hallado.
    4. Seleccionar de todos los caminos temporalmente almacenados el de menor distancia y guardarlo posteriormente.
    5. Repetir hasta $K-1$ veces.
  - **BFS**(Breadth-First): Como se verá más adelante, este algoritmo puede usarse también pra localizar el camino mínimo entre dos nodos puesto que recorre todos los vértices del grafo de forma progresiva. Aunque su comportamiento es análogo al del algoritmo de Dijkstra, su eficiente computacional es mayor, por lo que su uso resultará interesante en algunas de las funciones y algoritmos implementados.

IV. Diseño del Experimento.

1. Algoritmo de Dijkstra:

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 $O(|V|^2)$.

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:**

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 esta 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 árbol 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 misma 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 con un árbol con un único nodo.

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

V. Experimentación con el algoritmo de Dijkstra:

1. Aplicación a red telefónica:

- **Introducción del sistema de salida de seguridad**
  El sistema de salida de seguridad encuentra un camino más corto en un edificio de gran altura durante incidentes críticos. La evacuación se enfrenta a dos problemas principales que son los evacuados que encuentran difícil encontrar las mejores rutas y su comportamiento hace que el proceso sea más difícil. Estos problemas son importantes ya que está relacionado con la vida de los humanos. Al proporcionar el camino más corto y controlar el comportamiento del evacuado, pueden llevar a una evacuación exitosa que se logra con el sistema de salida de seguridad.
- **Tres pasos están involucrados**
  El primer paso es crear un plan de diseño, seguido de la creación del gráfico de visibilidad y finalmente la implementación del algoritmo de Dijkstra.
  Pasos:

  a) Plan de diseño del edificio:
  En primer lugar se toma la estructura completa (modelo azul) del edificio. Esto se proporciona como una entrada al sistema de salida de seguridad para que se ejecute el algoritmo de Dijsktra.

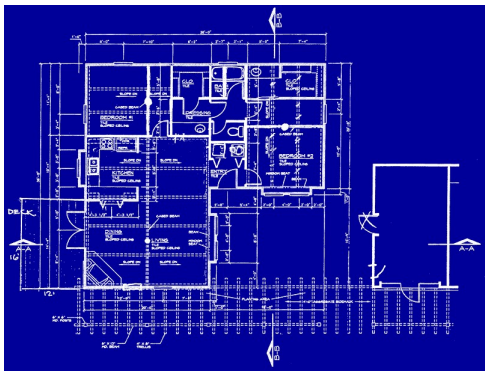

Figura 4: Modelo del edificio.

  b) Gráfico de visibilidad:
  Se crea un gráfico de visibilidad basado en divisiones como el piso, paralelos del edificio, etc. Teniendo en cuenta todos los puntos de salida en mente. El gráfico de visibilidad será direccional para considerar la dirección de salida. Los nodos en el gráfico serán sistemas orientados a sensores que tendrán sensores incorporados y el modo de indicación de señales y altavoces que dirigirán a los usuarios la dirección durante los desastres. En este sistema, el modo de indicación de señal y los altavoces solo se activan cuando se recibe una señal de alarma o el sensor detecta peligro.

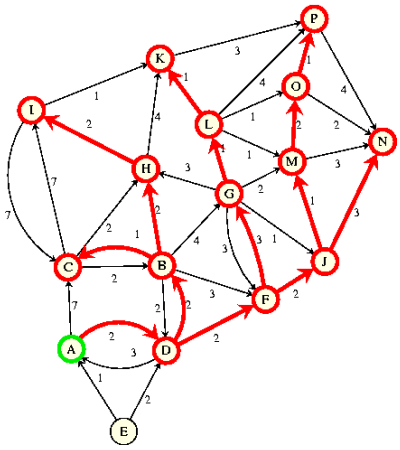

Figura 5: Esquema de red.

  c) Cálculo del camino más corto:
  Una vez que se activa la alarma de peligro, el sistema de salida de seguridad comienza su trabajo. Tiene incorporado un mecanismo central controlado que recibirá mensajes de los sensores. Una vez que estalló el desastre, todos los nodos aplican el algoritmo Dijkstra y calculan la ruta más corta de cada nodo para salir de la construcción. Así que las personas en el edificio solo tendrán que seguir la ruta especificada para salir sin pánico.

- **Ejemplo:**
  En la figura, si el incendio se rompió en A, el sistema calculará la ruta más corta desde cada nodo, el sistema en A indicará que irá recto, ya que D es la ruta más corta para salir. Cálculo del recorrido más corto y trabajo de sensores de la mano.
- **Ejemplo:**
  En la figura, si los incendios se extienden de A a D, se aplica nuevamente el algoritmo dijkstra y se considera la nueva ruta más corta eliminando los nodos de A y D. La siguiente ruta para la evacuación será F. Este es el funcionamiento del sistema de salida de seguridad.

2. Aplicación al sistema de seguridad:

a) Las redes telefónicas tienen varios componentes tales como:
  1) Dispositivo de comunicación (móvil o fijo)
  2) Interruptores
  3) Routers
  4) Centro de estación base BSC
  5) MSC Mobile Switching Center
  6) HLR Home Location Register
  7) Registro de ubicación de visitantes

b) La configuración general de una red es la siguiente:
  1) Un dispositivo de comunicación estará conectado a un BSC. segundo.
  2) El BSC será a su vez conectado al MSC. do.
  3) El MSC tiene una base de datos conocida como HLR y VLR que ayuda a enrutar las conexiones

c) Para conectar un teléfono a otro, la llamada se enruta a través de numerosos conmutadores que operan a

nivel local, regional, nacional o internacional. La conexión establecida entre los dos teléfonos se llama un **circuito**.

d) En una red telefónica las líneas tienen ancho de banda.

e) El ancho de banda representa la cantidad de información que puede transmitir la línea.

f) La red telefónica se puede representar esquemáticamente de la siguiente manera:

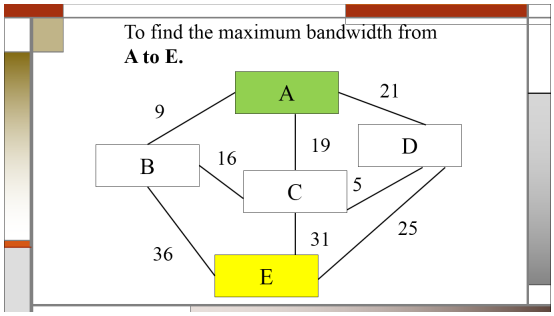

Figura 6: Esquema de red.

g) Los vértices representan la estación de conmutación, los bordes representan la línea de transmisión y el peso de los bordes representa el ancho de banda.

h) Ahora, cuando queremos conectar una llamada desde el dispositivo D1 al dispositivo D2 ocurrirán las siguientes acciones:

1) Las señales del dispositivo llegarán primero al BSC.
2) Varios BSC están conectados a un MSC.
3) Por lo tanto, el BSC transferirá la señal del BSC al MSC.
4) Ahora el MSC verificará con su base de datos.
5) Si el dispositivo está en itinerancia, verificará su base de datos VLR o, de lo contrario, lo hará con su base de datos HLR.
6) La base de datos tiene entradas que representan el mayor ancho de banda disponible entre dos estaciones y la ruta más corta para llegar a esa estación.

i) El algoritmo de dijkstra ayuda a encontrar la ruta más corta entre dos estaciones de conmutación.

j) El algoritmo funciona de la siguiente manera:

1) Desde el nodo A podemos conectarnos directamente al nodo B, C y D.
2) El ancho de banda disponible entre los nodos A y B es BW (A, B) = 9.
3) El ancho de banda disponible entre los nodos A y C es BW (A, C) = 19.
4) El ancho de banda disponible entre los nodos A y D es BW (A, D) = 21. Como nuestro requisito es el mayor ancho de banda, seleccionamos el nodo D.
5) Ahora desde el nodo D podemos conectarnos al nodo E y al nodo C.
6) El ancho de banda disponible entre los nodos D y C es BW (D, C) = 5.
7) El ancho de banda disponible entre los nodos D y E es BW (D, E) = 25.
8) En este caso, el ancho de banda más alto provisto entre D y E es el camino más corto.
9) Por lo tanto, la ruta sería A-D-E y el ancho de banda disponible será A + D + C = 21 + 25 = 46.

k) Así es como funciona el algoritmo de dijsktra para la red telefónica.

3. Aplicación en Google Maps:

Necesitamos obtener la latitud y longitud de origen (usuario) y la latitud y longitud de destino (objetivo) para poder calcular la ruta más corta utilizando el algoritmo de Dijkstra:

- Si los vértices (nodos) del gráfico representan ciudades y los pesos de borde representan distancias de conducción entre pares de ciudades conectadas por una carretera directa, el algoritmo de Dijkstra se puede utilizar para encontrar la ruta más corta entre dos ciudades. Además, este algoritmo se puede utilizar para la ruta más corta al destino en la red de tráfico.
- Usaremos este gráfico junto con el conocido algoritmo de dijkstra para encontrar el camino más corto entre dos puntos en un mapa.
- Para hacer esto, se mapeo puntos de intersección en una pequeña sección del mapa de la ciudad, luego calcularé la distancia más corta entre esos conjuntos de puntos usando nuestro algoritmo.

Ejemplo:

A continuación se muestra el mapa basado en una pequeña sección de Williamsburg en Brooklyn, NY. Todos los círculos representan vértices del gráfico y las líneas negras con flechas son bordes dirigidos con pesos asignados a cada uno de ellos. Los círculos verdes serán nuestros puntos de destino.

Figura 7: Resultado de las instrucciones del algoritmo de Dijkstra.

Basándonos en ese mapa, se crea un archivo de entrada que llenará el gráfico con bordes y vértices. Usaremos este archivo como datos de entrada de nuestro algoritmo.

Datos de ejemplo:

1 4 5.0
4 3 10.0
3 2 5.0
2 3 5.0
2 1 10.0
4 5 2.0
6 5 10.0
6 3 2.5
...

Del ejemplo anterior, cada fila se asigna al método de firma agregado (desde a, peso). La implementación del gráfico utilizará un enfoque de lista de adyacencia para almacenar información sobre los bordes. Intenté simplificar el código para que sea fácil de leer, por lo tanto, esta no es una implementación óptima y definitivamente podría mejorarse.

Algorithm 4 Código en java

```
import java.io.IOException;
import java.util.*;
public class DijkstraFind {
 private int size;
private HashMap¡Integer, Double¿weight; // store weights for each vertex
private HashMap¡Integer, Integer¿previousNode; // store previous vertex
private PriorityQueue¡Integer¿pq; // store vertices that need to be visited
private WeighedDigraph graph; // graph object
/**
* Instantiate algorithm providing graph
*
* @param graph WeighedDigraph graph
*/
 public DijkstraFind(WeighedDigraph graph) {
this.graph = graph;
size = graph.size();
}
/**
 * Calculate shortest path from A to B
*
 * @param vertexA source vertex
 * @param vertexB destination vertex
 * @return list of vertices composing shortest path between A and B
 */
public ArrayList¡Integer¿shortestPath(int vertexA, int vertexB) {
previousNode = new HashMap¡Integer, Integer¿();
weight = new HashMap¡Integer, Double¿();
pq = new PriorityQueue¡Integer¿(size, PQComparator);
 /* Set all distances to Infinity */
for (int vertex : graph.vertices())
```

La continuación del código se encuentra en el repositorio del grupo «https://github.com/franss2409/-minimal-path-algorithms-in-graphs»

Ahora veamos qué tan bien funcionó nuestro algoritmo en comparación con las instrucciones de Google Map. A continuación se muestra la imagen con los resultados anteriores trazados en el mapa. Y aquí están los resultados de Google Map :

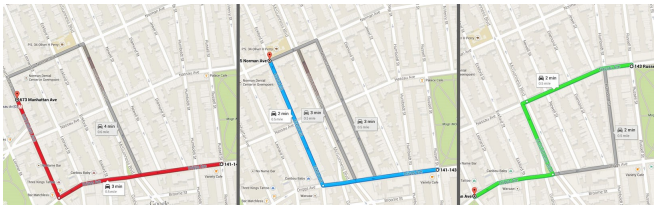

Figura 8: Resultado de las direcciones de Google Maps.

VI. Discusión y Resultados

La comparación del rendimiento del módulo de rutas frente a otras metodologías que resuelven el mismo problema se presentan a continuación en la siguiente tabla.

COMPARACIÓN ENTRE LOS TIEMPOS DE EJECUCIÓN				
	kilómetros	Tiempo medio de la implementación convencional (s)	Tiempo medio con la mejora (s)	Diferencia de tiempos (s)
Ruta corta	9,195	12,34	0,05 (0,4%)	12,29
Ruta media	54,82	16,7	0,17 (1,01%)	16,53
Ruta Larga	192,96	61,23	0,53 (0,865%)	60,7

Figura 9: Tabla que muestra la comparación de tiempos entre la implementación de algoritmos convecionales y el Dijkstra implementado en el proyecto.

Según se muestra en la tabla las mejoras implementadas respecto al algoritmo convencional suponen una mejora importantísima del tiempo de procesamiento, con ellas el algoritmo se convierte en una aplicación con utilidad para solventar problemas de manera inmediata. Sin embargo, si no se hubieran aplicado estas mejoras el algoritmo seguiría produciendo soluciones igualmente óptimas pero la validez de los resultados ya no sería la misma, puesto que la lentitud en encontrar la solución a una ruta que se quiere llevar a cabo de manera inmediata le resta eficiencia en el procesamiento de la información.

TIEMPOS DE EJECUCIÓN ALGORITMO DIJKSTRA DINÁMICO				
	kilómetros	Tiempo mínimo (ms)	Tiempo máximo (ms)	Tiempo medio (ms)
Ruta corta	9,195	47	54	50
Ruta media	54,82	156	185	166
Ruta Larga	192,96	485	576	522

Figura 10: Tiempos de ejecución del módulo de rutas implementado.

En la tabla anterior se pueden observar los tiempos de ejecución del algoritmo, para el cálculo de rutas dependiendo de la distancia desde el nodo origen al nodo destino. Para estimar los tiempos medios se han realizado 4 ejecuciones de cada ruta en las cuales el nodo origen y el nodo destino son siempre los mismos. Por otra parte los tiempos se encuentran

expresados en milisegundos de lo que se desprende que la aplicación posee buena calidad de resultados en referencia al tiempo necesario para obtenerlos. Por otra parte, es importante notar que el tiempo de procesamiento de un nodo siempre es el mismo, pero los tiempos de ejecución aumentan directamente con la distancia porque el número de nodos que deben ser examinados también lo hace.

A continuación se va a realizar un estudio de los tiempos medios de procesamiento por nodo, para ello se han realizado pruebas en las que se comprueba el número de nodos etiquetados permanentemente frente al tiempo que el algoritmo ha necesitado para ello. Asimismo se han realizado pruebas resolviendo el one-to-all SPP con el objetivo de que el algoritmo resolviera el SPP a los 123705 nodos que posee la red para comprobar el tiempo de ejecución. Los resultados obtenidos se muestran a continuación:

TIEMPOS MEDIOS DE PROCESAMIENTO POR NODO				
Nodos etiquetados permanente	**Tiempo mínimo(ms)**	**Tiempo máximo (ms)**	**Tiempo medio (ms)**	**Tiempo medio de procesamiento por nodo (ms)**
3253	82	112	90	0,0276667
41221	485	576	522	0,01266344
123705	2556	3255	2830	0,022877
			Tiempo medio (ms)	0,021069

Figura 11: Tiempos medios de procesamiento por nodo.

VII. Conclusiones

1. Como podemos ver,el algoritmo se logró implementar de una manera vérsatil en distintas aplicaciones:red telefónica,sistemas de seguridad y en google maps,lo que nos permite verficar su gran utilidad y rapidez al resolver problemas que requieren procesamiento de datos y optimización.
2. Con el algoritmo de Dijkstra implementado se ha obtenido mejora en el procesamiento de la información,tal como se puede observar en detalle en las tablas mostradas;es decir, podemos procesar gran cantidad de información en tiempo real y puesto que la velocidad de las variaciones no perjudica a los resultados se vuelve el más recomendable para realizar la optimización de rutas que otros algoritmos de reducción mencionados en el presente trabajo.

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.