EFICIENCIA ALGORÍTMICA EN APLICACIONES DE GRAFOS ORIENTADO A REDES GMPLS

Octavio Salcedo Parra
Magister en Ciencias de la Información y las Comunicaciones
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
ojsalcedop@udistrital.edu.co

Simar Herrera
Especialista en Ingeniería de software
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
seherreraj@udistrital.edu.co

Tipo: Artículo de revisión

Fecha de Recepción: Octubre 15 de 2011
Fecha de Aceptación: Noviembre 15 de 2011

EFFICIENCY OF GRAPH ALGORITHMS IN APPLICATIONS ORIENTED GMPLS NETWORKS

ABSTRACT
The algorithms used in the development and application of graphs, make use of measurable resources in time and space, the study of these costs is known as algorithmic complexity, the random use of any algorithm is frecuently made without performing an analysis of the same in the environment where they´re going to be executed, The aim of this study is an algorithmic analyze in common environments, in order to generate statistics that show the suitability of the use of specific algorithms.

Keywords: algorithm, complexity, efficiency, graph, optimization.

RESUMEN
Los algoritmos utilizados en el desarrollo y aplicación de grafos hacen uso de recursos medibles en tiempo y espacio, al estudio de estos costos se le conoce como complejidad algorítmica, frecuentemente se hace uso de cualquier algoritmo al azar sin realizar un análisis de los mismos en el ambiente en que se ejecutarán, el objetivo del presente es hacer un análisis algorítmico en ambientes comunes, con el fin de generar estadísticas que evidencien la conveniencia del uso de algoritmos específicos.

Palabras claves: algoritmo, complejidad, eficiencia, grafo, optimización.

1. INTRODUCCIÓN

Para trabajar con algoritmos relacionados con grafos y sus aplicaciones, es necesario tener en cuenta las ramas de las ciencias de la computación que prestan las herramientas para realizar un análisis detallado describiendo medidas en tiempo y espacio para evaluar la eficiencia de los mismos.

2. ANÁLISIS ALGORÍTMICO

El análisis de algoritmos se puede definir como el estudio que se realiza sobre un algoritmo para determinar si su rendimiento y comportamiento cumple con los requerimientos [15], adicionalmente permite considerar esta información para determinar la eficiencia del mismo. El objetivo del análisis de algoritmos es cuantificar las medidas físicas: "tiempo de ejecución y espacio de memoria" y comparar distintos algoritmos que resuelven un mismo problema [2]. En el análisis de algoritmos es necesario tener en cuenta inicialmente, que los algoritmos construidos deben ser correctos, es decir, deben producir un resultado deseado en tiempo finito. Los criterios para realizar esta evaluación pueden ser eficiencia, portabilidad, eficacia, robustez, etc [3]. El concepto de eficiencia de un algoritmo es relativo, dado que ante dos algoritmos que resuelven el mismo problema, uno es más eficiente que otro si consume menos recursos, presentándose que algunos dan una eficiencia en tiempo pero con un consumo alto de recursos, o por el contrario un uso óptimo de recursos, pero con un tiempo un poco más largo. De acuerdo con esto, en muchas ocasiones es bastante útil poder predecir cómo se comportará un algoritmo sin llegar a su implementación, es decir, analizar el algoritmo matemáticamente.

3. COMPLEJIDAD ALGORÍTMICA

La teoría de la complejidad computacional es la parte de la teoría de la computación que estudia los recursos requeridos durante el cálculo para resolver un problema [16]. Dado que en las ciencias de la computación los algoritmos son la herramienta más importante que se presenta, deben dar solución a diferentes problemas, con pasos concretos, claros y finitos. Cada algoritmo arroja un cálculo correcto a través de la recepción de datos de entrada y la generación de información de salida [5]. El análisis de complejidad de un algoritmo produce como resultado una "función de complejidad" [21] que da una aproximación del número de operaciones que realiza [17]. Cada algoritmo se puede medir en tiempo y espacio, el tiempo y recursos de máquina que requieren para su ejecución, características que cuando el algoritmo crece hacen necesaria una medición más exacta y apropiada, para esto se realizan ciertas operaciones matemáticas que identifican la eficiencia teórica del programa, a estos estudios se les denomina complejidad algorítmica [2].

3.1. Notación sintótica

Dentro del análisis de complejidad existen factores constantes que son poco relevantes y pueden ser omitidos en la comparación de tasas, para tal fin es que se utiliza la notación asintótica [6].

La eficiencia de un algoritmo se puede definir como una función *t(n)* [20]. Al analizar un algoritmo, lo relevante es el comportamiento cuando se aumenta el tamaño de los datos, esto se conoce como eficiencia asintótica de un algoritmo.

Para describir la notación asintótica se hace uso de las matemáticas (ecuación (1)), definiéndola como función para cual los números naturales N son su dominio [1].

$$o(f)\left\{ g: N \to R^{+} \;\middle|\; \begin{array}{c} \exists c \in R,\ \exists n_0 \in N: \forall n \geq n_0 \\ g(n) \leq cf(n) \end{array} \right. \quad (1)$$

Algunas reglas sobre esta notación son las siguientes [8]:

- *O(C*g)=O(g), C es una constante.*
- *O(f*g)=O(f)*O(g), y viceversa.*
- *O(f/g)=O(f)/O(g), y viceversa.*
- *O(f+g)=función dominante entre O(f) y O(g).*

3.1. Divide y conquistaras

En este contexto, divide y vencerás es una técnica de diseño de algoritmos que consiste en resolver un problema a partir de la solución de sub-problemas del mismo tipo, pero de menor tamaño. Si los sub-problemas son todavía relativamente grandes se aplicará de nuevo esta técnica hasta alcanzar sub-problemas lo suficientemente pequeños para ser solucionados directamente. Ello naturalmente sugiere el uso de la recursión en las implementaciones de estos algoritmos [7].

La resolución de un problema por medio de esta técnica se da a través de los siguientes 3 pasos como mínimo:

1. División: identificar los pequeños sub-problemas del mismo tipo del problema original y organizarlos en los k diferentes grupos.
2. Solución Sub-problemas: debe solucionarse de manera independiente todos los sub-problemas que estrictamente son de menor tamaño bien sea de forma directa o de forma recursiva.
3. Solución problema: dada una solución de los sub-problemas, articular estas soluciones para la construcción de la solución del problema en general.

3.2. Ordenes de complejidad

Utilizando la notación asintótica, podemos definir un "orden de complejidad" básico [8], de esta forma enumeramos lo siguiente:

- *O(1) Orden Constante.*
- *O($log_2 n$) Orden logaritmico.*
- *O(n) Orden líneal.*
- *O(n $log_2 n$).*
- *O(n^2) Orden cuadrático.*
- *O(n^α) Orden polinomial (α>2).*
- *O(α^n) Orden exponencial (α>2).*
- *O(n!) Orden factorial.*

3.3. Recurrencias

En argumentos lógicos o en algoritmos, se presenta la necesidad de resolver una sucesión de casos, para lo cual a nivel matemático normalmente se busca estructurar la conexión de cada caso con el anterior. La recurrencia es un método astuto que busca enlazar cada caso con el anterior es decir generalizar el procedimiento y que al resolver solo el último caso nos permita encontrar la respuesta de los demás, es decir al final sólo nos quedará un primer caso por resolver, del que se deducirán todos [9].

En algunos ejemplos se encuentra el cálculo de series, sucesiones o recorridos, que inicialmente son problemas iterativos, no obstante; a través del estudio o análisis de complejidad, es posible plantear una ecuación de recurrencia. Uno de los ejemplos nombrados y conocidos, es el cálculo de la serie de Fibonacci.

1-1-2-3-5-8-13-21-...

Dentro del manejo dado de las ecuaciones de recurrencia al llevarlas a un proceso algorítmico aparece una restricción y es que mientras se resuelve el último caso, todo el proceso está haciendo uso de un gran espacio de memoria y genera conflictos en ejecución, por tal motivo antes de hacer uso de esta técnica, siempre es necesario tener presente esto y tomar la decisión de combinarlo con otros métodos de solución o presentar otra posible solución desde el principio.

4. PROGRAMACIÓN DINÁMICA

Cuando se trabaja con algoritmos que buscan la optimización de procesos, más aún cuando se trabaja con algoritmos de representación en grafos, generalmente después de realizar el análisis correspondiente se encuentra que hay necesidad de utilizar un método más avanzado, por tal motivo se hace uso de la programación dinámica [22].

Es un método general de optimización de procesos de decisión por etapas, adecuado para resolver problemas cuya solución puede caracterizarse recursivamente y en la que los sub-problemas que aparecen en la recursión se solapan de algún modo, lo que significaría una

repetición de cálculos inaceptable si se programa la solución recursiva de manera directa [10].

En ocasiones las soluciones presentadas por otros métodos divisan un inconveniente y es cuando cada uno de los sub-problemas se solapa entre sí, impidiendo la solución de forma independiente de cada uno de ellos, mostrando que la recursividad no resulta eficiente por la repetición de cálculos que conlleva. En estos casos la Programación dinámica ofrece una solución aceptable, la eficiencia de esta técnica consiste en resolver los sub-problemas una sola vez, guardando las soluciones para su futura utilización [10].

La programación dinámica no sólo tiene sentido aplicarla por razones de eficiencia, sino porque además presenta un método capaz de resolver de manera eficiente problemas cuya solución ha sido abordada por otras técnicas y ha fracasado [19].

La programación dinámica tiene mayor aplicación en la resolución de problemas de optimización. Estos problemas generalmente presentan distintas soluciones y lo que busca esta técnica es encontrar la solución de valor óptimo.

La solución de problemas mediante esta técnica se basa en el llamado principio óptimo enunciado por Bellman en 1957 y dice:

"En una secuencia de decisiones óptimas, toda sub-secuencia ha de ser también óptima" [7].

En grandes líneas, el diseño de un algoritmo de programación dinámica consta de los siguientes pasos:

1. Planteamiento de la solución como una sucesión de decisiones y verificación de que ésta cumple el principio óptimo.
2. Definición recursiva de la solución.
3. Cálculo del valor de la solución óptima mediante una tabla en donde se almacenan soluciones a problemas parciales para reutilizar los cálculos.
4. Construcción de la solución óptima haciendo uso de la información contenida en la tabla anterior.

4.1. Funciones con memoria

En la programación dinámica la solución se basa en tener la solución de pequeños sub-problemas más grandes, construyendo un árbol que nos lleve a la solución general del problema [10].

func CombinacionesFM(m,n) return natural
if T(m,n) ≠0 then return T(m,n)
if n =0 ∪ n =m then
T(m,n)←1
return T(m,n)
{n≠0 ∩n ≠m ∩T(m,n) no calculado}
T(m,n)←CombinacionesFM(m-1,n-1)+
CombinacionesFM(m-1,n)
return T(m,n)

5. ALGORITMOS DE GRAFOS

Generalmente el diseñador del algoritmo o programador, busca que cada función sea resuelta de forma eficiente y a través del análisis que se puede realizar de la forma descrita en el presente documento definir cuál es el método a utilizar. Al afrontar problemas que tienen una necesidad de optimización y su estructura se puede definir como un grafo cualquiera, la combinación del desarrollo de estructuras de datos, algoritmos de recorridos, algoritmos de búsquedas y la programación dinámica indican una cantidad de algoritmos ya estudiados y utilizados por su optimalidad.

Dentro de esos algoritmos se encuentran los siguientes [11]:

1. Algoritmo de Dijkstra: orden de complejidad $O(|V|^2+|E|)$, orden que genera una respuesta que en problemas complejos de este tipo, tiene un tiempo de ejecución de no más de 1 s.
2. Algoritmo de Kruskal: orden de complejidad $O(log|n|)$
3. Algoritmo de Floyd-Warshall: orden de complejidad $O(n^3)$
4. Algoritmo de Prim: orden de complejidad

$O(V^2)$,$O(E \log V)$,$O(E \mid V \mid \log V)$.

5. Algoritmo de Bellman-Ford: orden de complejidad *O(E).*
6. Algoritmo de Ford-Fulkerson: orden de complejidad $O(VE^2)$.

En diferentes ámbitos cada uno de estos algoritmos presentan un excelente rendimiento y lo ideal es conocer en que ámbitos funcionan mejor cada uno de ellos.

5.1. El algoritmo de Dijkstra

En la complejidad, uno de los clásicos problemas es encontrar la ruta más corta entre un vértice inicial y cualquiera de los vértices de un grafo dado. El algoritmo de Dijkstra presenta una solución por etapas al estilo de la programación dinámica, cada etapa añade un nuevo vértice al conjunto de vértices a los que se les conoce su distancia al origen. Cada ruta obtenida se basa en tomar el camino óptimo, si para ir de vi a v_j es necesario pasar por v_k, los caminos v_i a v_k y v_k a v_j, han de ser mínimos.

Una aproximación del algoritmo de Dijkstra es:

```
public class DijkstraEngine {
   public int[] ejecutar(int[][] grafo, int nodo) {
      final boolean visitados[] = new
boolean[grafo.length];
      final int distanciasCortas[] = new int[grafo.
length];
      visitados[0] = true;
      for (int i = 1; i < distanciasCortas.length;
i++){
         if (grafo[nodo][i] != 0) distanciasCortas[i]
= grafo[nodo][i];
         else distanciasCortas[i] = Integer.MAX\_
VALUE;
      }
      for (int i = 0; i < (distanciasCortas.leng-
th-1); i++){
         final int siguiente = proximoVertice(dista
nciasCortas, visitados);
         visitados[siguiente] = true;
         for (int j = 0; j < grafo[0].length; j++) {
            final int d = distanciasCortas[siguiente]
+ grafo[siguiente][j];
            if ( distanciasCortas[j]
> d) distanciasCortas[j] =
distanciasCortas[siguiente] + grafo[siguiente]
[j];
         }
      }
      return distanciasCortas;
   }

   private static int proximoVertice (int []distan-
ciasCortas, boolean []visitados) {
      int x = Integer.MAX_VALUE;
    int y = -1;
  for (int i = 0; i < distanciasCortas.length; i++){
       if (!visitados[i] && distanciasCortas[i] < x){
          y = i;
          x = distanciasCortas[i];
       }
      }
      return y;
   }
} [4]
```

Este código presenta un orden de complejidad de $O(|V|^2+|E|)$ sin utilizar cola de prioridad, o $O(|E|+|V| \log|V|)$, si se utiliza cola de prioridad. Presenta una solución eficiente en cuanto a un camino entre dos nodos se refiere.

5.2. El algoritmo de Floyd

Continuando el trabajo con grafos, otra propuesta algorítmica es presentada, cuando se necesita indicar cuál es el camino más corto entre cualesquiera par de nodos el algoritmo Floyd, que dada la matriz L de adyacencia del grafo g, calcula una matriz D con la longitud del camino mínimo que une cada par de vértices. Una posible implementación para este se muestra a continuación:

```
static int [ ][ ]floyd;
for (int k=0;k<=n-1;k++)
{
for (int i=0;i<=n-1;i++)
{for (int j=0;j<=n-1;j++)
if ((floyd[i][k]!=-1)&&(floyd[k][j]!=-1))
floyd[i][j]=funcionfloyd(floyd[i][j],floyd[i]
[k]+floyd[k][j]);}
}
```

```
public static int funcionfloyd(int A, int B)
{
if ((A==-1)&&(B==-1))
return -1;
else if (A==-1)
return B;
else if (B==-1)
return A;
else if (A>B)
return B;
else return A;
}
```
[4]

La complejidad de este algoritmo es $O(n^3)$, el algoritmo resuelve eficientemente la búsqueda de todos los caminos más cortos entre cualesquiera nodos.

6. MPLS

Label Switching, se basa en asociar una pequeña etiqueta de formato fijo con cada paquete de datos para que pueda ser enviado a través de la red. Esto significa que cada paquete, ventana o celda, debe tener adicionalmente algún identificador que indica los nodos de la red por los cuales debe pasar.

En cada salto a través de la red el paquete está enviándose basado en el valor de la etiqueta de entrada y reenviado con un nuevo valor de etiqueta. La etiqueta es intercambiada y los datos son conmutados basados en el valor de la etiqueta, dando lugar a dos términos: "label swapping" y "label switching".

En una red MPLS, los paquetes son etiquetados por la inserción de una pieza adicional de información llamada shim header. Esto funciona entre la cabecera de red y la cabecera de IP [26].

7. DIJKSTRA Y GMPLS

El algoritmo Dijkstra presenta algunas modificaciones, y se ha generado sobre el mismo una gran cantidad de aplicaciones dentro de las cuales podemos nombrar las siguientes:

- Encaminamiento de paquetes por los routers.
- Aplicaciones para sistemas de información geográficos.
- Reconocimiento del lenguaje hablado.
- Enrutamiento de aviones, tráfico aéreo.
- Tratamiento de imágenes médicas.

Cada una de estas aplicaciones tiene su razón de ser, por ejemplo en el encaminamiento de paquetes se da que un mensaje puede tardar cierta cantidad de tiempo en atravesar cada línea. En este caso, tenemos una red con dos nodos especiales, el de inicio y el de llegada. Los pesos de las aristas serían los costes. El objetivo del algoritmo es encontrar un camino entre estos dos nodos cuyo coste total sea el mínimo. Una de las aplicaciones más claras en este tema, se da a través de la definición de protocolos como GMPLS [23].

Dentro de las mejores aplicaciones que se dan al uso de soluciones dadas a través de redes neuronales. El reconocimiento de lenguaje hablado el cual tiene dentro de sí mismo múltiples aplicaciones de grafos que en conjunto ayudan a generar un resultado más efectivo [25].

Dentro de los procesos de desarrollo para el ruteo de paquetes también podemos encontrar que otra aplicación se da con la idea de garantizar la calidad del servicio QoS [24], a través de la minimización en el uso de la red y la flexibilización del modelo de reservas.

El MPLS generalizado o GMPLS, es una especificación del MPLS, que busca eliminar algunos inconvenientes presentados dentro del transporte de información en las redes. Los inconvenientes se presentan en hardware, software o configuración y se tratan de eliminar por medio de este protocolo [26]. Dentro de la definición de GMPLS encontramos el uso de algunos de los algoritmos ya nombrados como lo son:

- Bellman-Ford.
- Dijkstra.
- Dijkstra modificado.
- Bread first search.
- Johnson.
- K Shortest Paths.

Estos algoritmos son nombrados en otros documentos que muestran el uso de los mismos, no se realizan comparaciones entre los mismos dado que no cumplen exactamente la misma función, son algoritmos que en términos generales realizan la misma tarea, pero cada uno tiene su especialidad y es eficiente en ella [28]. La mayoría de documentos que hacen referencia a estos algoritmos en conjunto, son documentos que presentan posibles usos de los mismos en una solución conjunta, entre ellos está el uso dado en las redes como en GMPLS.

7.1. Bellman-Ford

Este es un algoritmo que resuelve el problema dando a un vértice cualquiera la posibilidad de encontrar todos los caminos mas cortos a cualquiera de los vértices del grafo. En términos generales este algoritmo es usado off-line, es decir es utilizado para realizar un mapa de todas las posibles rutas que se puede tomar desde un punto a los demás, esto es aplicable a todos los dispositivos de ruteo locales que se utilizan en la red.

```
algorithm ExtendedBellmanFord(s,d)
{
initialize st(*) = st(0, *);
// compute st(*) = st(n − 1, *)
put the source vertex into list1;
for (int k = 1; k < n; k++)
{
// see if there are vertices whose
 st value has changed
if (list1 is empty)
break; // no such vertex
while (list1 is not empty){
delete a vertex v from list1;
foreach (edge (v,u)){
st(u) = st(u) ∪ {st(v) ∩ S T (v,u)};
if (st(u) has changed and
u is not on list2) add u to list2;
}
list1 = list2;
make list2 empty;
}
}
}
```

7.2. Dijkstra

Aunque este algoritmo también resuelve el problema de encontrar todos los caminos desde un vértice dado a todos los demás, es muy eficiente encontrando el camino más corto entre una par dado de vértices. Cuando se tiene un paquete de información y se desea construir el camino más corto entre el emisor y el receptor en la red es más económico y rentable utilizar este algoritmo para plantear la ruta.

7.3. Dijkstra modificado

El algoritmo de Dijkstra presenta algunos fallos para grafos, donde algunos arcos tienen pesos negativos. La razón de esto es que un vértice removido de la cola de prioridad U no vuelve a ser re etiquetado ni reinsertado dentro de U. En grafos con arcos no negativos esto funciona, sin embargo cuando se presenta arcos con peso negativo el algoritmo Dijkstra en su estado original presenta fallas para encontrar los estados o las rutas m\'as cortas, por tal motivo se hace necesario presentar una alternativa, que podría ser cambiar de algoritmo o modificar el Algoritmo de Dijkstra re etiquetando los vértices removidos o reinsertándolos en la cola de prioridad. Este algoritmo ha sido planteado de la siguiente manera:

```
do for every v ϵ V
d[v]=∞; π[v]=NIL
d[s]=0
L=0, U=V
do while U!=0
  u = EXTRACT_MIN_KEY_ENTRY(U)
  L=L+u
  do for each arc a(u,v) ∈ Originating(u)
  if d[v] = d[u] + w(a) then
  d[v] = d[u] + w(a), π[v]=u
  if v ϵ U
  then DECREASE_ENTRY_KEY(U,v)
  else L=L-v, INSERT_ENTRY(U,v)
```

Orden de complejidad: $O(|V|^2+|E|)$ teniendo unas modificaciones sobre el algoritmo de Dijkstra que se pueden clasificar como insignificantes dado que solo se agrega procedimientos constantes al mismo.

7.4. Breadth first search

Este algoritmo baja en eficiencia dentro de los algoritmos que construyen todos los caminos más cortos desde un punto dado hasta los demás, sin embargo, previene que dentro del proceso no se encuentren ciclos negativos, ya depende del uso de la red y como se necesite el revisar la rentabilidad de su uso.

```
ExtendedBreadthFirstSearch(s,d,prev)
{
Label vertex s as reached.
Initialize Q to be a queue with only s in it.
while (Q is not empty)
{
Delete a vertex w from the queue.
Let u be a vertex (if any) adjacent from w.
while (u ≠ null)
{
if (u has not been labeled and edge (w, u) has
bandwidth b or more available from
time tstart to tend)
{
prev[u] = w;
if (u == d) return;
Label u as reached.
Add u to the queue.
}
u = next vertex that is adjacent from w.
}
}
}
```
[27]

7.5. Johnson

Este es un algoritmo muy completo, que permite encontrar todos los caminos más cortos entre cada par de vértices presente en el grafo, es utilizado con el fin de realizar un mapa completo de la red, haciendo evidente el posible uso de segundas rutas más cortas. Es un algoritmo más para uso off-line, es decir dada su implementación es más para redes con un cambio en sus dispositivos conectados mínimo.

7.6. K Shortest Path

No siempre los caminos más cortos son los más efectivos, el K Shortest Path, busca los primeros k caminos más cortos entre un par dado de vértices, orientándonos en otras posibles rutas, que pueden ser más cortas, y dentro de estas no se incluya la primera, ya que en este caso puede presentarse una congestión sobre la misma.

8. MODELOS Y DISCUSIONES ACERCA DE LA APLICACIÓN EN GMPLS

Cada uno de los algoritmos tratados en el presente documento presenta una aplicabilidad en diferentes ámbitos, las redes GMPLS (figura 1) tienen heurísticas que se asocian con los algoritmos de grafos y se han planteado a lo largo del documento.

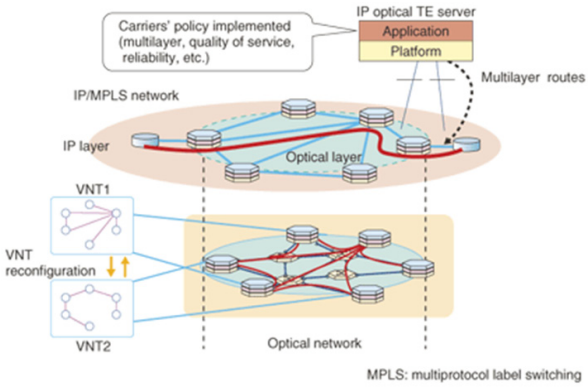

Figura 1. Modelo MPLS.

Inicialmente cada uno de los algoritmos tiene su aplicación dentro del modelo GMPLS, es posible hacer un resumen de sus aplicabilidades.

8.1. Bellman-Ford

Algoritmo que presenta propiedades interesantes en el sentido que es un algoritmo que considera las restricciones de enrutamiento, es decir identifica el camino óptimo entre el origen y el destino a través de máximo h saltos. Utilizado en RIP que es la implementación más popular de este tipo de algoritmos [29]. Como es posible notar en la sección 7.1, este algoritmo presenta en el mejor de los casos un Orden de su número de aristas, sin embargo también es posible que el orden cambie a un máximo de su número de aristas por su número de vértices es decir $O(E*V)$.

- for (*int k = 1; k < n; k++*) identifica un ciclo con el fin de recorrer y analizar cada uno de los vértices.
- if (*list1 is empty*) identifica una condición que permite bajar el tiempo de ejecución.
- foreach (*edge (v,u)*) ciclo que revisa cada uno de los arcos y esta anidado en el anterior ciclo nombrado.

De esta manera se identifica el orden de V*E.

8.2. Dijkstra's algorithms

Cada router GMPLS sirve como la base para el descubrimiento de caminos, cálculos realizados por medio de estos algoritmos, cada nodo de la red realiza esta tarea cada vez que recibe un mensaje a enrutar. La complejidad de estos algoritmos es muy baja cuando se trata de solo calcular el camino más corto entre un par dado de vértices, revisando los algoritmos secciones 5.1 y 7.3, el orden para un solo par de vértices es de máximo O(E), el orden se extiende por los ciclos for anidados en el conteo de vértices hasta un máximo de O(|V|^2+|E|), siendo este el orden para calcular todas las rutas mínimas dadas desde un punto dado a todas las demás ubicaciones.

- for (*int i = 0; i < (distanciasCortas.length - 1); i++*) Ciclo que recorre todos los vértices identificando las rutas más cortas de cada uno.
- for (*int j = 0; j < grafo[0].length; j++*) Este ciclo esta anidado al anterior recorriendo las posibles rutas más cortas y dando el orden máximo de V*V.

8.3. Breadth first search

Algoritmo utilizado en el plano de control del GMPLS, posee una característica especial en este control y es que dados dos caminos cortos en peso él toma como óptimo el que tiene menor número de arcos [26], el algoritmo presentado en la sección 7.4 muestra un orden en el cual los ciclos se basan en la cantidad de vértices y una cola de prioridad, presentando un orden *O(nLog |n|)*.

- while (Q is not empty) ciclo que recorre n vértices en la cola de prioridad.
- while (u ≠ null) este ciclo esta anidado al anterior y con una condición hace que el rango se disminuya en cada acceso que se tiene a este ciclo.
- if (u has not been labeled and edge (w, u) condición que permite reducir el tiempo de ejecución de forma proporcional a los nodos recorridos.

8.4. K shortest path

En los enrutadores de las redes GMPLS, se tiene control en la perdida de información necesitando como opción la selección de caminos que no necesariamente son los más cortos pero son más seguros y eficientes, el k-shortest path calcula los primeros k caminos entre dos nodos con el fin de definir cuál le genera mejor seguridad.

- Este es un algoritmo al cual no se le define un orden dado que maneja el uso de otros algoritmos para obtener cada camino más corto.
- Generalmente es usado el algoritmo de dijkstra para obtener el camino más corto y se elimina esta ruta como opción.
- El algoritmo genera un orden aproximado de k veces O(V*E).

El análisis de cada uno de estos algoritmos se ha realizado con el fin de mostrar cada uno de los órdenes de complejidad que tiene cada uno de estos algoritmos, también plantear la discusión de que no se debe elegir entre ellos, cada uno cumple funciones específicas en muchos de los campos de acción y es bastante importante la labor que cumplen cada uno de ellos dentro del modelo de la red.

A pesar de que en las redes GMPLS se utilizan todos estos algoritmos en su forma original [26], algunos autores prefieren realizar una construcción de nuevos algoritmos que a nivel de investigación muestran un mejor resultado, esta construcción la hacen con el fin de mejorar algunas características de las redes [27], sin desconocer que los algoritmos aquí nombrados

dan solución al problema lo que hacen es dar solución a casos específicos añadiendo a ellos fragmentos de código específico.

9. CONCLUSIONES

Dentro de los temas tratados en este documento se generan unas discusiones acerca de los temas tratados por diferentes autores en la solución de problemas de grafos eficientes para determinadas situaciones.

El tema de Grafos presenta una gran cantidad de algoritmos, y es un tema que nos permite hacer un mejor análisis de los mismos, obteniendo así un estudio de complejidad y de esta manera, haciendo posible determinar de acuerdo a diferentes arquitecturas, qué algoritmo se debe usar en las situaciones presentadas.

De acuerdo con cada una de las características de los problemas y a las necesidades, se puede hacer uso de los diferentes algoritmos, los algoritmos presentes en el actual documento son eficientes en sus respectivos ámbitos.

Si el problema requiere de una solución óptima de camino más corto entre 2 nodos definidos, es posible aplicar Dijkstra o Floyd, sin embargo es más eficiente Dijkstra, dado que Floyd encuentra todos los caminos más cortos entre cualquier par de nodos.

Otras soluciones de caminos más cortos implican la generación de árboles recubridores que permiten reconstruir cualquier ruta desde un nodo específico. Si este es el requerimiento, es posible utilizar algoritmos de gran eficiencia como el algoritmo de Kruskal o el de Prim.

Cuando se habla de algoritmos que ayudan a optimizar la solución de problemas que se pueden representar en forma de grafos, no implica que se tenga que utilizar un solo algoritmo. Los pseudocódigos presentados en este documento, en su análisis presentan una eficiencia general muy buena, son los algoritmos mejor clasificados en la solución de problemas de optimización.

La eficiencia de los algoritmos es medible de acuerdo con la utilización de espacio y tiempo que requieren, a través del análisis es posible generar una función matemática que representa su eficiencia a esto lo debemos llamar análisis de complejidad algorítmica.

Así mismo es necesario recordar que existe una amplia gama de aplicaciones de estos estudios. Algunas de ellas están en el ámbito de las redes de información [18], en las redes de tráfico o en estados de planeación que sirven para cualquier estructura organizacional.

En GMPLS se presenta el uso de algunos algoritmos incluido la modificación del algoritmo de Dijkstra manteniendo su nivel de complejidad en un orden definido en el algoritmo de Dijkstra [29].

Los algoritmos aquí consignados no solo presentan una excelente eficiencia sino que a su vez es importante presentar que en otras comparaciones se hace uso de la mayoría de ellos en forma combinada para mejorar la ejecución y las aplicaciones que pueden tener estos mismos [28].

El orden de complejidad presentado en este documento para cada uno de los algoritmos obedece a pruebas realizadas sobre JAVA, en las cuales se evidencia que el algoritmo corre de acuerdo con esto, por lo que también es necesario tener en cuenta los mecanismos de lectura y escritura de la información para evitar aumentar los tiempos de ejecución.

Referencias Bibliográficas

[1] A. Duch; Análisis de Algoritmos. [En línea], Consultado en Junio 15 de 2011, disponible en: http://www.lsi.upc.edu/duch/home/análisis.pdf.

[2] E. Scali., R. Carmona; Análisis de algoritmos y complejidad. [En línea], consultado en Junio 10 del 2011, disponible en: http://ccg.ciens.ucv.ve/~esmitt/ayed/II-2011/ND200105.pdf, Venezuela, ND 2001 05

[3] T. Cormen, C. Stein, R. Rivest, C. Leiserson; Introduction to algorithms, 3rd McGraw-Hill Higher education, New York, 2009.

[4] S. Skiena, M. Revilla; Programming challengers, the programming contest training manual, Springer-Verlag, New York, 2003.

[5] H. Zenil, J. Delahaye; Un método estable para la evaluación de la complejidad algorítmica de cadenas cortas, Computational Complexity (CC), Information Theory (IT), 2011.

[6] J. Villalpando; Análisis asintótico con aplicación de funciones de Landau como método de comprobación de eficiencia en algoritmos computacionales, e-Gnosis, España, 2003.

[7] R. Guerequeta, A. Vallecillo; Técnicas de diseño de algoritmos, Universidad de Málaga, Segunda edición, España, 2000.

[8] J. Mañas, Análisis de algoritmos: Complejidad. [En línea], consultado en Septiembre 9 de 2010 disponible en: http://www.lab.dit.upm.es/~lprg/material/apuntes/o/index.html, Noviembre de 1997.

[9] P. Gallardo; Ecuaciones de recurrencia, notas de matemática discreta. [En línea], consultado en Noviembre 11 de 2010, disponible en: http:// www.uam.es/personal_pdi/ciencias/gallardo/md.htm/

[10] J. Bermúdez; Diseño de algoritmos, Universidad del País Vasco, España, 2008.

[11] E. Coto; Algoritmos básicos de grafos. [En línea], consultado en Julio 11 de 2010, disponible en: http://wwwciens.unv.ve/~ernesto/nds/CotoND200302.pdf, Venezuela, ND 2003 02.

[12] P. Fillottrani; Algoritmos y complejidad, algoritmos sobre grafos. [En línea], consultado en Octubre 18 de 2010, disponible en: http://www.cs.uns.edu.ar/prf/teaching/AyC09/clase15.pdf, 2009

[13] H. Carrasco; Grafos. [En línea], consultado en Enero 5 de 2011, disponible en: www.ganimides.ucm.lc.

[14] J. Hopcroft, R. Tarjan; Efficient algorithms for graph manipulation, Cornell University, Ithaca New York 14850, 2008.

[15] J. Velásquez; Introducción al análisis de algoritmos. [En línea], consultado en Enero 15 de 2011, disponible en: http://148.201.94.3:8991/F/?func=direct/&local_base=ite01&doc_number=0001116302.

[16] A. Cortés, Teoría de la complejidad computacional y de la computabilidad. [En línea], consultado en Agosto 8 de 2010, disponible en: http://sisbib.unmsm.edu.pe/BibVirtualData/publicaciones/risi/N1_2004/a14.pdf.

[17] I. Dora, C. León, C. Rodríguez, G. Rodríguez, A. Rojas; Complejidad algorítmica: de la Teoría a la Práctica. [En línea], consultado en Septiembre 10 de 2010, disponible en: http://bioinfo.uib.es/joemiro/aenui/procJenui/Jen2003/docomp.pdf, La Laguna, Tenerife.

[18] M. Amoretti, F. Zanichelli, G. Conte; Performance evaluation of advanced rouning algorithms for unstructured peer-to-peer networks, Procceding of the 1rt international conference on performance evaluation methodologies and tools (valuetools '06), ACM, New York, 2006.

[19] Y. Yuan, Z. Cao, Z. Hou, M. Tan; Dynamic programming field based environment learning and path planning for mobile robots, Intelligent Control And Automation (WCICA), IEEE 2010 8th World Congress on, pp.883-887, 7-9,China, 2010.

[20] P. Marbach, J. Tsitsiklis; A neurodynamic programming approach to admission control in ATM networks: the single link case, Acoustics, Speech, and Signal Processing, 1997.

ICASSP-97., 1997 IEEE International Conference on, pp.159-162 vol.1, 21-24, Italy, 1997.

[21] P. Singhal, R. Sharma; Dynamic programming approach for solving power generating unit commitment problem, Computer and Communication Technology (IC-CCT), 2011 2nd International Conference on, vol., no., pp.298-303, 15-17, Berlin 2011.

[22] T. Hu, T. Wu, J. Song, Q. Liu, B. Zhang; A new Tree structure for weighted dynamic programming based stereo algorithm, Image and Graphics (ICIG), 2011 Sixth International Conference on, pp.100-105. 12-15, Anhui, 2011.

[23] O. Parra, C. Manta, G. Rubio,; Dijkstra's algorithm model over MPLS/GMPLS, Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on, vol., no., pp.1-4, 23-25, China, 2011.

[24] S. Tanwir, L. Battestilli, H. Perros, G. Karmous; Dynamic scheduling of network resources whit advance reservations in optical grids. [En línea], consultado en Febrero 20 del 2011, disponible en: http://dx.doi.org/10.1002/new.680.

[25] P. Hegyi, M. Maliosz, A. Ladanyi, T. Cinkler; Shared protection of virtual private networks, 2003. (DRCN 2003), Proceedings. Fourth International Workshop on, vol., no., pp. 448-454, EE.UU, 2003.

[26] A. Farrel, I. Bryskin; GMPLS: architecture and applications, San Francisco: Elsevier/Morgan Kaufman, 2006.

[27] A. Al-Khwildi, H. Al-Raweshidy; A proficient path selection for wireless Ad Hoc routing protocol, Advanced Communication Technology, 2006, ICACT 2006. The 8th International Conference, vol.1, pp.599-604, 20-22, Hong Kong, 2006.

[28] W. Xiao, B. Hee, C. law, Y. Liang; Evaluation of heuristic path selection algorithms for multi-constrained QoS routing, Networking, Sensing and control, 2004 IEEE International Conference on, vol.1, no., pp. 112-116, Vol.1, 21-23, New York 2004.

[29] L. Nguyen; Routing and wavelength assignment in GMPLS-based 10 Gb/s Ethernet long haul optical networks with and without linear dispersion constraints. [En línea], consultado en Marzo 9 de 2011, disponible en: http://www.SRPublishing.org/journal/ijcns/, 2008, 2, 105-206.