UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

UNIDAD ACADEMICA DE LOS CICLOS
PROFESIONAL Y DE POSGRADO DEL
COLEGIO DE CIENCIAS Y HUMANIDADES

ESTUDIO Y ANALISIS DE ALGORITMOS DISTRIBUIDOS PARA HALLAR ARBOLES GENERADORES MINIMOS

TESIS DE POSGRADO

PARA OBTENER EL TITULO DE

MAESTRA EN CIENCIAS DE LA COMPUTACION

P R E S E N T A:

LIC. ISABEL CAJIAS DE LA VEGA

MEXICO, D. F. 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

LISTA DE FIGURAS

CAPITULO 1

INTRODUCCION

El presente trabajo se desarrolla fundamentalmente dentro del área de Algoritmos Distribuídos. Se aprovechan resultados de la Teoría de Gráficas ampliamente usados en el contexto de los Sistemas Distribuídos debido a la posibilidad de modelar éstos como una gráfica.

1.1 Justificación

Estudiaremos en concreto algoritmos para hallar un Arbol Generador Mínimo (AGM) de una gráfica que representa a un tipo específico de Sistema Distribuído: Una red de computadoras.

Estos algoritmos utilizan técnicas típicas en algoritmos distribuídos y que los hacen, por sí mismos, interesantes y dignos de estudio.

Nos centramos en particular en el análisis del algoritmo propuesto por Gallager, Humblet y Spira (GHS). Este algoritmo presenta características especialmente significativas que lo han hecho objeto de estudio como uno de los algoritmos fundamentales en el area de Algoritmos Distribuídos.

Por otra parte, hay que señalar la gran importancia que han cobrado los Arboles Generadores Mínimos (AGM) en el contexto de las redes de computadoras que a su vez son uno de los temas que suscitan mayor interés y estudio actualmente dentro de las Ciencias de la Computación. Un problema de comunicación frecuente en redes es realizar la difusión de información (broadcast) de un nodo a los demás de la forma más barata posible teniendo en cuenta que hay un costo de envío de mensajes asociado con cada línea de transmisión en la red. Proveyendo un AGM, podemos difundir la información con el menor costo total posible usando exclusivamente las líneas que forman el árbol.

Adicionalmente existen otros problemas de control de redes cuyas complejidades de comunicación se reducen teniendo un AGM tales como el encontrar un líder, contar el número de nodos en la red, ubicar el nodo con mayor identificador, etc. Notemos que tanto en éstos

como en otros problemas necesitamos simplemente un árbol generador (no obligatoriamente mínimo). Sin embargo, en estos casos podemos también aplicar los algoritmos para hallar un AGM pues no tienen mayor complejidad que los que encuentran un árbol generador cualquiera.

1.2 Contenido

Como señalamos, nos concentramos en el análisis y estudio exhaustivo del algoritmo GHS. Por exhaustivo entendemos una justificación completa y detallada de los argumentos teóricos y el desarrollo paso por paso del algoritmo, que servirá, esperamos, para enfatizar la amplia gama de ideas inteligentes existentes en su construcción, empezando en su filosofía de dividir y distribuir el problema y terminando en una serie de detalles más bien implementativos pero también esenciales (capítulo 6).

Asimismo, en el capítulo 7 se realiza el análisis del comportamiento del algoritmo de GHS en distintos modelos y en el capítulo 8 se describen modificaciones que han realizado sobre el mismo distintos autores a fin de mejorar la complejidad en tiempo o en relación a la tolerancia de fallas.

Para facilitar algunas comparaciones, construimos -en el capítulo 5- dos algoritmos para resolver el mismo problema (hallar el AGM de una red) utilizando una filosofía centralizada. Tratamos de llegar a un algoritmo lo más eficiente posible y de justificar las desventajas que presentaba trabajar sobre un esquema centralizado. Sin embargo no afirmamos en ningún momento que no haya soluciones mejores.

El capítulo 4 trata del problema y sus aplicaciones, concentrándose en la relación del problema de hallar un AGM con el de elección de líder.

La tesis es un trabajo autocontenido en cuanto a que incluye también la revisión de conceptos, aún los más básicos, de Teoría de Gráficas (capítulo 2) y Algoritmos Distribuídos (capítulo 3).

Por otro lado, se ha puesto cuidado en desarrollar detalladamente cada una de sus partes, aclarando y demostrando inclusive pasos que podrían parecer obvios o triviales. Esta última característica de la tesis no sólo sirven para lograr el objetivo de que su contenido esté al alcance de personas que tengan pocos conocimientos sobre los temas tratados, sino que nos parecen imprescindibles tomando en cuenta que el trabajo es netamente teórico y que por tanto nada puede ser afirmado si no está bien fundamentado.

CAPITULO 2

NOCIONES DE TEORÍA DE GRÁFICAS

Se dan las definiciones y los resultados básicos de Teoría de Gráficas. Estos permiten entender el concepto de árbol generador de peso mínino (AGM) de una manera formal y completa. Adicionalmente se estudian propiedades claves para desarrollar algoritmos distribuidos para construir un AGM [HAR].

2.1 Notación

Previamente daremos algunas convenciones de notación que se usan en el desarrollo del capítulo.

- Se usa el símbolo $-$ (menos) como diferencia de conjuntos.

- Sea E un conjunto de aristas de una gráfica y e una arista de tal conjunto. El conjunto $E - \{e\}$ es denotado también como $E - e$.

2.2 Definiciones

Una *gráfica* $G = (V, E)$ consiste de un conjunto finito no vacio $V = V(G)$ de n *vértices* y de un conjunto $E = E(G)$ de m parejas no ordenadas de vértices de V. Cada par $e = (u, v)$ que pertenece a E es una *arista* de G.

En este trabajo suponemos que la gráfica no tiene lazos, es decir, no hay aristas de un vértice a sí mismo, $e = (v_i, v_i)$. Lo que sí se permite, es tener aristas paralelas, es decir varias aristas entre un mismo par de vértices. Si G no tiene aristas paralelas, se dice que es una *gráfica simple.*

Sea $e = (u, v)$ una arista, entonces u y v son sus *vértices adyacentes.*

Un *camino* de una gráfica G es una secuencia alternada de vértices y aristas $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ que empieza y termina en un vértice y en la cual cada arista es incidente con el vértice que inmediatamente la precede y el que inmediatamente la sigue $e_i = (v_{i-1}, v_i)$. Se dice que este camino une a los vértices v_0, v_n y, si la gráfica es simple, también se puede denotar como $(v_0, v_1, v_2, \ldots, v_n)$.

Un *camino simple* es aquel que no repite vértices.

La gráfica $G_1 = (V_1, E_1)$ es una *subgráfica de* G si $V_1 \subseteq V$ y $E_1 \subseteq E$.

La gráfica $G = (V, E)$ es *conexa* si todo par de vértices de V están unidos por al menos un camino.

Una *subgráfica conexa maximal* de una gráfica $G = (V, E)$ es una subgráfica conexa de G que no está contenida en ninguna otra subgráfica conexa.

Una *componente conexa* de una gráfica $G = (V, E)$ es una subgráfica conexa maximal de G.

Un *ciclo* es un camino cerrado, es decir uno en el cual $v_0 = v_n$ y si todos los demás vertices intermedios $(v_1, v_2, \ldots, v_{n-1})$ son distintos, el ciclo es simple.

$G = (V, E)$ es una *gráfica acíclica* si no tiene ningun ciclo.

Un *bosque* es una gráfica acíclica $G = (V, E)$.

Un *árbol* es una gráfica conexa y sin ciclos.

Hemos repasado algunas definiciones que nos permiten uniformizar criterios y notación para, ahora sí, dar las definiciones que nos interesan:

Definición 2.2.1 Un bosque generador *de una gráfica* $G = (V, E)$ *es un bosque que contiene todos los vértices en* V *y sus aristas están contenidas en* E.

Hay que notar que las componentes conexas son árboles por lo que el bosque se puede denotar como $\bigcup_i (V_i, E_i)$ donde $V_i(E_i)$ es el conjunto de todos los vértices (aristas) del árbol i. (ver Ejemplo 1 de la figura 2.1)

Definición 2.2.2 *Un* árbol generador *de una gráfica* $G = (V, E)$ *es un árbol que contiene todos los vértices de* V. *Es decir, si un bosque generador de* G *es conexo, entonces es un árbol generador de* G *(ver Ejemplo 2).*

Definición 2.2.3 El peso de una gráfica $G = (V, E)$ *se define como la suma de los pesos de las aristas de* E *donde el* peso *asociado a una arista* e *es un número entero* $w(e)$ *(ver Ejemplo 3).*

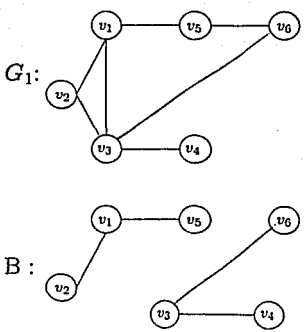


Figura 2.1: Ejemplo 1. B es un bosque generador de G_1

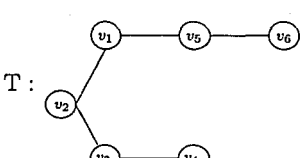


Figura 2.2: Ejemplo 2. T es un árbol generador de G_1 del anterior ejemplo

Definición 2.2.4 Un árbol generador de peso mínimo -AGM- *(Minimum spanning tree) de una gráfica $G = (V, E)$ es un árbol generador de G cuyo peso total no es mayor que el peso de ningún otro árbol generador de G (ver ejemplo 4).*

Definición 2.2.5 Un fragmento de un AGM *se define como una subgráfica conexa del AGM,*

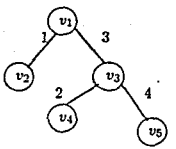


Figura 2.3: Ejemplo 3. $Peso(G_1) = 1 + 3 + 2 + 4 = 10$

es decir un subárbol del AGM (ver ejemplo 5).

Definición 2.2.6 Una arista de salida *de una subgráfica de G es una arista en la cual uno de sus vértices adyacentes está en la subgráfica y el otro no.*

Ejemplo 6: Entre los fragmentos de G_2 del ejemplo 5, (v_2, v_3) es una arista de salida de ambos fragmentos.

A continuación incluimos algunas definiciones más que serán de utilidad para el análisis del algoritmo de GHS y para uno de los algoritmos descritos en el capítulo 8.

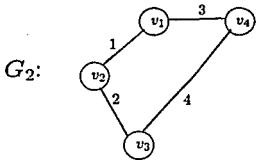
Definición 2.2.7 Un conjunto dominante de vértices *de una gráfica* $G = (V, E)$ *es un conjunto* $V_1 \subset V$ *talque* $\forall v \in V - V_1, \exists e = (v, v_1) \in E, v_1 \in V_1$. *Es decir todos los nodos fuera de* V_1 *son adyacentes a al menos un nodo de* V_1.

Definición 2.2.8 Un conjunto independiente de vértices *de una gráfica* $G = (V, E)$ *es un conjunto* $V_1 \subset V$ *tal que* $\forall v_1, v_2 \in V_1, (v_1, v_2) \notin E$. *Esto es, ningún vértice en* V_1 *es adyacente a otro vértice en dicho conjunto.*

Definición 2.2.9 Un conjunto independiente maximal de vértices *de una gráfica G=(V,E) es un conjunto independiente de vértices que no está contenido en ningún otro conjunto independiente de vértices.*

Definición 2.2.10 Un conjunto independiente de aristas *de una gráfica G=(V,E) es un conjunto* $E_1 \subset E$ *tal que* $\forall e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E_1, u_1(\ v_1) \neq u_2$ *y* $u_1(\ v_1) \neq v_2$. *Esto es, ningún par de aristas de* E_1 *tiene alguno de sus vértices adyacentes en común.*

Definición 2.2.11 Un conjunto independiente maximal de aristas *de una gráfica G=(V,E) es un conjunto independiente de aristas que no está contenido en ningún otro conjunto independiente de aristas.*



tiene los siguientes árboles generadores :

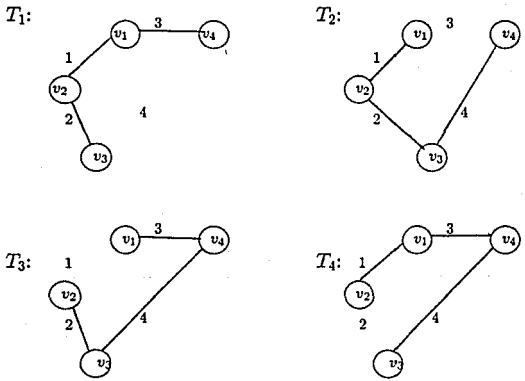


Figura 2.4: Ejemplo 4. T_1 es el AGM de G_2

2.3 Propiedades

Lema 2.3.1 *Sea $T = (V, E_1)$ un árbol generador de la gráfica $G = (V, E)$. Sea $G_1 = (V, E_1 \bigcup e)$ tal que $e \in E - E1$. Entonces G_1 contiene un ciclo.*

Demostración T es un árbol generador de G, por tanto en T existe un camino entre todo par de vértices de G. Supongamos que para construir G_1 agregamos una arista $e \in E$ ($e = (u_1, u_2)$)

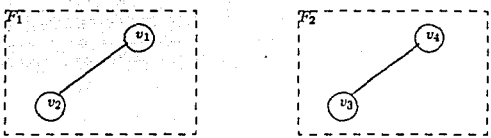


Figura 2.5: Ejemplo 5. Fragmentos de T_2 del anterior ejemplo.

a T. Entonces en G_1 habrá dos caminos que unan a u_1 y u_2: El que ya los unía en T y la nueva arista e. Estos dos caminos forman un ciclo en T_1: u_1,(camino $u_1 - u_2$ en T),e,u_1 en T_1[1] (ver figura 2.6). ■

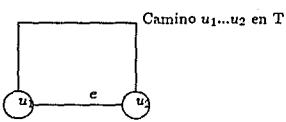


Figura 2.6: Ciclo u_1, *camino* $u_1 - u_2, e, u_1$

Lema 2.3.2 *Sea $G = (V, E)$ una gráfica conexa que contiene un único ciclo C. Sea $e = (v_1, v_2)$ una arista cualquiera de las que forman el ciclo. Entonces la gráfica $G_1 = (V, E - e)$ es un árbol generador de G.*

Demostración Para que G_1 sea un árbol generador de G debe contener todos los vértices de G, ser conexa y no tener ciclos.

Que contiene todos los vértices[2] de G viene directamente de su construcción.

Por otra parte, si el ciclo C se ve como la siguiente secuencia: $v_1 \ldots v_2, e, v_1$. Eliminando e, este ciclo quedará como un camino simple $v_1 \ldots v_2$ y estaremos eliminando el único ciclo de G. G_1 no tiene ciclos.

[1] Usamos la notación $u - v$ para simbolizar el camino del nodo u al nodo v

[2] A lo largo del trabajo usaremos indistintamente los términos 'vértice' y 'nodo'

Además los vértices de C seguirán unidos en G_1 y cualquier otro par de vértices que se unía con un camino de la forma $v_j \ldots v_1, e, v_2 \ldots v_i$ (es decir que pasaba por $v_1 - e - v_2$) tiene la alternativa de unirse por el otro camino que une v_1 a v_2 de la forma $v_j, \ldots, v_1, \ldots, v_2, \ldots, v_i$ de donde se ve que G_1 es conexa.

■

El siguiente teorema nos permite establecer la forma de elegir aristas que unan los distintos árboles o fragmentos de un bosque de manera que podamos obtener un árbol generador de peso mínimo que contenga al bosque original.

En concreto, lo que afirma el teorema es que si tomamos la arista de salida de menor peso de un fragmento cualquiera, ésta estará seguro en alguno de los árboles generadores de peso mínimo que incluya las aristas del bosque. Y esto se puede aplicar a cualquiera de los árboles del bosque.

Sin embargo, hay que notar que lo que se construye es un árbol generador de peso mínimo que incluye al bosque y no un AGM de la gráfica completa.

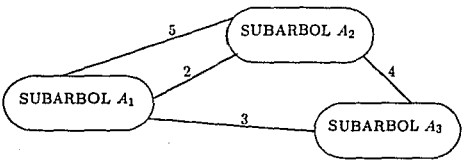


Figura 2.7: Ejemplo de una gráfica de supervértices $\dot{G}$

Partimos de un bosque generador de G. Entonces tenemos un conjunto de árboles. En cada uno de ellos se cubre un subconjunto de los vértices de G y la unión de todos ellos es V.

Podemos abstraernos de la estructura interna de cada árbol y construir una gráfica $\dot{G}$ que tenga como supervértices a estos subárboles y como aristas a las aristas que van de un árbol a otro (de salida). Es de notar que en $\dot{G}$ se podría tener aristas paralelas.

Usando $\dot{G}$ es más fácil entender por que se debe elegir las aristas de menor peso entre las que unen los subárboles:

Vamos a tener que analizar cada uno de los vértices para ver que arista consideramos para incluirla en el árbol generador (es necesario tomar al menos una de las aristas que salen de él para tener la conectividad requerida en un árbol generador).

Veamos la figura 2.7. Si tomamos el nodo A_1 y, a partir de él, vemos qué arista podemos tomar para incluir en el árbol generador, tenemos tres posibilidades: Arista de peso 3, de peso 5 o de peso 2. Si elegimos la de peso 5 o la de peso 3 y cualquiera de las posibles que unan A_2 y A_3, se ve claramente que el árbol obtenido es de mayor peso que si hubiéramos tomado la de peso 2, es decir la arista de salida de menor peso de A_1.

Teorema 2.3.3 *Sea G=(V,E) una gráfica conexa donde cada arista $e \in E$ tiene un peso w(e) asociado. Sea (V_j, E_j) con $1 \leq j \leq k$ un bosque generador de G con $k > 1$. Fijar cualquier i, $1 \leq i \leq k$. Sea e la arista de menor peso en $E - \cup_{j=1}^{k} E_j$ (es decir que todavía no está incluida en ningun árbol) tal que exactamente un vértice adyacente a e esté en V_i. Entonces, existe un árbol generador para G que incluye a $e \cup (\cup_{j=1}^{k} E_j)$ y tal que ningun otro árbol generador para G que incluya a $\cup_{j=1}^{k} E_j$ tiene menor peso.*

Demostración Sea T_1 el árbol generador de peso mínimo que incluye a $e \cup (\cup_j E_j)$.

Se hará la demostración por reducción al absurdo.

Supondremos que T es un árbol generador de G que incluye a $\cup_j E_j$ de menor costo que T_1 y que por tanto no incluye a e. Podemos agregar e a T y obtendremos un ciclo (por el lema 2.3.1). Este ciclo contiene a otra arista $e_1 \neq e$ que sale de V_i (ver figura 2.8 pues como T es un árbol debe existir al menos una arista que una (V_i, E_i) con otro subárbol (V_f, E_f), $1 \leq f \leq k$ y $f \neq i$.

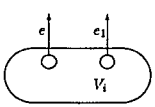


Figura 2.8: e y e_1 son aristas de salida de V_i

Por hipótesis, sabemos que $peso(e_1) \geq peso(e)$. Ahora consideramos $\hat{T}$ construido borrando e_1 y adicionando e en T. Este es un nuevo árbol generador (por lema 2.3.2) para la gráfica y cuyo peso es menor al de T y por tanto al de T_1. Contradicción! pues tenemos un árbol generador que incluye a todas las aristas del bosque original (quitamos solamente e_1 que era una arista de salida de V_i) y a e y que tiene un peso menor al de T_1, que es el menor de todos los árboles que las incluyen.

■

El siguiente teorema establece que el AGM de una gráfica cuyas aristas tienen todas distintos pesos, es único. Esto no sucede necesariamente si existen al menos dos aristas de igual peso, incluso si éstas no tienen un vértice en común.

Consideramos que resulta de utilidad dar ejemplos de lo que podría pasar en ambos casos.

Ejemplo a: En el caso del ejemplo 4 todas las aristas son distintas y se ve que todos los árboles generados tienen distinto peso y hay un único de peso mínimo (T_1).

Ejemplo b: La figura 2.9 ilustra lo que sucede si se tiene la misma gráfica que en el ejemplo a, pero con otra asignación de pesos a las aristas.

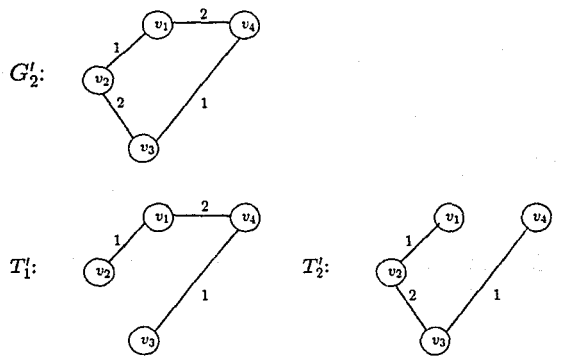


Figura 2.9: G_2' tiene dos árboles generadores de peso mínimo:T_1' y T_2'

Teorema 2.3.4 *Si todas las aristas de una gráfica conexa G tienen distinto peso, entonces el AGM de G es único.*

Demostración Supongamos que existen dos árboles $T_1 = (V, E_1)$ y $T_2 = (V, E_2)$, que son árboles generadores mínimos de G. Como no son el mismo árbol existe al menos una arista no común, o sea que está en uno y no en otro. Formamos el conjunto de estas aristas $S = \{etalquee \notin (E_1 \cap E_2)\}$ y de este conjunto elegimos la de menor peso, digamos e_1. Sin pérdida de generalidad, supongamos que $e_1 \in T_1$. Entonces $e_1 \cup T_2$ contiene un ciclo (por el lema 2.3.1) y al menos una de las otras aristas del ciclo (llamémosla e_2) no está en T_1 (Si todas estuvieran, T_1 tendría ese mismo ciclo) y por tanto está en S. Ya que los pesos de las aristas son todos

diferentes y dada la elección de e_1, necesariamente $peso(e_2) > peso(e_1)$. Pero esto implica que $(T_2 \cup e_1) - e_2$ es un árbol generador (por el lema 2.3.2) con menor peso que T_2 ! ■

CAPITULO 3

FUNDAMENTOS DE ALGORITMOS DISTRIBUIDOS

El propósito de este capítulo es el de brindar el marco teórico necesario para el desarrollo, entendimiento y análisis de los algoritmos que se presentan posteriormente.

3.1 Definiciones

Como sucede con muchos conceptos usados en distintas áreas de las Ciencias de la Computación, no existe una definición de Sistema Distribuido aceptada universalmente. Las distintas definiciones varían en función de ciertos parámetros tales como si la comunicación entre los procesadores que forman el sistema se realiza enviando mensajes o a través de memoria compartida; si dichos procesadores están sincronizados o no; o bien si el sistema es tolerante a fallas (de todo tipo o de algún tipo específico) o no.

La elección de una entre las varias definiciones existentes depende tanto del sistema real que se quiera modelar como de los aspectos relevantes del mismo que se desee analizar. El modelo que escogemos aquí es suficientemente sencillo y general para el estudio que nos interesa llevar a cabo.

En términos generales, asumiremos que un *sistema distribuido* es un conjunto de procesadores con memoria propia que pueden comunicarse enviando y recibiendo mensajes a través de líneas de comunicación provistas por una red [SEG],[AWE].

A su vez, definimos un *Algoritmo Distribuido* como un conjunto de algoritmos secuenciales que contienen entre sus instrucciones recepción y envío de mensajes y tal que cada uno de estos algoritmos corre solo en un procesador.

Cabe recalcar que los algoritmos distribuídos pueden ser *balanceados* o *no balanceados*. A los segundos tambien se los conoce con el nombre de *algoritmos centralizados*, ya que se

caracterizan porque su ejecución se concentra en uno de los procesadores del sistema. Eventualmente sus instrucciones incluyen envío y recepción de mensajes para recolectar información pertinente de otros procesadores.

En general es deseable que todos los procesadores trabajen aproximadamente en la misma medida y que no se concentre en uno toda la tarea. De otra forma, el procesador que realiza toda (o casi toda) la actividad se convierte en un cuello de botella ya que es muy probable que el resto de los procesadores se mantengan inactivos la mayor parte del tiempo esperando respuestas (u 'ordenes') del único que trabaja. Adicionalmente, un algoritmo que trabaje bajo un esquema centralizado tiende a ser muy sensible a fallas: una falla en el procesador que hace la mayoría del trabajo es prácticamente imposible de recuperar.

3.2 Modelo

Definimos el esquema de Sistema Distribuído sobre el que vamos a trabajar como *una gráfica* $G = (V, E)$ *donde cada vértice* $v \in V$ *es asociado a un procesador y el conjunto de aristas* E *representa las líneas de interconexión entre parejas de procesadores en una red. Los procesadores se comunican enviando mensajes a lo largo de las aristas.* Adicionalmente se les asocia pesos a las aristas que representan el costo de enviar un mensaje por cada una de ellas -en función del tráfico sobre la línea, la velocidad de transmisión, etc.

Con base en este esquema general se construyen distintos modelos más específicos según las características particulares y las restricciones que interese aplicar sobre las líneas de transmisión, los procesadores y el tipo de comunicación.

Nosotros elegimos uno que es comúnmente usado en distintos algoritmos distribuídos [GHS].

i. La red de interconexión G es una gráfica cualquiera (topología cualquiera).

ii. Las redes son estáticas pues no están sujetas a cambios, es decir no se agregan ni eliminan procesadores y/o líneas de transmisión.

    En un sistema real, es difícil asegurar que ningún procesador y ninguna comunicación vaya a fallar o que no se agreguen nuevos componentes al sistema. Para el modelo, la red resultante de cualquiera de esas posibles modificaciones será tratada como una nueva red.

iii. Los mensajes llegan siempre a su destino aunque el tiempo transcurrido entre que se envía y se recibe un mensaje no está acotado y no se conoce de antemano.

    Normalmente no se puede suponer que un sistema real cumpla con este requerimiento. En caso de que no lo haga, se pueden introducir números de mensajes y protocolos tales que el envío de un mensaje requiera un reconocimiento (acknowledge) de parte del receptor para detectar, y en lo posible recuperar, la pérdida de mensajes.

iv. Cada canal satisface la propiedad de que los mensajes llegan en el orden en que fueron enviados (forma FIFO, del inglés First In - First Out), es decir los mensajes que viajan sobre un mismo canal o línea de transmisión no pueden saltarse unos a otros.

v. Los procesadores tienen un nombre o *identificador*. Estos identificadores son distintos entre sí. Tal suposición no solo se apega a lo que sucede con frecuencia en la práctica, sino que es fundamental ya que se sabe que si una gráfica no tiene ni los identificadores de los vértices ni los pesos de las aristas diferentes, entonces no existe ningún algoritmo distribuido que efectúe el cómputo de un AGM enviando un número acotado de mensajes [1].

   Para tener n identificadores distintos, la longitud de los mismos debe ser de al menos $\log(n)$ bits.

vi. Las aristas tienen distintos pesos y cada uno de ellos es representado con O(log(n)) bits.

   En la práctica los pesos de las aristas no son necesariamente distintos pero es posible simularlo, por ejemplo, concatenando los números de los identificadores de los nodos extremos de una arista al final de su peso [peso(u,v) número(u) número(v)]. Cabe notar que esta modificación no varía el orden existente entre los pesos originales y solo establece un orden arbitrario entre las aristas con igual peso.

   La importancia de tener aristas con pesos distintos viene del hecho de que esa característica asegura la unicidad del AGM (Ver teorema 2.3.4).

vii. Cada procesador conoce solamente sus aristas incidentes y los pesos de las mismas. Ningún nodo sabe cuál es la topología completa de la red. La información no local debe aprenderse enviando mensajes.

3.3 Análisis de Complejidad

Al diseñar algoritmos distribuidos, nos interesa analizar su desempeño en términos tanto del número de mensajes enviados en su ejecución como del tiempo que lleva la misma [JOH],[AHO]. En el presente trabajo no pondremos mayor énfasis en los recursos de memoria necesarios ya que los algoritmos descritos tienen bajos requerimientos en este sentido.

En cualquier caso, se desea calcular la complejidad de un algoritmo en función de uno o varios parámetros del tamaño de la entrada significativos para el análisis particular. En el caso de los algoritmos que trabajan sobre una red se toman como parámetros del tamaño el número de procesadores ($|V|$) y el número de líneas de transmisión ($|E|$) que la forman.

[1] Esto se puede ver fácilmente en una gráfica completa con tres vértices y donde los pesos de todas las aristas son iguales. Cualquier par de aristas forma un AGM pero ya que los vértices son idénticos, siguen el mismo algoritmo y no tienen manera de elegir. Si los nodos eligen aleatoriamente identificadores, entonces el algoritmo trabajaría correctamente una vez que los tres identificadores fueses distintos pero no hay manera de garantizar que esto se logre en un número finito de elecciones

Notación Asintótica

El análisis de la complejidad de un algoritmo -secuencial o distribuído- se realiza usualmente mediante la notación matemática asintótica . Usando dicha notación determinamos la complejidad -de alguno de los recursos mencionados- como una función del tamaño de la entrada sin tomar en cuenta constantes multiplicativas y/o aditivas ni los términos menos significativos y de manera asintótica, es decir ignorando valores pequeños del tamaño. Así proveemos una forma general de la función que es muy útil -y normalmente suficiente- sobre todo en casos donde determinar la complejidad exacta es un problema particularmente difícil.

La notación asintótica puede tener una de las siguientes tres formas, donde n es un entero que representa el tamaño de la entrada

1. *(O)-Notación 'orden de'.* Nos da una cota superior de la función que se está evaluando.

   Formalmente:

   Sean f y g funciones sobre los enteros positivos :

   $$f(n) = O(g(n)),$$

   es decir f(n) es de orden g(n) si existen constantes $C > 0$, $n_0 \in \mathcal{N}$ tal que,

   $$|f(n)| \leq C|g(n)|$$

   $\forall n > n_0$

2. *(Ω)-Notación 'Omega'.* Brinda una cota inferior de la función que se está evaluando.

   Formalmente:

   Sean f y g funciones sobre los enteros positivos :

   $$f(n) = \Omega(g(n)),$$

   es decir f(n) es omega de g(n), si existen constantes $C > 0$ y $n_0 \in \mathcal{N}$ tal que,

   $$|f(n)| \geq C|g(n)|$$

   $\forall n > n_0$

3. *(Θ)-Notación 'Theta'.* Para evaluar la función de una manera más precisa.

   Formalmente:

   Sean f y g funciones sobre enteros positivos :

   $$|f(n)| = \Theta(|g(n)|),$$

   es decir f(n) es theta de g(n), si existen constantes positivas C1 y C2, n_0 tales que,

   $$C1|g(n)| \leq |f(n)| \leq C2|g(n)|$$

   $\forall n > n_0$

   o bien

   $$f(n) = \Omega(g(n)) \text{ y } f(n) = O(g(n))$$

Notemos que si bien la notación anterior está definida para un solo parámetro (n), se puede generalizar fácilmente a más de uno. En el caso de algoritmos sobre redes, por ejemplo, $n = |V| + |E|$ donde $|V|$ es el número de nodos en la red y $|E|$ es el número de líneas de comunicación en la misma.

Medidas de Complejidad

La notación matemática sobre la cual trató el anterior punto es obviamente independiente del análisis de algoritmos aunque se use ampliamente en el mismo.

Al contrario, los siguientes tres criterios sí son específicos para evaluar los recursos necesarios al ejecutar un algoritmo, es decir para evaluar su complejidad.

1. *Peor Caso.* Evaluamos la complejidad seleccionando la estructura o los valores de los datos de entrada que dan la peor ejecución del algoritmo para cada tamaño de entrada n.

2. *Mejor Caso.* En este caso seleccionamos la estructura de los datos de entrada que dé la mejor ejecución posible para cada tamaño de entrada n.

3. *Caso Promedio.* Se evalua el funcionamiento promedio del algoritmo sobre todas las entradas de cada tamaño n.

La elección de uno de los tres enfoques dependerá de la aplicación particular y del grado de dificultad que signifique el cálculo de uno u otro.

A lo largo de este trabajo se analiza la complejidad, en número de mensajes y tiempo de ejecución, en el peor caso.

• La *Complejidad de mensajes* bajo el criterio del peor caso es igual al máximo número posible de mensajes enviados durante una ejecución del algoritmo sobre una red de tamaño $|E|$ y $|V|$.

Consideramos, como es usual, la longitud o tamaño máximo de un mensaje como de orden O(log(n)) donde n es el número de procesadores en la red. Esto quiere decir que si se desea enviar un bloque de información de mayor tamaño, éste tendrá que dividirse en dos o más mensajes. La razón de esta última restricción es no permitir mensajes arbitrariamente grandes que en una red real, por lo general, no podrían ser enviados en un bloque . Además se desea que no sea posible codificar en un solo mensaje la topología de toda la red. Mediante esta restricción solo se pueden codificar un número constante de identificadores puesto que, como ya se estableció, en una red de n procesadores, la longitud de los identificadores deben tener una longitud de al menos log(n).

• La *Complejidad de tiempo* de un algoritmo distribuido es igual al máximo número posible de unidades de tiempo que toma el algoritmo en cualquier ejecución sobre G=(V,E) desde que comienza hasta que termina suponiendo que el tiempo de envío de un mensaje sobre una línea

de transmisión es a lo más una unidad [2] y el tiempo de procesamiento interno de un procesador es despreciable. Esto debido a que, normalmente, se considera al tiempo que lleva la ejecución de una instrucción en un procesador mucho menor al que consume el envío de un mensaje.

Notemos que ni en la evaluación del número de mensajes ni en la del tiempo se han tomado en cuenta factores particulares de cada red, como el tamaño específico de mensaje, la velocidad de transmisión, el retraso de mensajes debido a congestionamiento, etc. En particular asumimos por simplicidad que el peso asociado a cada arista es 1 para los análisis de complejidad.

[2] Esta suposición simplifica el análisis y es frecuente al analizar algoritmos distribuidos. Estamos asumiendo que el mensaje más lento necesita una unidad de tiempo para llegar a su destino y los demás tardan menos. De otra manera sería imposible hacer una generalización sobre cuánto es el máximo tiempo que tarda un mensaje para viajar de un punto a otro.

CAPITULO 4

EL PROBLEMA Y SUS APLICACIONES

4.1 El problema

Hallar el AGM de una gráfica es un problema ampliamente discutido en el área de Teoría de Gráficas. Existen muchas propuestas para resolverlo, inicialmente usando algoritmos secuenciales. Los trabajos de DKIJSTRA-PRIM y de KRUSKAL que respectivamente presentan una solución centralizada y una solución compuesta según algún criterio por soluciones parciales sobre subgráficas [1], son determinantes y de hecho el resto de los algoritmos, incluso los distribuidos, siguen en gran medida una de las dos ideas o una mezcla de ellas.

Como ya se mencionó brevemente, encontrar el AGM ha suscitado especial interés en el área de Sistemas Distribuidos debido a que permite importantes reducciones en las complejidades de comunicación, sobre todo cuando se trata de protocolos que involucran la difusión de información a todos los nodos de la red.

Notemos que la característica que le permite a un árbol generador ser una estructura por la cual viajen los mensajes que se desee difundir, es que es una subgráfica conexa y por tanto se puede lograr comunicación entre cualquier par de procesadores usando solamente las aristas del árbol generador. Adicionalmente tiene la ventaja de ser la gráfica con esta propiedad que tiene menor número de aristas y más aún si es un árbol generador mínimo con el menor peso (costo) entre todos los árboles generadores posibles.

4.2 Algunas aplicaciones específicas

Existen varios problemas en el contexto de Sistemas Distribuidos cuya solución puede mejorarse contando con el árbol generador mínimo.

[1] Para mayor detalle consultar las referencias [HAR] [EVE]

Poner en evidencia la forma en que el AGM permite reducir las complejidades de cada uno de estos problemas sería una tarea demasiado larga y nos desviaría del propósito central del presente trabajo. En lugar de ello, decidimos dar un ejemplo que, por un lado, sea significativo o sea que resuelva un problema de especial interés dentro del área de algoritmos distribuidos y, por otro lado, sea suficientemente claro para poder inferir de él la forma en que se resuelven los demás problemas.

Eligimos el problema de hallar un líder, cuya relación con el del AGM ha sido ampliamente citada por distintos autores ([AWE], [KOR], [AFE]) debido a la importancia y utilidad de las soluciones de ambos problemas.

Hallar un líder consiste en la elección de un nodo que se distinga de todos los restantes que forman la red. Contar con tal nodo es sumamente útil ya que se le puede hacer responsable de una serie de tareas tales como sincronización de relojes, inicialización del sistema, etc.

Daremos la descripción general de un algoritmo que elige un líder en una red usando el AGM para el envío de los mensajes [2]. Su estructura es sencilla de manera que las ventajas de contar con el AGM puedan notarse fácilmente. Además incluimos un análisis formal de las complejidades de tiempo y mensajes.

4.2.1 Algoritmo de elección de Líder

Construimos un algoritmo que elige un líder suponiendo que se tiene ya un árbol generador mínimo.

Vamos a trabajar sobre el modelo general propuesto.

1. Cada nodo observa sus aristas incidentes en el árbol para ver si es hoja (es decir si tiene una sola arista incidente del árbol) o no.

2. Un nodo hoja envía el mensaje BUSCAR-LIDER al nodo al otro extremo de su única arista incidente y se pone en estado **TERM**.

   Un nodo interno con m aristas incidentes en el AGM, espera mensajes BUSCAR-LIDER de $m-1$ de sus vecinos (estado **ESPERA**). Cuando detecta que ya recibió los $m-1$ BUSCAR-LIDER, reenvía el mensaje al vecino restante y pasa a estado **TERM**.

3. Si un nodo **u** en estado **TERM** recibe un nuevo mensaje, necesariamente lo recibe por la arista **(u,v)** que él mismo usó para transmitir. Es decir los mensajes se cruzaron.

   En este caso manda su identificador -**ID(u)**- a v (otro extremo de la arista) y pasa a estado **CANDIDATO** y espera un mensaje de v con su identificador -**ID(v)**-.

[2] "Es fácil elegir un líder en una gráfica si ésta es un árbol (rompiendo la simetría causada por ciclos es la tarea más difícil de un algoritmo general de elección de líder). Entonces una estrategia razonable para la elección de líder puede ser encontrar inicialmente un árbol generador....." [LYN]

4. Cuando u, un nodo en estado **CANDIDATO** recibe el mensaje ID(v), compara el identificador de v con el suyo. Si el suyo es el mayor entonces se proclama líder.

5. Una vez que un nodo se ha proclamado líder, difunde su identidad como líder al resto de los nodos utilizando las aristas del AGM.

4.2.2 Correctez y Complejidades

Lema 4.2.1 *Sea G=(V,E) un árbol. Se cumple que todo nodo $v \in V$ envía uno y solo un mensaje BUSCAR-LIDER en cualquier ejecución del algoritmo sobre G.*

Demostración Haremos la demostración por reducción al absurdo y tomando en cuenta que un nodo no puede enviar BUSCAR-LÍDER si está esperando mensajes de nodos adyacentes (o sea está en estado ESPERA).

Supongamos que existe al menos una ejecución Λ del algoritmo tal que al terminar el conjunto de nodos $V_1 \subset V$, $V_1 = \{$*Los nodos que no enviaron mensajes durante* $\Lambda\}$ es distinto de vacío.

Tomemos cualquier $v \in V_1$ y veamos de cuales de sus nodos adyacentes no recibió el mensaje. Se pueden dar dos casos

i) Si es de uno solo, entonces recibió el mensaje de todas sus aristas incidentes menos una y por tanto ya no puede estar en estado **ESPERA**.

ii) Si hay más de un nodo adyacente a v que está esperando el mensaje, entonces se pasa a analizar alguno de ellos y así se sigue la secuencia de nodos en estado ESPERA hasta encontrar el primero . No puede ser una hoja porque una hoja no espera por nadie, entonces, al ser el primero en estado **ESPERA** todos sus nodos adyacentes, menos el que está esperando por él (el siguiente en la secuencia), ya le enviaron el mensaje entonces no puede estar esperando. Notemos que en este caso usamos el hecho de que G es un árbol: G no contiene ciclos y no existe un par de nodos esperando el uno por el otro. Los nodos en estado ESPERA forman necesariamente una cadena y por eso podemos hablar del *primer nodo en estado ESPERA*. Se concluye que no existe un nodo que se mantenga en estado **ESPERA** en ninguna ejecución del algoritmo.

Además cada nodo envía un único BUSCAR-LIDER porque inmediatamente que lo envía se pone en estado **TERM** y ya no participa en el algoritmo a no ser si eventualmente le toca enviar su identificador. ∎

Lema 4.2.2 *Sea $G = (V, E)$ un árbol. Sea $E_1 \subseteq E$,*

$E_1 = \{$aristas por las que se manda $BUSCAR - LIDER$ en ambas direcciones$\}$.

Entonces, en cualquier ejecución sobre G, $|E_1| = 1$.

Demostración i) $|E_1| > 0$: El lema 4.2.1 implica que se envían n mensajes BUSCAR-LÍDER y puesto que $m = n - 1$ (G es un árbol) concluímos que existe al menos una arista usada dos veces para mandar el mensaje por ella, uno en cada dirección pues cada nodo envía un solo mensaje.

ii) $|E_1| = 1$ exactamente: Por reducción al absurdo, supongamos que $|E_1| = q > 1$. Por el lema 4.2.1 sabemos que se envían n mensajes. De ellos $n - 2q$ se envían por aristas distintas. De donde:

No. aristas por las que se envía al menos un mensaje=
No. aristas por las que se envía un mensaje +
No. aristas por las que se envía al menos dos mensajes=
$(n - 2q) + q < n - 1 = m$ pues supusimos que $q > 1$

Por tanto hay aristas que no se usan para mandar BUSCAR-LÍDER en ninguna dirección. Sea (u_1, u_2) una de ellas, ni u_1, ni u_2 han recibido ni enviado mensajes por ella. Entonces al menos uno de los dos está esperando recibir un mensaje por ella y por el lema 4.2.1 sabemos que ningún nodo se queda en estado ESPERA. Contradicción ! ∎

Teorema 4.2.3 *El algoritmo AGM de Líder suponiendo que se tiene el AGM es correcto.*

Demostración Debemos demostrar que el algoritmo termina habiendo hallado un único líder para la gráfica.

Para ver que el algoritmo termina podemos notar que todos los nodos trabajan un tiempo finito en él. Una vez que un nodo envía un mensaje, se mantiene inactivo el resto del algoritmo (ver lema 4.2.1).

Por el lema 4.2.2 existe una única arista por la cual se envían mensajes en ambas direcciones. Sea e=(u,v) tal arista, entonces u y v son los únicos nodos que reciben el mensaje por todas sus aristas incidentes y por tanto pasan a estado CANDIDATO. En ese estado ambos intercambian mensajes para informar su identidad al otro y solo el que tenga el mayor identificador se proclama líder. ∎

Teorema 4.2.4 *La complejidad del algoritmo en número de mensajes es O(n)*

Demostración - El lema 4.2.1 implica que se mandan O(n) mensajes BUSCAR-LIDER.

- El mensaje de difusión de la identidad del nodo elegido como líder viaja también una sola vez por todas las aristas del árbol que son n-1.

Por tanto, *Complejidad de mensajes del algoritmo es O(n)* ∎

Teorema 4.2.5 *La complejidad del algoritmo en tiempo es O(n)*

Demostración El tiempo que tarda el mensaje BUSCAR-LÍDER en llegar a la raiz equivale al tiempo que tardan en enviarlo todos los nodos en el camino más largo de un nodo a la raiz y esto es O(n).

Similarmente el tiempo que consume avisar la identidad del líder a todos los nodos es a lo sumo n

Por tanto, *Complejidad de tiempo del algoritmo O(n)* ∎

4.2.3 La comparación

Hemos obtenido un algoritmo lineal sobre el número de nodos de la red tanto en el tiempo como en el número de mensajes. Comparando estos resultados con el $\Omega(n \log n)$ ([AFE]) hallada para la complejidad de mensajes de los algoritmos asíncronos para elegir un líder sobre una gráfica general se ve que la mejoría es notoria.

De hecho, en términos del número de mensajes es óptimo pues es un problema esencialmente global, es decir su solución requiere la participación de todos los nodos de la red y por tanto cada uno debe recibir al menos un mensaje a lo largo del algoritmo.

Para analizar lo que sucede con el tiempo, notemos que el algoritmo en realidad hace uso de la propiedad de contar con un árbol generador no necesariamente de peso mínimo. Es decir, funciona igual con cualquier árbol generador y en ese sentido podría parecer mejor utilizar un árbol generador cuya altura sea de O(diámetro(G)) para que el tiempo sea también O(diámetro(G)) y ya no el AGM cuya altura es O(n). De todas maneras un tiempo O(n)es suficientemente bueno [3] y más si consideramos que la información se difunde con el menor costo posible. Aunque esta última ventaja no se refleje en el algoritmo es obvia su importancia en redes donde el costo asociado a cada línea de comunicación es una característica muy significativa al momento de decidir la conveniencia de un algoritmo u otro para solucionar un problema, más uno tan frecuente como el de elegir un líder.

Notemos que el problema recíproco, es decir hallar el AGM habiendo encontrado previamente un líder no es simple. El problema que si se simplifica es el de hallar un árbol generador cualquiera; sin embargo, no se conoce hasta el momento un algoritmo para resolver el problema del AGM que se beneficie sustancialmente de contar con un líder. Intuitivamente se pensaría en intentar que el líder compute o al menos 'comande' el algoritmo que encuentra el AGM pero, como se verá en el siguiente capítulo, estas dos aproximaciones no son eficientes.

4.2.4 La Cota Inferior

La estrecha relación entre el problema del líder y el del AGM de una gráfica tiene un interés adicional. La cota inferior de $\Omega(n \log n)$ que mencionamos en la anterior subsección sirve

[3] Muchos autores aceptan tiempo O(n) como óptimo en una red de n nodos considerando que en el peor caso el diámetro de una red es n

también para establecer la cota inferior en el caso de algoritmos asíncronos para hallar el AGM. Intuitivamente, ésta es $\Omega(e + n \log n)$, el término e viene de la necesidad de enviar al menos un mensaje por cada arista para examinarla y, $n \log n$ viene de la cota inferior de la elección de líder. De lo contrario, es decir si existiera un algoritmo que encontrara el AGM con menor complejidad de comunicación, se podría encontrar un líder aplicando primero tal algoritmo y luego un algoritmo que, como el presentado en este capítulo, halle el líder con complejidad $O(n)$.

Además del interés teórico, una de las ventajas obvias de contar con una cota inferior demostrada formalmente es que sirve para evaluar la eficiencia de cualquier algoritmo para el problema del AGM que se pudiese construir.

CAPITULO 5

DOS ALGORITMOS CENTRALIZADOS PARA HALLAR EL AGM

Se desarrollan dos algoritmos centralizados donde los procesadores de una red trabajan para hallar el árbol generador mínimo de la misma.

Además de las características presentadas en el modelo general descrito en el capítulo 3, vamos a suponer la existencia de un nodo distinguido (si inicialmente no se cuenta con uno, se deberá aplicar previamente un algoritmo de elección de líder) que es el que recibe la señal para empezar a procesar el algoritmo.

5.1 Primera versión

La primera idea que surge, sabiendo que se tiene un nodo distinguido, es cargarle todo el trabajo. Es decir, que él sea el encargado de computar el AGM en su totalidad mientras que la participación de los otros procesadores se limite a enviarle la información que requiere para ejecutar el algoritmo (por ejemplo, los pesos de todas las aristas) antes del inicio del mismo. Esto último debido a que, como se estableció, ningún nodo conoce la topología de la red en su totalidad.

5.1.1 Descripción general del algoritmo

El algoritmo consiste de cuatro pasos principales:

i. El nodo distinguido INIC pide al resto de los nodos que le envíen sus aristas adyacentes y los pesos de las mismas.

ii. Cada nodo envía su información a **INIC**.

iii. A partir de la información recolectada, **INIC** computa el AGM usando el algoritmo de PRIM [1].

iv. **INIC** difunde el árbol encontrado a los demás nodos.

5.1.2 El algoritmo

1. Cuando el nodo distinguido -llamémoslo nodo **INIC**- es despertado realiza un broadcast pidiendo el identificador y las aristas incidentes a cada nodo usando la idea del algoritmo PIF (Propagation of Information with Feedback) [2]; cuando un nodo recibe por primera vez un mensaje que contiene este requerimiento, denota la arista por la cual lo recibió con una marca especial **PRE** y reenvía el mensaje por todas sus aristas incidentes menos por la marcada. Por ella, manda un aviso al nodo del otro extremo avisándole que es su **PADRE**.

   Al utilizar el algoritmo PIF, aseguramos que, al terminar este proceso, las aristas marcadas con **PRE** forman un árbol generador con raíz **INIC**.

2. Si un nodo recibió ya el mensaje de requerimiento de todos sus vecinos, se prepara para responder al nodo **INIC**. En este momento, ya sabe que todos sus vecinos están en posibilidades de responder [3] y espera las respuestas de todos los nodos adyacentes que lo marcaron como **PADRE** (si existe al menos uno). Luego las reenvía a través de la línea de transmisión que marcó con **PRE** en el primer paso y envía tambien su propia respuesta.

   Es decir, cada nodo que no es hoja en el árbol generado es responsable de reenviar las respuestas de sus hijos, agregando la suya propia, exclusivamente hacia el nodo que es su padre. Las hojas simplemente mandan su información a sus respectivos padres.

   Para asegurar que no lleguen datos repetidos, se adopta la convención de que solo uno de los dos extremos de una arista, el de mayor identificador, la envíe.

   El nodo **INIC** va guardando la información que recibe en una lista de nodos y una lista de aristas.

3. Una vez que el nodo **INIC** haya recibido las respuestas de todos sus vecinos, ya cuenta con toda la información de la red en las listas respectivas y puede computar él solo el AGM.

   Para ello usa una versión sencilla del algoritmo (secuencial) de Prim. Este algoritmo mantiene un conjunto de nodos ya incluidos en el AGM (conjunto S -inicialmente vacío-) y uno de nodos no incluidos (conjunto N -al principio todos los nodos están en él-) .

[1] Para mayor detalle consultar las referencias [HAR], [EVE]

[2] Para mayor detalle consultar referencia [SEG].

[3] Si le enviaron el mensaje de requerimiento, es porque ellos ya lo recibieron al menos una vez

Además maneja un arreglo donde almacena, para cada vértice v que aun esté en N, el peso de la arista de menor peso que va desde algún vértice de S a v y otro arreglo donde se guarda, también para cada vértice, su padre en el árbol (inicialmente está en nil).

A grandes rasgos, lo que hace el algoritmo es tomar el conjunto de nodos N y mientras éste no esté vacío, saca el nodo u cuya arista a algún nodo de S -digamos v- sea la de menor peso entre todas las que van de un nodo de S a un nodo de N y lo agrega a S. Pone a v como padre de u y actualiza la arista de menor peso de todos los nodos adyacentes a u (pues ahora él también está en S).

4. Cuando termina la construcción del árbol generador mínimo de la red, el nodo **INIC** manda a todos los nodos la estructura del AGM a través de las aristas del árbol.

5.1.3 Estructuras de datos

- Cada nodo tiene un arreglo de sus aristas incidentes junto a sus pesos y un campo donde, en caso necesario, se la marque como **PRE**.

- El nodo **INIC** requiere una lista S y una N donde se mantienen los nodos incluidos y no incluidos respectivamente y una lista de **ARISTAS** que contiene todas las aristas de la red con su peso asociado; un arreglo **AMP** que contiene la arista de menor peso de S a cada nodo de N y una lista de padres donde, al fin de cuentas, se va almacenando el AGM.

5.1.4 Mensajes

Los mensajes enviados para ejecutar el algoritmo son de los siguientes tipos:

- **REQUERIM:** Pedido del nodo INIC de la información de cada nodo.

- **PADRE:** El mensaje que manda cada nodo al nodo del cual recibió el mensaje REQUERIM por primera vez.

- **RESPU:** El identificador y las aristas incidentes que envía cada nodo al INIC.

- **COPIA.AGM:** Estructura del AGM enviada por INIC a todos los nodos una vez terminado su cómputo.

En el análisis de complejidades n será el número de procesadores y m el número de líneas de transmisión.

5.1.5 Complejidad de mensajes

Broadcast inicial:

El mismo mensaje es enviado a través de todas las líneas de comunicación.

Número de mensajes REQUERIM enviados $= O(m)$

Y para construir el árbol por donde viajarán las respuestas se necesitan n-1 mensajes.

Número de mensajes PADRE enviados $= O(n)$

Respuesta con identificador y aristas incidentes:

Cada procesador envía su respuesta hasta el nodo INIC. Este mensaje debe ser replicado por todos los procesadores en el camino entre el nodo y el nodo INIC sobre el árbol. En el peor caso o sea si el árbol es una cadena que comienza en INIC, la longitud del camino del nodo INIC al nodo más cercano será de 1, al siguiente será de 2 y así sucesivamente hasta que al más lejano será de $n-1$. Entonces, el total de mensajes RESPU será $1+2+...+n-1 = (n-1)n/2$.

Sin embargo, la longitud de los mensajes RESPU es de $2n\log(n)$ (El $2n$ viene de que, en el peor caso contiene n aristas y, por lo tanto, $2n$ identificadores. El término $\log(n)$, a su vez, se debe a que, como se dijo , la longitud de los mismos debe ser al menos igual a $\log(n)$). Es decir, tienen una longitud mucho mayor a la que mencionamos para considerar un mensaje como unidad. Para un análisis correcto de complejidad, cada uno de estos bloques de información se divide en $2n$ mensajes. Por tanto,

Número de mensajes RESPU enviados $= 2n[(n-1)n/2] = O(n^3)$

Aviso:

El mensaje es enviado exclusivamente por las aristas del árbol que son $n-1$. Sin embargo, cada mensaje LISTAAGM es muy largo pues contiene $n-1$ aristas o sea $2(n-1)$ identificadores. Por las mismas razones expuestas al analizar los mensajes RESPU, estos mensajes deberán dividirse en $2(n-1)$ pedazos por lo que se tiene:

Número de mensajes LISTAAGM $= O(n^2)$

Por tanto:

La complejidad en Número de mensajes es $O(m+n^3+n^2) = O(n^3)$

5.1.6 Complejidad de tiempo

Broadcast inicial:

El tiempo que tarda el mensaje REQUERIM desde INIC al nodo más lejano en el peor caso es n. Los otros mensajes tardarán menos. En este tiempo se incluye al mensaje PADRE pues su envío se realiza simultáneamente a la propagación del mensaje REQUERIM. Luego,

Tiempo del broadcast y formación del árbol $= O(n)$

Respuesta:

En el peor caso, el mensaje más lejano viaja a través de n nodos, pero como ya se expuso cada nodo envía su información completa en n mensajes . Además cada nodo interno debe

esperar los mensajes de sus hijos para reenviarlos y enviar los suyos propios secuencialmente a través de línea de transmisión que marcó como PRE, por lo que,

Tiempo respuesta $= O(n^3)$

Tiempo del algoritmo ejecutado en INIC:

El tiempo, en este caso está medido de acuerdo a lo que tarda cada instrucción en ejecutarse y es determinado por el algoritmo de Prim.

Tiempo del algoritmo $= O(n^2)$

Aviso:

El tiempo es $O(n^2)$. En el peor caso los $2(n-1)$ mensajes deben pasar por n nodos.

Por tanto, tenemos que en total:

Complejidad en tiempo del algoritmo $= O(n^3)$

5.1.7 Análisis del algoritmo

Observando las complejidades se puede ver que el algoritmo dista mucho de ser óptimo, ni siquiera bueno tanto en términos de tiempo como de número de mensajes. Esto se puede explicar si hacemos dos consideraciones:

i. En gran medida los mensajes tienen como origen o destino el nodo INIC. Esto implica que se mueven prácticamente a lo largo de toda la red (con los consecuentes reenvíos) y no por fragmentos de la misma.

ii. El gran tamaño de los bloques de información que se envían obliga a particionarlos en mensajes cuya longitud sea la aceptada en el modelo propuesto.

Estas dos características del algoritmo son consecuencias directas de su forma de proceso centralizado. Es poco factible -y tampoco sería eficiente- que cada nodo de una red conozca toda la información acerca de la misma. Normalmente los nodos cuentan con datos locales y para acceder a la información remota, deben comunicarse unos con otros mediante el envío de mensajes. Mientras más información requiera concentrar un nodo, el número de los mensajes y su tamaño crecerán más rápidamente.

Este tipo de procedimiento hace prácticamente infructuoso cualquier intento de rebajar las complejidades del algoritmo:

El uso del algoritmo de PIF no solo nos sirve para que el nodo INIC esté seguro que al recibir las respuestas de sus hijos, está obteniendo la información completa de toda la red (es decir ningún nodo se queda sin enviar su información), sino también para evitar que las respuestas, que como ya se vió son extensas, viajen a través de todas las aristas de la red y lo hagan exclusivamente por las aristas del árbol. Aún así la complejidad no baja de $O(n^3)$.

Se podría también intentar, por ejemplo, reducir el número de mensajes debidos al envío de la estructura completa del AGM a todos los nodos y enviar cada arista del árbol solamente a sus dos nodos incidentes. Sin embargo esta modificación no reduciría la complejidad total en número de mensajes ni en tiempo.

Lo que no se puede evitar, así como está planteado el algoritmo , es que todos los nodos envíen la información de toda la red al nodo INIC con las consecuencias ya mencionadas.

5.2 Segunda Versión

Por lo visto en el anterior punto, nos interesa construir una nueva versión, mejor que la primera.

No se modifica la suposición de existencia de un líder (nodo distinguido), sin embargo se intenta repartir el trabajo en forma más equitativa entre los distintos procesadores haciendo que participen más activamente.

5.2.1 Descripción general del algoritmo

Se mantiene la idea básica del anterior algoritmo (basado en el algoritmo de Prim). Es decir, se construye el AGM empezando por el líder (es la raiz del árbol) y se va agregando los demás nodos uno a uno eligiendo en cada fase el nodo al que llega la arista de menor peso incidente a alguno de los nodos ya incluidos en el AGM.

Al término del algoritmo, cada nodo conoce sus aristas incidentes que están en el árbol y el nodo distinguido envía simplemente una señal avisando que ya finalizó.

5.2.2 El algoritmo

El algoritmo consiste de los siguientes pasos:

1. Inicialmente, cuando el líder -INIC- recibe una señal del exterior para que empiece a ejecutar el algoritmo, se incluye en el AGM [4].

2. En cada fase, a través de las aristas del AGM ya construído el nodo INIC envía un mensaje a todos los nodos ya incluidos en él pidiendo que le avisen cual es su arista incidente de menor peso -APM- que llegue a un nodo no incluído en el AGM. En la primera etapa solo INIC está en el AGM así que el mismo elige la de menor peso entre sus aristas incidentes.

[4] Cuando se hable de inclusión de un nodo en el AGM, en términos estrictos nos estaremos refiriendo a su incorporación al subárbol del AGM que se haya construido hasta ese momento.

3. Al recibir el requerimiento del nodo INIC, cada nodo del AGM manda un mensaje por sus aristas incidentes que no estén en el árbol o a las que aún no haya excluido del análisis (Una arista de la que se sabe que no está en el árbol pero cuyos dos nodos extremos ya estén incluidos en él ya no se toma en cuenta) [5]; y espera la respuesta de los nodos al otro lado de tales aristas. Cada uno de ellos debe avisarle si está o no ya incluído en el AGM (a través de otra arista incidente a él).

   Si un nodo avisa que ya está en el AGM, el nodo que pidió la información aprovecha para desechar la arista que los une y así no considerarla en las siguientes etapas.

4. Conociendo las aristas que aun pueden ser tomadas en cuenta, los nodos hojas del árbol eligen entre las suyas la de menor peso y la envían a su padre. Este espera las aristas de peso mínimo de sus hijos, elige la de menor peso entre ellas y la suya propia y a su vez la envía a su padre. Así sucesivamente hasta que el nodo INIC recibe las aristas elegidas por sus hijos y encuentra la de menor peso entre ellas y la suya .

5. El nodo INIC avisa al nodo del árbol incidente a la arista elegida, que esa es la arista de menor peso global para que la marque como incluída en el árbol y para que le avise al nodo al otro lado de la arista que ya forma parte del AGM. Después de hacerlo, el nodo avisa a INIC que ya incluyó la nueva parte del AGM.

   Una vez que el nodo INIC recibe ese aviso, procede de la misma manera con la siguiente fase hasta que todos los nodos estén en el AGM.

6. El nodo INIC avisa a través de las aristas del árbol que ya terminó el cómputo del AGM. Cada nodo sabe cuales de sus aristas incidentes forman parte del mismo pues las fue marcando en las distintas fases.

5.2.3 Estructuras de datos

- Cada procesador, incluido INIC, debe tener una bandera de INCLUIDO/NO para saber si forma parte del AGM o todavía no.

- Cada nodo, incluido INIC, tiene una lista de sus aristas incidentes incluyendo el peso de cada una y un campo para señalar si está en el AGM, si ya no hay que considerarla o si todavía no se sabe nada sobre ella.

Los valores son MARCADA, EXCLUIDA y NOMARCADA respectivamente.

5.2.4 Mensajes

Los mensajes que se intercambian entre los procesadores son de los siguientes tipos:

- REQMIN: Mensajes enviados por el nodo INIC a todos los nodos ya incluidos en el árbol para pedirles sus aristas APM's (paso 2)).

[5] Agregarla al árbol implicaría formar un ciclo.

- **REQINCI:** Cada nodo del árbol pregunta a sus nodos adyacentes a través de sus aristas NOMARCADAS incidentes si ya están incluidos o no en el subárbol del AGM construido hasta el momento (paso 3)).

- **RESPINCI:** Los nodos adyacentes responden a REQINCI (paso 3)).

- **RESPMIN:** Mensaje enviado por cada nodo del AGM a su padre en el árbol para identificar y avisar el peso de su APM (paso 4).

- **AVISO:** El nodo INIC avisa al nodo incidente a la arista de menor peso global que esa es la elegida (paso 5).

- **INCL:** El nodo que recibe el AVISO manda un mensaje al nodo del otro lado de la arista para incluirlo como parte del AGM (paso 5).

- **TERMFASE:** El nodo respectivo avisa al nodo INIC que ya concluyó la adición del nuevo nodo al AGM para que proceda con la siguiente fase (paso 5).

- **FIN:** El nodo INIC avisa a través de las aristas del árbol que ya finalizó el cómputo del AGM (paso 6).

5.2.5 Complejidad en número de mensajes

Analizaremos que sucede con cada uno de los mensajes.

- REQMIN: Estos mensajes viajan a través del subárbol ya construído. En la primera fase serán 0 [6], en la segunda será 1 y en la última serán $n-1$. De aqui,

Número de mensajes REQMIN $= 0+1+\ldots+n-1 = (n-1)n/2 = O(n^2)$

- REQINCI: En el peor caso, cada nodo es adyacente a los $n-1$ restantes. En la segunda fase, ya hay dos nodos en el AGM con a lo sumo $n-2$ aristas NOMARCADA incidentes a cada uno de ellos o sea se envian a lo sumo $2(n-2)$ mensajes y así sucesivamente. En la j-esima fase, es decir cuando se han agregado ya j nodos, un nodo puede seguir teniendo a lo sumo $n-(j-1)$ aristas incidentes no marcadas pues estas son las aristas que van a nodos aun no incluídos en el árbol [7] que en esta fase son justamente $n-j$. El $j-1$ viene del hecho de que posiblemente el nodo incluído en la etapa $j-1$ haya sido uno adyacente a él y recién podrá darse cuenta de ello cuando le mande este mensaje en la etapa j). De donde,

Número de mensajes REQINCI $= O(n^2)$

- RESPINCI: Nuevamente el número de estos mensajes es $O(n^2)$. Por cada arista por la que se envía un mensaje REQINCI, se devuelve uno de estos.

- RESPMIN: En cada etapa se envía este mensaje por cada arista del subárbol ya construido, 0 en la primera etapa, 1 en la segunda hasta que en la última se envían $n-1$ pues cada

[6] En el nodo INIC simplemente

[7] Recordar que tanto una arista MARCADA como una EXCLUIDA tienen sus dos nodos extremos en el árbol.

nodo del árbol manda uno solo de estos mensajes a su padre y no hay la necesidad de que sea retransmitido. Entonces,

Número de mensajes RESPMIN = $O(n^2)$

- AVISO: Hay $O(n^2)$ pues para agregar una nueva arista en cada fase, se los envía por un solo camino del árbol y hay n fases.

- TERMFASE: Este mensaje responde al anterior a traves de las mismas aristas. Se tiene $O(n^2)$.

- INCL: Hay $n-1$ mensajes pues se incluyen $n-1$ nodos al árbol.

- FIN: Se envían $n-1$ de estos mensajes por el árbol para que lleguen a todos los nodos.

Por tanto:

$$\text{Complejidad total en número de mensaje} = O(n^2)$$

5.2.6 Complejidad en tiempo

Para analizar el tiempo que lleva este algoritmo, hay que notar que cada fase, que corresponde a agregar un nuevo nodo, debe completarse antes de empezar la siguiente. Por lo tanto el tiempo total será la suma del tiempo que lleva cada fase. A esto se agrega el tiempo que lleva enviar el mensaje FIN al termino de todas las fases.

Recordemos que consideramos como unidad de tiempo el que le lleva a un mensaje viajar de un nodo a otro. El trabajo interno de los nodos se limita básicamente a comparar los pesos de las aristas y a actualizar el estado de éstas por lo que no lo tomaremos en cuenta. Se considera que su ejecución no afecta significativamente el tiempo total.

- Paso 2: En la primera etapa no se consume tiempo en el envío de ningún mensaje, en la segunda lleva una unidad de tiempo, etc. En la última, en el peor caso, es decir si el árbol es una cadena, lleva n-1 unidades de tiempo. Entonces,

Tiempo del paso 2 = $O(n^2)$

- Paso 3: El requerimiento se envía a todos los nodos adyacentes al mismo tiempo (en una unidad de tiempo). Sin embargo, el nodo que envió el mensaje REQINCI debe recibir todas las respuestas de sus nodos incidentes antes de proceder que en la j-esima etapa son, como ya se determinó, $n-(j-1)$ en el peor caso. Así,

Tiempo del paso 3 = $O(n^2)$

- Paso 4: En este paso, cada nodo -menos las hojas- espera los mensajes de sus hijos para enviar el suyo propio. Los únicos mensajes que podrán ser enviados en paralelo son los de los nodos tal que ninguno sea descendiente (o antecesor) el uno del otro. Es decir que pertenezcan a distintas ramas del árbol.

En el peor caso,

Tiempo paso 4 $= O(n^2)$

Ahora bien, estos pasos pueden solaparse ya que un nodo puede recibir las aristas de menor peso de uno o varios de sus hijos antes de recibir RESPINCI de todos sus nodos adyacentes o bien un nodo puede ya estar enviando su respuesta antes que otro haya recibido el requerimiento. Estas posibilidades no presenta la necesidad de controles adicionales pues de todas maneras un nodo debe esperar las respuestas de todos sus hijos y de todas sus aristas incidentes para proceder [8].

Se ha dividido el análisis en los pasos separados por simplicidad. De todas maneras el posible sobrelape en el envío de los mensajes no reduce la complejidad de tiempo, en el peor caso, de ninguno de los dos pasos.

- Paso 5: Para el envío de los mensajes de AVISO e INCL se toma el camino en el árbol que vaya de INIC al nodo respectivo. Considerando el peor caso, tenemos,

Tiempo paso 5 $= O(n^2)$

Es decir, la complejidad total de agregar todos los nodos al AGM es $O(n^2)$ más el tiempo de enviar el mensaje FIN que es O(n) tenemos:

$$\textbf{Complejidad en tiempo del algoritmo} = O(n^2 + n) = O(n^2)$$

5.2.7 Análisis segundo algoritmo

Respecto al tiempo, esta versión es bastante mejor que la primera versión pero sigue sin ser óptima. Es evidente que el que el algoritmo sea básicamente secuencial pesa en que el tiempo de ejecución no sea óptimo. Por ejemplo, un nodo que ya terminó de hacer su trabajo en cierta fase permanece inactivo hasta que *todos* los demás terminen su parte y recien se pueda pasar a la siguiente fase.

Notemos que el requerimiento de avisar al nodo INIC que ya se incluyó el nuevo nodo en el AGM para que aquel inicie la siguiente etapa se debe a que la velocidad de transmisión en las distintas aristas es variable y que el algoritmo no controla en ningún momento que alguna información sobre el árbol no esté actualizada (por ejemplo, que en una etapa se pregunte a un nodo si ya forma parte del árbol y que éste responda que no aun cuando esté incluído en el mismo por no haber recibido todavía el mensaje de INCL apropiado.

El número de mensajes, por su parte, ha sido reducido de $O(n^3)$ a $O(n^2)$ respecto al anterior algoritmo. Esto se debe básicamente a que el nodo INIC no requiere conocer información de toda la red. A pesar que se necesita enviar mensajes durante el desarrollo de todo el algoritmo, éstos son significativamente más cortos que los que se necesitaban en la primera versión, aunque en aquella solo se envían antes y después del cómputo del AGM.

Sin embargo, esta característica del algoritmo no sería suficiente para disminuir la complejidad de mensajes si no se tomara la decisión de no enviar las APMs de todos los nodos del

[8] Esta característica asegura que ningún nodo pueda mandar información no actualizada.

árbol hasta el nodo INIC. De hacerlo así, cada una de ellas debería ser reenviada por los nodos intermedios entre un nodo y el INIC lo que llevaría a $O(n^3)$ mensajes RESPMIN [9].

Una mejora adicional, que también evita el aumento en la complejidad de mensajes es la de ir desechando las aristas que seguro ya no formarán parte del AGM pues si no fuese así, un nodo tendría que observar todas sus aristas incidentes que en el peor caso son n en cada etapa. Solo no se consideraría las que ya tiene incluídas en el árbol - que pueden ser 0 o muy pocas al menos en las primeras etapas del algoritmo- .

Por último recalcamos, que en cierta etapa un nodo no necesariamente conoce todas sus aristas incidentes que ya no serán consideradas pues el algoritmo no avisa a los nodos del AGM que arista se incluyó en una fase, salvo al nodo incidente a ella (El resto se entera solo en la siguiente fase al enviar REQINC). Otra opción sería avisarles a todos y en ese momento cada nodo observaría su lista de aristas para actualizarla en caso necesario. Sin embargo este cambio no introduce mejoras ni a nivel tiempo ni a nivel de número de mensajes.

5.3 Conclusiones

Aunque la segunda versión es claramente superior a la primera, aún adolece de ciertos problemas.

Una de las principales desventajas de ambos algoritmos es la realización del trabajo en forma centralizada: Se desaprovecha la capacidad de procesamiento de todos los nodos, las tareas se realizan básicamente en forma secuencial y los mensajes 'viajan' casi siempre por toda la red. Es poco conveniente sobrecargar de trabajo a un nodo ya que éste se convierte en un cuello de botella degradando notoriamente el desempeño general del sistema y haciéndolo menos tolerante a fallas; lo más probable es que un problema en el procesador que realiza la tarea haga que el algoritmo ya no se pueda ejecutar aun cuando el resto de los procesadores estén funcionando correctamente.

Si distribuímos el trabajo equitativamente en grupos de procesadores que vayan encontrando soluciones parciales que combinadas permitan obtener la solución global, tendremos mayor paralelismo y en su mayoría los mensajes viajarán internamente dentro de cada grupo y no por toda la red. Obviamente estas características no aseguran por si solas que el algoritmo que las tenga sea bueno. Hay que usarlas de una forma inteligente.

[9] En el peor caso, en la segunda etapa se enviarían 1 mensaje, en la tercera $1+2=3$, en la cuarta $1+2+3=6$ y así hasta que en la última se enviarían $1+2+3...+n-1$ mensajes o sea $1+3+6+...+(n-1)n/2 = O(n^3)$ mensajes.

44

CAPITULO 6

EL ALGORITMO GHS:DE LAS IDEAS BASICAS AL CODIGO

Los capítulos desarrollados hasta ahora nos han permitido construir el entorno del algoritmo de Gallager, Humblet y Spira [GHS]: Discutir la importancia y el uso del problema de hallar un árbol generador de peso mínimo en general, presentar aproximaciones centralizadas para resolverlo y revisar ciertas definiciones y propiedades pertinentes de Teoría de Gráficas y Sistemas Distribuidos de manera que ahora sí podamos concentrarnos específicamente en el estudio del algoritmo.

Este algoritmo sigue una política distribuida evitando los problemas de una aproximación centralizada que vimos en el capítulo 5. Sin embargo, trabajar distribuidamente no es una condición suficiente para asegurar que el algoritmo sea bueno; es más, tener procesadores trabajando independiente y, como en este caso, asíncronamente obliga a controlar una serie de situaciones que en un algoritmo centralizado no se hubieran presentado. Gallager, Humblet y Spira usan ideas muy interesantes que les permiten obtener un algoritmo correcto y eficiente donde la presencia de cada decisión, cada principio y cada paso está cuidadosamente diseñada y plenamente justificada.

Si bien una primera aproximación puede dejar la falsa impresión de que es sencilla y hasta obvia su manera de trabajar, desde las ideas centrales hasta los detalles pequeños involucran aspectos destacables y no tan fáciles de entender. Más aún, cercionarse de que el algoritmo logre su objetivo combinando toda esta multitud de detalles, que interactuan entre si, es una tarea complicada. Por ello decidimos hacer un 'refinamiento sucesivo', es decir un acercamiento a su construcción empezando por los conceptos más abstractos y terminando en detalles de naturaleza más bien implementativa.

6.1 La idea general

La idea, ampliamente usada en computación, es dividir el problema en subproblemas cuyas soluciones puedan combinarse de cierta manera para que al final, se obtenga la solución completa.

Para utilizar esta técnica de programación en un problema específico se debe decidir cuidadosamente como se dividirá el problema y luego como se combinarán las soluciones de cada parte para obtener la del problema original. La clave en la reducción de la complejidad de la solución final está en lograr que los subproblemas sean más o menos del mismo tamaño y que sea sencillo combinar las soluciones parciales.

En el caso de gráficas la forma más común es dividir la gráfica en subgráficas sobre las cuales se aplica un algoritmo para obtener las semisoluciones a partir de las cuales construimos la solución correspondiente a la gráfica completa. En particular, para asegurar la correctez de su algoritmo distribuido Gallager, Humblet y Spira utilizan el teorema 2.3.3. visto en el capítulo 2.

Durante la ejecución del algoritmo, se tiene a cada momento un conjunto de fragmentos que cubre todos los nodos de la gráfica formando un bosque generador sobre el cual se aplica la propiedad citada. Se comienza con n fragmentos, cada uno formado por un solo nodo y en cada nueva fase los nodos se unen en fragmentos de mayor tamaño hasta que se tenga uno solo que incluya todos los nodos de la red.

Cada fragmento F constituye el árbol generador de peso mínimo de sus nodos. Estos nodos cooperan en un algoritmo distribuido para encontrar la arista de salida de peso mínimo (ASPM) del fragmento completo.

Estas aristas servirán para ir conectando fragmentos en fragmentos de mayor tamaño hasta que se tenga uno solo con todos los nodos de la gráfica conectados por las aristas del árbol generador de peso mínimo total (ver figura 6.1).

Así, el aplicar el teorema 2.3.3. cada vez que elegimos la arista de salida de menor peso de cada fragmento que se vaya formando desde el inicio [1], coadyuva a asegurar que elegimos las aristas que formarán parte del AGM de la gráfica original.

Sin embargo esto no es suficiente pues estamos suponiendo que, inicialmente cada nodo y, conforme se van construyendo, cada fragmento elige su ASPM independientemente de lo que estén haciendo los otros, lo que podría llevar a una situación como la que se ve en la figura 6.2.

F_1, F_2, F_3 tienen cada uno dos aristas de menor peso (peso=1) de manera que cada cual puede elegir arbitrariamente una de ellas. F_1 elige a A al mismo tiempo que F_2 elige a B y F_3 a C. Obviamente no podemos incluir las tres aristas en el árbol generador de peso mínimo porque se formaría un ciclo. Para evitar este problema es importante la condición incluída en nuestro modelo estándar de tener aristas con pesos distintos pues así podemos aplicar el

[1] Recordemos que al inicio se tiene un bosque donde los fragmentos son los nodos solos.

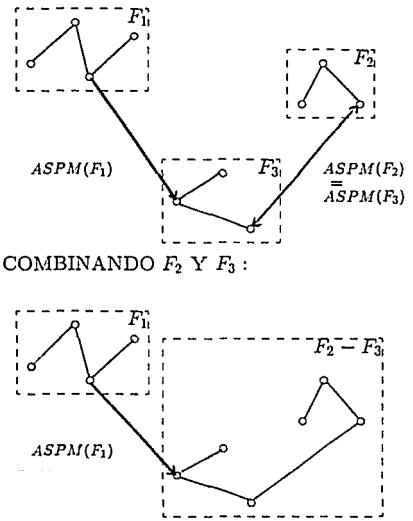


Figura 6.1: COMBINACION DE FRAGMENTOS

teorema 2.3.4. del capítulo 2 y asegurarnos que, sin importar el orden en que se vayan eligiendo las ASPMs de los distintos fragmentos, todas ellas estarán en el AGM.

6.2 Balance: Operaciones de FUSION y ABSORCION

Las dos propiedades usadas aseguran, en principio, que el algoritmo funcione correctamente pero además nos interesa que lo haga enviando el menor número de mensajes posible. Qué sucedería si un fragmento va comiendo todos los nodos uno a uno? (ver figura 6.3)

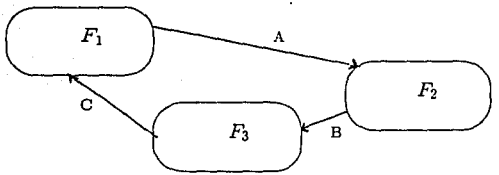


Figura 6.2: Peso(A)=Peso(B)=Peso(C)=1

Esta situación puede darse debido a que los procesadores y las distintas líneas de comunicación trabajan a distintas velocidades. El fragmento grande envía los mensajes más rápidamente y 'gana' a los demás en ir agregándose nodos. Cada nuevo nodo que se adiciona involucra el intercambio de información entre los nodos del fragmentos grande, de hecho el número de mensajes que se requiere enviar es proporcional al tamaño del fragmento. En este caso se envían al menos $1 + 2 + 3 \ldots + (n-1) = O(n^2)$.

Este problema sugiere la necesidad de mantener, en lo posible, fragmentos de aproximadamente igual tamaño que crezcan rápidamente (no de uno en uno como en el caso de la figura 6.3). Para mantener este crecimiento balanceado se intenta combinar fragmentos de más o menos el mismo tamaño mediante la operación de:

FUSION(MERGE)

Para que dos fragmentos F_1, F_2 se puedan fusionar, se requiere que tengan aproximadamente el mismo tamaño y compartan la misma ASPM.

La condición de que tengan aproximadamente el mismo número de nodos asegura que al fusionarse formen un fragmento F_{12} con al menos el doble del tamaño del fragmento más chico entre F_1 y F_2. Una situación totalmente distinta a la vista en la figura 6.3 donde el tamaño del fragmento crecía apenas de uno en uno.

La otra condición parecería estar de más pues basta que un fragmento elija una arista como su ASPM para que ésta esté en el AGM; sin embargo la coincidencia de las ASPMs es un importante requisito para controlar el crecimiento de los fragmentos. Si no fuera así, podría suceder que un fragmento F_1 encuentre que su ASPM va a otro F_2 y éste a su vez encuentre que su ASPM va a F_3, con F_1, F_2, F_3 aproximadamente de igual tamaño.

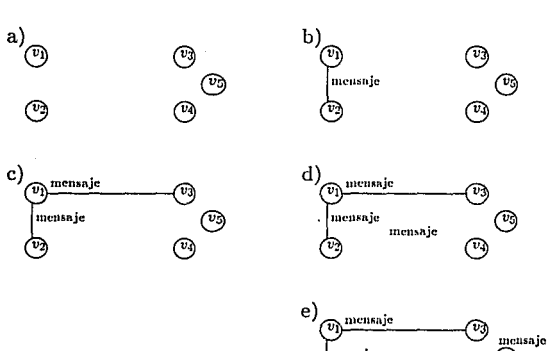


Figura 6.3: Un fragmento adicionandose todos los nodos

Supongamos que F_1 y F_2 se fusionan primero e inmediatamente después F_3 (ver figura 6.5). Nuevamente tendríamos un fragmento grande agregándose uno más chico. Incluso la cadena podría ser más larga y tendríamos que el fragmento $F_1 - F_2$ se 'come' a F_3, $F_1 - F_2 - F_3$ a F_4 y así sucesivamente.

Ahora bien, la forma en que estén repartidos los pesos en las aristas puede forzarnos a unir fragmentos de -posiblemente muy- diferente tamaño como en el ejemplo de la figura 6.6.

v_2, v_3, v_4, v_5 tienen sus ASPM hacia v_1 y por tanto los nodos no se podrán poner de acuerdo de dos en dos. Inicialmente v_1 se fusionará con v_2 formando un fragmento de tamaño 2 al cual se querrán adicionar fragmentos de tamaño 1 (v_3, por ejemplo).

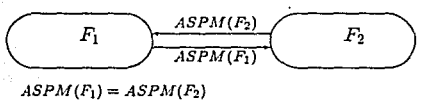


Figura 6.4: Condiciones para una fusión

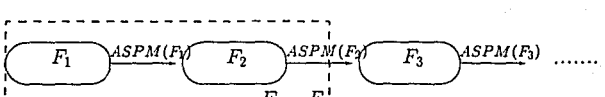


Figura 6.5: Cadena de fragmentos queriendo fusionarse al siguiente

Adicionalmente, los fragmentos trabajan asíncronamente a distintas velocidades y puede suceder que fragmentos pequeños (que sean más 'lentos') queden rezagados.

Aún así queremos controlar el número de mensajes que se envían y asegurar que el tamaño de un fragmento al menos se duplique al unirse con otro. Para lograrlo no interpretamos la situación como que un fragmento grande decide adicionarse uno pequeño sino como que es el pequeño el que decide incluirse ('absorberse') al grande. Aunque esto parezca simplemente otra forma de decir lo mismo, la idea subyacente es más profunda y constituye una de las claves del algoritmo: En vez de enviar mensajes a todos los nodos del fragmento grande, son los pocos nodos del fragmento chico los que tienen que cooperar para hallar la ASPM global del fragmento y, a través de ella, unirse al fragmento grande (ver figura 6.7) asegurándose además de al menos duplicar su tamaño. Para estos casos se utiliza la operación:

ABSORCION(ABSORB)

Para que un fragmento F_1 se pueda absorber a otro F_2, el tamaño de F_1 debe ser menor

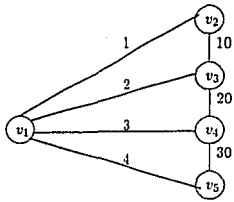


Figura 6.6:

que el de F_2 y su ASPM debe ir a un nodo de F_2 (ver figura 6.8).

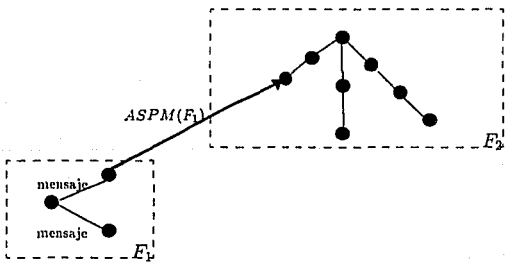


Figura 6.7: El intercambio de mensajes se realiza dentro del fragmento pequeño

6.2.1 Primera aproximación a la correctez del algoritmo

Las propiedades en las que se basa el algoritmo nos aseguran su correctez sin tomar en consideración los aspectos dinámicos del algoritmo, vale decir su evolución en el tiempo al ejecutarse

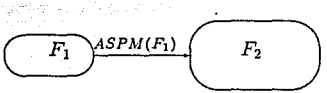


Figura 6.8: F_1 decide unirse a F_2

sobre una red. Ahora nos interesa ver si las únicas operaciones que se permiten para combinar fragmentos, las de fusión y absorción, son suficientes para que el algoritmo funcione correctamente y termine encontrando el AGM de la gráfica.

Afirmación 6.2.1 *Si tenemos una situación inicial en la cual cada fragmento consiste de un único nodo y aplicamos cualquier secuencia de operaciones fusión y absorción, entonces siempre es posible seguir aplicando al menos una de las dos operaciones hasta finalizar con un fragmento que contenga todos los nodos.*

Demostración Vamos a demostrar que en cualquier estado al que se llega después de haber aplicado una serie de fusiones y absorciones, podemos aplicar al menos una de las dos operaciones.

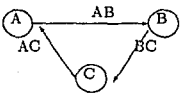
Para verlo, imagemos que paramos la ejecución del algoritmo en un momento dado t y según la configuración obtenida hasta t, construímos una "digráfica de fragmentos" $\dot{G} = (\dot{V}, \dot{E})$ donde:

$\dot{V} = \{v_f$ tal que f es un fragmento en t$\}$ y

$\dot{E} = \{(v_{f1}, v_{f2})$ si la ASPM de f1 va a un nodo de f2$\}$

Claramente $\dot{G}$ es una digráfica con igual número de nodos que de aristas ($|\dot{V}| = |\dot{E}|$) y por tanto tiene al menos un ciclo. Los ciclos posibles tienen longitud 2 ya que las aristas tienen pesos distintos. Si existiera un ciclo de longitud 3 o más, llegaríamos a una situación como la que se ve en la figura 6.9 que, por supuesto, no está permitida. De ahí que la digráfica de fragmentos $\dot{G}$ se vea como en la figura 6.10.

Si al menos una de las ASPMs no incluídas en un ciclo equivale a una absorción se puede aplicar inmediatamente esa operación. En caso contrario, ninguna de esas ASPMs corresponde a una operación aplicable en t pues cada una es una ASPM que va de un fragmento grande a



$peso(BC) < peso(AB)$ pues B la eligió.
$peso(AB) < peso(AC)$ pues A la eligió.
$peso(AC) < peso(BC)$ pues C la eligió.
y llegaríamos a $peso(BC) < peso(BC)$.

Figura 6.9: Un ciclo de longitud tres llevaría a una contradicción

Figura 6.10: Ciclos de longitud dos en $\dot{G}$

uno más chico o bien a un pedido de fusión no correspondido . Sin embargo al menos podemos estar seguros que los dos fragmentos de un ciclo de longitud 2 se pueden combinar. Si ambos tienen el mismo tamaño, se pueden fusionar pues sus ASPMs coinciden. Si son de distinto tamaño el más pequeño se puede absorber al otro. ■

6.3 Niveles e identificadores de los fragmentos

NIVEL.-

Al describir las operaciones de absorción y fusión hemos supuesto que se conocen los tamaños de los fragmentos y que por tanto sabemos cual de las dos operaciones se aplica en un caso dado.

Sin embargo, cuando la gráfica con la que estámos trabajando es, en la práctica, una red que sigue nuestro modelo estándar, el saber el tamaño exacto de cada fragmento presenta una gran dificultad. Tanto contar los nodos de un fragmento en cuanto éste se forme, como sumar los tamaños de los fragmentos que lo componen es algo muy difícil tomando en cuenta que el algoritmo es muy dinámico y los fragmentos se están uniendo continuamente. Mientras se está

contando los nodos o recopilando la información del tamaño de los fragmentos componentes, nuevos fragmentos pueden absorberse al fragmento cambiando su tamaño y posiblemente invalidando lo contado hasta el momento; lo que puede suceder continuamente. Esto sin mencionar el número de mensajes adicionales requeridos para el conteo.

A pesar de ello, uno de los principios fundamentales del algoritmo sigue siendo la combinación de fragmentos de aproximadamente igual tamaño mientras sea posible y, en caso contrario, la absorción del más pequeño al más grande. Por tanto se debe contar al menos con una estimación del tamaño de los fragmentos . GHS lo logra asociando un *NIVEL* a cada fragmento.

Con él, las reglas de aplicación de una absorción o una fusión se redefinen de la siguiente forma:

- F y F' se fusionan si cumplen las dos condiciones siguientes:
  a) $nivel(F) = nivel(F')$
  b) $ASPM(F) = ASPM(F')$
  y el fragmento resultante de la fusión de F y F' tiene nivel igual a $nivel(F) + 1$.
- F se absorbe a F' si se cumple que:
  a) $nivel(F) < nivel(F')$
  b) $ASPM(F)$ va a F'

El *NIVEL* de cualquier fragmento formado a lo largo del algoritmo da una cota inferior del logaritmo del número de sus nodos y solo se incrementa al aplicar una fusión.

Afirmación 6.3.1 *Sea F un fragmento cualquiera*

$$nivel(F) = l \Rightarrow No.\ nodos(F) \geq 2^l$$

Demostración Se hace la prueba por inducción sobre l.

BASE: $l = 0$: Los únicos fragmentos que tienen nivel 0 son los formados por un único nodo. Se cumple que $No.\ nodos \geq 2^0 = 1$.

PASO INDUCTIVO: Supongamos que se cumple para l. Un fragmento de nivel $l+1$ resulta de la fusión de dos fragmentos de nivel l, F_1 y F_2.

Por hipótesis,

$$No.\ nodos(F_1) \geq 2^l$$
$$No.\ nodos(F_2) \geq 2^l$$

Luego:

$$\begin{aligned} No.\ nodos(F) &= No.\ nodos(F_1) + No.\ nodos(F_2) \\ &\geq 2^l + 2^l \\ &\geq 2^{l+1} \end{aligned}$$

Adicionalmente, si al fragmento F se absorben uno o más fragmentos de nivel menor que $l+1$, el número de nodos de F se incrementa y sigue siendo mayor a 2^{l+1}. ■

Notemos que las absorciones hacen que fragmentos de nivel l puedan ser mucho más grandes que 2^l, incluso puede suceder que un fragmento de menor nivel pueda tener más elementos que uno de nivel más alto, vale decir:

$$nivel(F) < nivel(F') \not\Rightarrow No.nodos(F) < No.nodos(F')$$

Sin embargo resulta que los niveles son una aproximación suficientemente buena para lograr la complejidad deseada.

IDENTIFICACION DE FRAGMENTOS

Notemos que cada fragmento debe saber cuales de las aristas incidentes a sus nodos son de salida para elegir entre ellas la ASPM y por tanto cada fragmento debe tener un identificador.

El nivel no solo sirve para mantener el balance en el tamaño de los fragmentos sino para identificar el fragmento al que pertenece un nodo. Para distinguir fragmentos de un mismo nivel, se usa lo que se llama *arista memoria (core)* que es la arista sobre la cual se realizó la fusión que dio lugar al fragmento.

Así el identificador de un fragmento F es el par (nivel(F), arista-memoria(F)).

Las reglas para actualizar el identificador de los fragmentos se pueden resumir en lo siguiente:

1. Si F_1 y F_2, ambos de nivel l, se fusionan por la arista e para formar el fragmento F_{12}: Identificador(F_{12}) = (l+1,e)
2. Si F_1 de nivel l se absorbe a través de la arista e_1 a F_2 con Identificador(F_2) = $(k,e), k > l$, el nuevo fragmento conserva como identificador la pareja (k,e).

Ahora bien, el identificador de un fragmento F debe ser propagado a todos los nodos de F para que éstos sepan a que fragmento pertenecen y entonces puedan saber cuales de sus aristas incidentes son de salida preguntando a cada nodo adyacente su identificador y luego comparándolo con el suyo. El tiempo que consume la propagación implica que, al hacer tal comparación, se consideren tres posibles casos (ver figura 6.11).

i. arista-memoria(p) = arista-memoria(q) y nivel(p)=nivel(q) entonces q está en el mismo fragmento que p. No hay dos fragmentos que puedan tener la misma arista-memoria ya que a través de una arista se puede hacer a lo sumo una fusión.

ii. arista-memoria(p) ≠ arista-memoria(q) y nivel(p) < nivel(q) aún existe la posibilidad de que q esté en el mismo fragmento pero que todavía no lo sepa, la información sobre la identidad del fragmento ha sido propagada a p pero q aún no la ha recibido. En este caso q retrasa su respuesta a p hasta que su nivel sea mayor o igual al de p.

iii. Si arista-memoria(p) $\neq$ arista-memoria(q) y nivel(q) $\geq$ nivel(p) entonces seguro que están en fragmentos distintos. En el curso del algoritmo un nodo sólo puede estar en un fragmento de un nivel particular, es decir q no puede llegar a ser del fragmento de p pues q está o ya pasó el nivel de p.

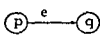


Figura 6.11:

6.3.1 Segunda aproximación a la correctitud del algoritmo

Ya que al demostrar en la anterior sección que el algoritmo termina encontrando el AGM no consideramos el retardo que puede tener la respuesta de q, es necesario preguntarse si esta espera no puede llevar a una situación donde cada fragmento esté esperando por otro y, consiguientemente, no sea posible ninguna absorción o fusión. Para ver que ese no es el caso, se usa esencialmente el mismo argumento que antes, pero esta vez solo necesitamos considerar las ASPMs de los fragmentos de menor nivel en la digráfica de fragmentos -llamémoslo nivel MIN-.
Si un fragmento de nivel MIN encuentra que su ASPM va a un fragmento de mayor nivel, entonces una operación de absorción es posible y si todos los fragmentos de nivel MIN tienen su ASPM hacia otro fragmento de nivel MIN, entonces debe existir al menos un ciclo entre dos fragmentos de nivel MIN y ambos se pueden fusionar.

6.4 Descripción general del algoritmo

Daremos una descripción de "alto nivel", sin entrar aún en mucho detalle, del algoritmo de GHS.

Cada fase del algoritmo consiste de dos etapas que se realizan en cada fragmento hasta que se tenga uno solo con todos los nodos.

Sea F un fragmento:

ETAPA I. Encontrar la ASPM de F

1. La arista_memoria avisa a cada nodo del fragmento que debe encontrar su ASPM local.
2. Un nodo que recibe el requerimiento observa todas sus aristas incidentes y elige la de menor peso entre las que salgan del fragmento.

3. Cada nodo envía la información de regreso hacia la arista_memoria.

4. Cuando ésta recibe toda la información, elige la ASPM de todo el fragmento.

ETAPA II. Unión de fragmentos

1. Se pide al nodo extremo de la ASPM que está en F que la incluya en el AGM.

2. A través de la ASPM se envía un pedido de unión al otro fragmento que, según el caso se convertirá en fusión o absorción.

Notemos que la arista_memoria sirve, además de para identificar el fragmento, como sitio de origen del proceso y donde se colecciona la información requerida.

La figura 6.12 puede ayudar a entender el flujo de información que se está generando al interior de F.

Claramente en un fragmento de nivel 0, es decir uno formado por un único nodo, el algoritmo es mucho más sencillo. El nodo que lo forma elige su arista incidente de menor peso y a través de ella realiza un pedido de unión al otro fragmento.

6.5 El algoritmo en más detalle

Nuestro objetivo ahora es describir cada uno de los pasos ejecutados por el algoritmo, la información que tienen que intercambiar los nodos y, de acuerdo a ella, las acciones que ejecutan.

6.5.1 Los mensajes

Para simplificar la explicación, listaremos a continuación los mensajes enviados en el algoritmo:

INICIAR (INITIATE) Aviso enviado por la arista_memoria de un fragmento a los demás nodos para que cada uno busque su arista de salida de menor peso.

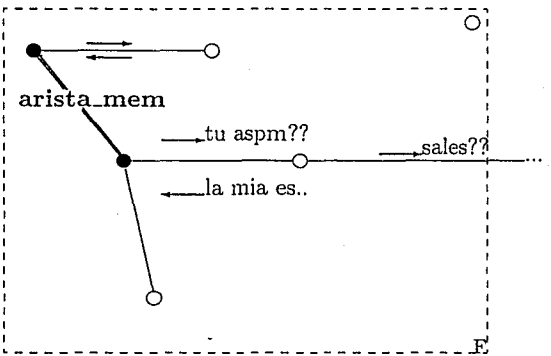
REPORTE (REPORT) Con este mensaje cada nodo envia la arista que encuentra como respuesta al requerimiento anterior.

TEST Lo envía cada nodo a sus nodos adyacentes para averiguar si éstos están en su fragmento o no.

DIFERENTE (ACCEPT) El nodo que responde está en un fragmento distinto al del que le envió el TEST.

IGUAL (REJECT) Respuesta de un nodo a un mensaje TEST señalando que está en el mismo fragmento.

ETAPA I



ETAPA II

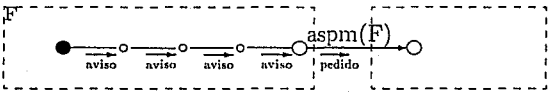


Figura 6.12: Esquemas de las etapas I y II del algoritmo

CAMBIO-RAIZ (CHANGE-ROOT) El mensaje enviado al nodo extremo de la ASPM en el fragmento, una vez que ésta fue hallada, para incluirla en el árbol.

CONECTAR (CONNECT) Mensaje enviado a través de la ASPM cuando un fragmento se quiere combinar con otro.

6.5.2 Etapa I

Sea F un fragmento tal que $Identificador(F) = (l, e)$ es decir de nivel l (mayor a 0) y con arista_memoria $e = (v_1, v_2)$. Los pasos que se siguen en F para ejecutar la primera etapa son:

1. Los dos nodos extremos de la arista_memoria, v_1 y v_2 difunden el mensaje INICIAR a todos los nodos de sus subárboles respectivos. Este mensaje no solo le avisa a cada nodo que debe buscar su arista de salida de menor peso sino que también le informa la identidad del fragmento.

2. Al recibir el mensaje INICIAR, cada nodo v de F realiza el siguiente protocolo ("TEST-REJECT-ACCEPT protocol").

   (a) v revisa sus aristas incidentes y lo hace enviando por ellas mensajes TEST en los que incluye $Identificador(F)$.
   Para evitar mensajes redundantes, v mantiene una lista de sus aristas incidentes ordenadas de menor a mayor tamaño. En esa lista se asocia a cada arista, su 'estado' que puede ser uno de los tres siguientes:
   - MARCADA: Arista que ya ha sido incluída como arista del fragmento, por tanto del AGM.
   - RECHAZADA: Arista que ya no puede estar en el AGM pues el nodo al otro extremo de la arista también está en él y su inclusión formaría un ciclo.
   - BASICA: Arista de la cual v no sabe nada todavía.
   v recorre la lista (en orden creciente de pesos) hasta hallar la primera arista BASICA, llamemósla $e' = (v, v')$

   (b) v envía TEST(l,e) por e'.

   (c) Al recibir el mensaje TEST, v' compara el identificador de su fragmento con el incluído en tal mensaje.
   Si son iguales, envía un mensaje IGUAL a v.
   Si su arista_memoria es distinta a la enviada (e) y su nivel es mayor o igual a l , envía un mensaje DIFERENTE a v.
   Por último, si su arista_memoria es distinta y su nivel es menor que l, retarda su respuesta hasta que pueda enviar un mensaje IGUAL o DIFERENTE.

   (d) Si v recibe IGUAL de v', v (y también v') marca e' como RECHAZADA, busca en su lista la siguiente arista BASICA y vuelve a realizar los dos pasos anteriores hasta que una arista le responda con un mensaje DIFERENTE o bien encuentre que ya no tiene aristas de salida.
   En el primer caso, es decir si recibe DIFERENTE de algún nodo adyacente, considera la arista que los une como su ASPM.
   En el segundo caso, define su ASPM como de peso infinito.

3. Los nodos de F envían sus ASPMs hacia v_1 o v_2, según el subárbol al que pertenezcan usando el siguiente proceso convergente:

   - Cada nodo hoja envía su ASPM a su padre en el fragmento.
   - Cada nodo interno espera las ASPMs de todos sus hijos y una vez que recibe todas, elige entre ellas y su propia ASPM, la de menor peso enviándola a su padre en el fragmento.

   Al final de este proceso v_1 (v_2) elige entre su ASPM y las que recibe de sus hijos, la arista de salida de menor peso del subárbol que tiene como raiz a v_1 (v_2).

4. v_1 y v_2 intercambian e_{v_1} y e_{v_2}, las aristas que hallaron en el paso anterior. Si $w(e_{v_1}) > w(e_{v_2})$, e_{v_1} es la ASPM global de F, en caso contrario lo es e_{v_2}.

6.5.3 Etapa II

Llamemos $e_f = (v_f, v_{f'}), v_f \in F, v_{f'} \in F' <> F$ a la arista de salida de menor peso del fragmento F hallada en la etapa I.

1. El nodo raiz del subárbol al que pertenece v_f, v_1 o v_2, le envía el mensaje CAMBIAR-RAIZ para que cambie el estado de e_f a MARCADA en su lista de aristas incidentes.

2. Luego de marcar e_f, v_f envía un mensaje CONECTAR a través de ella a $v_{f'}$ y F deja de funcionar como fragmento. Al absorberse o fusionarse, sus nodos pasan a formar parte de un fragmento más grande G.
   Notemos que los nodos de F no son notificados de ello en ese momento. Por tanto todas las aristas que van de un nodo en F a otro en G-F siguen marcadas como BASICAs hasta que G difunda INICIAR y cada nodo envíe los TESTs respectivos [2].

Las dos etapas se realizan hasta que los dos nodos de la arista_memoria detecten que ya no hay aristas de salida, es decir cuando todos los nodos del fragmento definan su ASPM como de peso infinito.

6.6 Algunas consideraciones más

6.6.1 Manipulación de los Apuntadores PADRE

Un detalle que no incluímos en la anterior sección para evitar complicar aun más la descripción de las dos etapas del algoritmo es la manipulación que se realiza sobre los apuntadores que definen la estructura de árbol que tiene el fragmento. Cada nodo v tiene asociado un apuntador PADRE que, como su nombre lo indica, apunta al nodo padre de v.

[2] Por eso decimos que las aristas BASICAs son aquellas de las que aun no se sabe nada

Usando la 'dirección' señalada por estos apuntadores es que la información se difunde desde la arista_memoria y luego converge en ella, como describimos anteriormente aunque sin mencionar explícitamente a los apuntadores.

En el curso del algoritmo, vimos que cada árbol o fragmento encuentra su ASPM $e_f = (v_f, v_{f'})$ y al unirse al árbol del otro lado de esa arista, se vuelve un subárbol de un árbol más grande. Esta unificación se representa moviendo la raiz del subárbol corespondiente (uno de los dos nodos de la arista_memoria = (v_1, v_2)) al extremo interno de la arista elegida (o sea a v_f), cambiando los apuntadores PADRE de acuerdo a ello al momento de enviar el CAMBIAR-RAIZ y creando uno nuevo que vaya de v_f a $v_{f'}$. Supongamos sin pérdida de generalidad que v_f está en el subárbol cuya raiz es v_1. Los apuntadores PADRE de los nodos que no están en el camino $v_1 \ldots v_f$ no se modifican. Los de los nodos que están en ese camino se revierten, vale decir que un apuntador que va de v_x a v_y se reemplaza por uno que va de v_y a v_x de tal manera que los nodos padres se convierten en nodos hijos a lo largo del camino $v_1 \ldots v_f$ (ver figura 6.13)

Notar que dos árboles se pueden unificar el uno al otro si ambos tienen la misma ASPM, es decir si se fusionan. En ese caso, esa arista se convierte en la arista_memoria resultante y sus dos extremos son raices de sus respectivos subárboles. Recordemos que ambos intercambian información sobre las ASPMs de sus subárboles y por tanto está bien que cada uno tenga su apuntador PADRE dirigido hacia el otro.

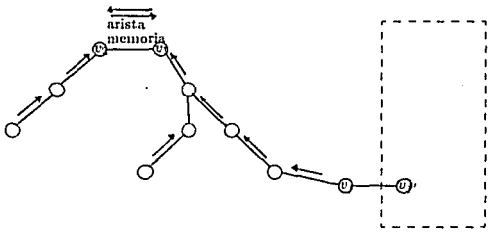
6.6.2 Cuando se forma un nuevo fragmento: Fusión

Vimos que al término de la segunda etapa, un fragmento F envía CONECTAR por su ASPM que se traduce, en cierto momento, en una operación de fusión o en una de absorción según el caso.

Sea $e_f = (v_f, v_{f'}), v_f \in F, v_{f'} \in F' <> F$. Si el nivel de F' es mayor que el de F, F se absorbe inmediatamente a F'. Sin embargo, si ambos fragmentos tienen igual nivel (digamos l), al momento de enviar el CONECTAR, F no puede saber si se absorbe o fusiona. Si la ASPM de F' es también e_f, habrá un 'cruce' de CONECTAR por esa arista dando lugar a una fusión. En caso contrario, F' será absorbido o se fusionará a otro fragmento pasando, en cualquiera de los dos casos, a formar parte de un fragmento F'' de mayor nivel que l y que eventualmente absorberá a F (ver figura 6.14).

Notemos que en cualquier caso, F realiza las mismas acciones y envía el CONECTAR por e_f independientemente de la operación que vaya a tener lugar sobre esa arista. Falta ver como se detecta que se ha formado un nuevo fragmento para empezar a aplicar el algoritmo sobre el mismo :

v_f sabe que ha enviado un CONECTAR por e_f así que si recibe por ella un CONECTAR de regreso, detecta que es uno de los extremos de la arista_memoria de un nuevo fragmento. Entonces incrementa su nivel en 1 (a l+1) y cambia su arista_memoria por e_f formando así el identificador del nuevo fragmento que luego difunde por el subárbol del cual es raiz mediante



Apuntadores PADRE modificados en el camino $v_1, \ldots, v_f$ y el nuevo de v_f a $v_{f'}$:

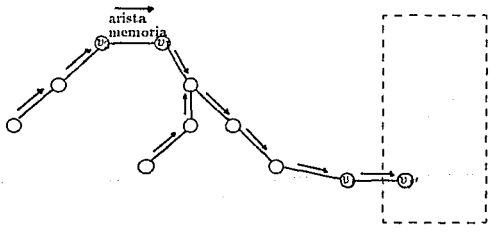


Figura 6.13: manipulación apuntadores PADRE

el mensaje INICIAR. Recíprocamente $v_{f'}$ sabe que recibió un CONECTAR por e_f entonces, luego de enviar CONECTAR por ella, procede con los mismos pasos que describimos para v_f.

Adicionalmente, si hay fragmentos F_i de nivel l que han enviado CONECTAR a nodos del fragmento F o F' antes que se fusionen, el mensaje INICIAR pasa también a ellos incluyéndolos en el nuevo fragmento. El mensaje también pasa a todos los fragmentos de nivel l que enviaron CONECTAR a algún nodo de los fragmentos F_i y así sucesivamente.

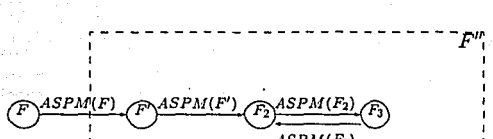


Figura 6.14: Pedido de fusión no aceptado

6.6.3 Las Absorciones

Que pasa si un fragmento de nivel l envía CONECTAR a otro F' de nivel $k > l$. Recordemos que al enviar CONECTAR por la arista $e = (v_f, v_{f'})$, v_f define su apuntador PADRE hacia $v_{f'}$, entonces el último mensaje INICIAR que haya recibido $v_{f'}$ será reenviado a través de e a v_f quien lo difunde por el subárbol del cual es raiz con lo que este subárbol se acopla al trabajo en el fragmento resultante.

Debido a la política de nunca hacer esperar a un fragmento de bajo nivel, $v_{f'}$ envía inmediatamente el mensaje INICIAR a v_f. Sin embargo, el algoritmo debe tomar en cuenta que es posible que el fragmento F se esté absorbiendo mientras F' está buscando su propia ASPM. Hay dos casos a considerar (para entender la notación ver figura):

1. Que ASPM($v_{f'}$) no haya sido aun determinada. En este caso se debe buscar la ASPM de F' en F también.
   Como $v_{f'}$ aun no conoce su propia ASPM, hay todavía la posibilidad que ésta sea la arista de salida de peso mínimo de todo el fragmento F' original y que coincida con e, vale decir:
   $$ASPM(v_{f'}) = ASPM(F'_{inic}) = e = ASPM(F)$$
   Entonces $ASPM(F'_{nuevo})$ puede emanar de uno de sus nodos recién incorporados (los de F):
   $$e = ASPM(F'_{inic}) \Rightarrow \forall e' \in F'_{inic}, peso(e') > peso(e) \text{ y}$$
   $$e = ASPM(F) \Rightarrow \forall e'' \in F, peso(e'') > peso(e)$$
   entonces podría suceder que:
   $$\exists e^x \in F \text{ tal que } \forall e' \in F'_{inic}\ peso(e^x) > peso(e')$$

2. $ASPM(v_{f'})$ ha sido ya encontrada al tiempo que F se absorbe a F'. En ese caso, $ASPM(v_{f'})$ no puede ser e ya que la única manera que ASPM($v_{f'}$) pueda conocerse es que llegue a un fragmento del cual ya se sabe que tiene un nivel al menos tan grande como el de F' y sabemos que $nivel(F) < nivel(F')$:

$$ASPM(v_{f'}) \neq e \Rightarrow ASPM(F') \neq e$$

y

$$e = ASPM(F) \Rightarrow \forall e' \in F, peso(e) < peso(e')$$

de donde:

$$e \neq ASPM(F') \Rightarrow \forall e' \in F, ASPM(F') < peso(e) < peso(e')$$

Por tanto, cuando ASPM($v_{f'}$) ya se conoce, F' no necesita ver su ASPM global en los nuevos nodos. Esta propiedad es muy importante ya que si F' tendría que observar su ASPM entre los nuevos nodos, podría ser muy tarde: Para el tiempo en que la absorción tiene lugar, $v_{f'}$ ya pudo haber reportado su propia ASPM y F' pudo haber decidido su ASPM global sin conocer los nodos de F. Como en este caso F' no se tiene que preocupar por ellos, el algoritmo continúa trabajando correctamente.

Para distinguir entre los dos casos, en el mensaje INICIAR se incluye un campo de información más: El ESTADO-NODO que puede tomar los valores BUSCANDO (en el primer caso) o ENCONTRO (en el segundo). Según este parámetro los nodos en el fragmento F pasan a estado BUSCANDO o ENCONTRO y solo enviarán los mensajes TESTs correspondientes a la búsqueda de sus ASPMs si están en estado BUSCANDO.

6.7 Análisis de ciertos detalles

Ya señalamos que el mantenimiento del balance en el tamaño de los fragmentos (con las operaciones de fusión y absorción asociadas) es la idea central que le permite al algoritmo GHS ser eficiente en términos del número de mensajes. Nos asegura que cada vez que se construye un nuevo fragmento y éste inicia la ejecución del algoritmo, su tamaño es al menos el doble que el de sus fragmentos componentes. Así los mensajes que se envían durante las dos etapas, y que de hecho viajan por todo el fragmento, no se están enviando cada vez que se agregue uno o pocos nodos. Sin embargo, la utilidad de ello está sujeta a que se mande un número de mensajes proporcional al tamaño del fragmento. De nada serviría controlar el crecimiento de los fragmentos si el número de mensajes que viajan en cada uno de ellos no es del orden de sus tamaños. Por tanto, es evidente la necesidad de seguir protocolos eficientes para el intercambio de información entre los nodos internos de un fragmento y entre los nodos de distintos fragmentos.

Hemos querido exponer en forma simple y clara el algoritmo GHS para que se haga patente la forma inteligente con que los autores han resuelto cada uno de los pasos necesarios para que se ejecuten eficientemente, además de controlar los posibles conflictos que hubieran podido presentarse dada la naturaleza asíncrona del algoritmo. A pesar de ello consideramos útil señalar explícita, aunque brevemente, algunos detalles importantes.

- Se aprovecha la existencia de una arista (y de sus nodos extremos) distinguida para usarla como lugar de inicio del proceso y centralización de la información. De otra forma se necesitaría aplicar algún protocolo para hallar un líder del fragmento.

• Un nodo buscando su ASPM local no envía TEST al mismo tiempo a todas las aristas BASICAS que tenga en ese momento sino una por una de menor a mayor peso hasta que una le responda DIFERENTE. Esto evita mensajes y esperas innecesarios.

• Aun más importante es la idea de marcar las aristas incidentes a un nodo v. Si no señaláramos las aristas de las que ya se sabe que van a nodos en el mismo fragmento de v como RECHAZADAS, v tendría que revisar todas sus aristas cada vez que busque su ASPM.

• La forma convergente en la que se envían las aristas de menor peso asegura que cada nodo v envía solo una arista hacia su padre. Enviar las ASPMs de todos los nodos del fragmento hasta la arista_memoria implicaría el reenvío de la arista elegida por cualquier nodo v a través de todas las aristas en el camino de v al nodo extremo correspondiente de la arista_memoria.

• El hecho de que cada nodo u que recibe un mensaje TEST de otro v de mayor nivel retarde su respuesta hasta tener un nivel mayor o igual que el de v asegura que los nodos (y por tanto los fragmentos a los que pertenecen) reciban solo respuesta de nodos (y por tanto de fragmentos) de mayor o igual nivel de manera que al elegir sus ASPMs locales (y al final la del fragmento) seguro que ésta irá a un nodo (fragmento) de mayor o igual nivel y se podrá aplicar una de las dos operaciones permitidas: Fusión o absorción. Pero además sirve para evitar respuestas no actualizadas que podrían producir errores.

Notemos que todas las anteriores ideas benefician al algoritmo también en términos del tiempo de su ejecución. Sin embargo, dadas las características del algoritmo, el tiempo no solo será consumido por el intercambio de mensajes sino también por las esperas ocacionadas por nodos comunicándose con otros de menor nivel.

Para terminar, recordemos que el propósito de esta sección es analizar brevemente los protocolos que se siguen para el intercambio de mensajes. Ideas de 'más alto nivel' como las operaciones de absorción y fusión y, las directamente asociadas a su implementación como las de identificador, nivel y manejo de apuntadores PADRE, han ocupado secciones separadas por razones de comprensión y en virtud del "refinamiento sucesivo" que nos hemos propuesto llevar a cabo.

6.8 Correctez y Complejidades

Teorema 6.8.1 *El algoritmo GHS es correcto*

Demostración [Esquema] Dar una demostración completa de la correctitud de este algoritmo no es una tarea sencilla. Usando la aproximación propuesta por Welch, se prueba la correctitud de una descripción de alto nivel del algoritmo (en este caso se hace al nivel de la gráfica de fragmentos). Habiendo hecho eso, se puede probar independientemente que el código sigue correctamente lo definido por la descripción de alto nivel ([WEL]).

Las dos aproximaciones a la correctez del algoritmo descritas en las secciones 6.2.1. y en la 6.3.1. son suficientes al nivel de gráfica de fragmentos. El siguiente paso de la demostración se puede formalizar implementando un algoritmo de alto nivel con un automata

de Entrada/Salida; el código de bajo nivel con otro automata de Entrada/Salida y luego mostrando que existe un mapeo entre las acciones del automata de bajo nivel a las del otro automata de tal manera que los comportamientos de ambos se corresponden [LYN] [3].

■

Lema 6.8.2 *Sea G=(V,E) una gráfica sobre la cual se aplica el algoritmo de GHS y $v \in V$ un nodo cualquiera. Entonces, v puede pertenecer a lo sumo a un fragmento de cualquier nivel l de los que se alcance durante el algoritmo.*

Demostración Sea v un nodo incluído en un fragmento F de nivel l.

La única manera de que v cambie de fragmento es que F pase a formar parte de uno más grande, llamémoslo F'. Si F' resulta de una fusión de F con otro fragmento de nivel l su nivel k, cumple que $k = l + 1$, si F es incluído en F' por una absorción, $k \geq l + 1$.

Se dice 'a lo sumo' justamente por esta última desigualdad pues las absorciones pueden implicar que un nodo de nivel l se 'salte' niveles y pase a formar parte de un fragmento de nivel k, con $k > l + 1$. ■

Teorema 6.8.3 *La complejidad de mensajes del algoritmo de GHS es $O(e + n \log n)$.*

Demostración Para la demostración analizaremos por separado cada uno de los dos términos:

$O(e)$ - Es consecuencia de que cada árista se observa al menos una vez.

En particular, la respuesta IGUAL a un mensaje TEST ocurre a lo sumo una vez por cada arista. Una vez que es enviada dicha respuesta por una arista, ésta es marcada como RECHAZADA y no se la vuelve a tomar en cuenta hasta el final del algoritmo.

$O(n \log n)$ - Ya habiendo analizado las parejas TEST-IGUAL, nos queda por ver que sucede con la información restante: TEST-DIFERENTE, INICIAR-REPORTE, CAMBIAR-RAIZ y CONECTAR. Como habíamos señalado anteriormente, el algoritmo utiliza protocolos que aseguran que el número de estos mensajes enviados dentro de un fragmento sea proporcional al tamaño del mismo. De hecho hay a lo sumo un mensaje de cada uno de ellos asociado a cada nodo v de un fragmento: v recibe un solo mensaje INICIAR y responde con un solo REPORTE, envía un solo mensaje TEST con respuesta DIFERENTE. El mensaje CONECTAR solo va por la arista elegida y CAMBIAR-RAIZ por los nodos en el camino de un nodo extremo de la arista_memoria hasta el extremo interno de la ASPM.

Lo anterior se cumple para los nodos de todos los fragmentos construídos a lo largo del algoritmo. Agrupemos los fragmentos por niveles, es decir los fragmentos de nivel 0, los de

[3] La segunda parte de la demostración da tema para todo un trabajo como en el caso de Welch. Nosotros hemos optado por no incluirla en mayor detalle en el presente trabajo dado que estamos más interesados en otros aspectos y características del algoritmo que en su implementación.

nivel 1, etc.; por el Lema 6.8.2 la suma de todos los nodos de los fragmentos de cualquier nivel l es a lo sumo n. Ya que en un fragmento con f nodos se envía $O(f)$ mensajes, el número total de mensajes que se envían en todos los fragmentos de un nivel cualquiera l es $O(n)$.

Falta ver, entonces, cual es el mayor nivel al que se puede llegar en el algoritmo:

- En el nivel 0 $\rightarrow$ cada fragmento tiene $1 = 2^0$ nodos.

- En el nivel 1 $\rightarrow$ cada fragmento tiene al menos $2 = 2^1$ nodos.

$\vdots$

- En el nivel l $\rightarrow$ cada framento tiene al menos 2^l nodos.

- En el último nivel, llamémoslo z, cada fragmento tiene al menos 2^z nodos. Sabemos que al final se tiene un solo fragmento que incluye todos los nodos, luego:

$$n = 2^z \Rightarrow z = \log n$$

Teniendo $O(n)$ mensajes enviados en un nivel y $O(\log n)$ niveles, se obtiene el segundo término $O(n \log n)$ ■

Teorema 6.8.4 *La Complejidad en tiempo del algoritmo de GHS es* $O(n \log n)$

Demostración Como bien señalan Gallager, Humblet y Spira, hay configuraciones tales que si inicialmente se despierta un solo nodo, los mensajes se envían casi secuencialmente. Si el tiempo es importante, es preferible despertar a todos los nodos (lo que requiere n-1 unidades de tiempo) antes de empezar la ejecución del algoritmo propiamente dicho.

Demostraremos ahora que aun bajo la suposición de que "Todos los nodos son despertados previamente", el algoritmo requiere $O(n \log n)$ unidades de tiempo.

Inicialmente hay que notar que al tiempo n, todos los nodos están despiertos y enviarán un mensaje CONECTAR. Al tiempo $2n$, cada nodo está en el *nivel 1* luego de la difusión del mensaje INICIAR. En el peor caso se forma una cadena de ASPMs de manera que el nodo al inicio de la cadena debe esperar n unidades de tiempo hasta que se opere una absorción a través de su ASPM (ver figura 6.15).

Hecha esta observación, demostraremos por inducción sobre el nivel de un fragmento que toma a lo sumo $5(l)n - 3n$ unidades de tiempo que todos los nodos lleguen al nivel l.

BASE: l=1. Ya vimos que al tiempo $2n$ todos los nodos están en *nivel 1* y $2n = 5(1)n - 3n$

PASO INDUCTIVO: Asumamos que se cumple para l.
En el nivel l, cada nodo envía a lo sumo n mensajes TEST que serán respondidos antes del tiempo $5(l)n - n$: En el peor de los casos puede suceder que un nodo v sea vecino a todos los restantes y justamente la última de sus aristas incidentes (ordenadas de menor a mayor peso) sea su ASPM debido a que las otras van a nodos de su mismo fragmento aunque v aun no lo sabe. Por hipótesis sabemos que a lo sumo al tiempo $5(l)n - 3n$, todos los nodos ya están en el *nivel l* y por lo tanto la respuesta al último de los n mensajes TEST será recibida por v en

Figura 6.15: En este ejemplo v_4 y v_5 se fusionan y el INICIAR correspondiente se propaga a v_3, luego a v_2 y recién a v_1

tiempo a lo sumo:
$5(l)n - 3n + n(por\ los\ TESTs) + n(por\ los\ IGUAL\ y\ el\ ult.\ DIFERENTE) = 5(l)n - n.$

La propagación de REPORTE en el fragmento lleva a lo sumo n unidades de tiempo, al igual que el CAMBIAR-RAIZ y CONECTAR posteriores. Finalmente la difusión del mensaje INICIAR que propaga el nuevo nivel $l+1$ requiere también n unidades de tiempo.

Por tanto al tiempo:

$$5(l)n - n + 3n = 5(l)n + 2n = 5(l)n + 5n - 3n = 5(l+1)n - 3n$$

todos los nodos están a nivel l+1.

En el último nivel, solo se usan los mensajes TEST e IGUAL y como este nivel es a lo sumo $n \log n$:

El algoritmo termina en tiempo $5n \log n$

■

6.9 La Implementación

Describimos a continuación el algoritmo completo seguido por cada nodo y la información que requiere para su correcta ejecución. El algoritmo consiste de una lista de acciones que el nodo debe realizar en respuesta a cada tipo de mensaje que recibe.

Se asume que cada nodo tiene una cola donde guarda los mensajes que le llegan para procesarlos en orden de llegada.

Variables locales de un nodo

- EN – Estado del nodo. Tiene uno de tres valores posibles: DURMIENDO (Estado inicial antes de empezar a ejecutar el algoritmo), BUSCANDO (Mientras participa en la búsqueda de la ASPM de su fragmento) y ENCONTRO (Luego de enviar su REPORTE).

- AA – Lista de aristas adyacentes. AA(j) contiene el estado de la j-ésima arista adyacente y puede asumir los valores BASICA, MARCADA o RECHAZADA. Inicialmente AA(j)=BASICA para cualquier j.

- PESO – Arreglo de pesos de las aristas adyacentes.

- IDF – Identidad fragmento (arista_memoria).

- NIV – Nivel.

- apuntador_PADRE – Arista que lo une con su nodo padre en el fragmento.

- Mejor_peso – Va guardando el peso de la arista de menor peso del subárbol del cual el nodo es raiz.

- Mejor_arista – Arista por la cual recibió el menor peso. Una vez hallada la ASPM de un fragmento, estas aristas forman el camino entre el extremo de la arista_memoria correspondiente y el extremo externo de la ASPM encontrada. A través de las mejoras aristas se enviará CAMBIAR-RAIZ hasta el extremo interior de la ASPM que a su vez tendrá como Mejor_arista justamente la ASPM y por tanto enviará por ella el CONECTAR (Revisar figura 6.13).

- Arista_obs – Arista sobre la que se va a enviar TEST en ese momento.

- Contador_busq – Cuenta los hijos de un nodo a los cuales envía INICIAR.

- Cola de mensajes – Contiene los mensajes que recibe el nodo. En general, al tomar un mensaje de esta cola, se proceso todo lo referente a él antes de analizar el siguiente. Sin embargo, en algunos casos especiales la respuesta implica el poner el mensaje nuevamente al final de la cola, retrasando su tratamiento.

Información que incluye cada mensaje

- INICIAR(nivel, peso(arista_memoria), estado_nodo).
Se envía simplemente el peso de la arista_memoria porque es suficiente para reconocerla unívocamente ya que los pesos son todos diferentes.

- REPORTE(peso).

- TEST(arista_memoria,nivel)

- IGUAL, DIFERENTE sin parámetros.

- CAMBIAR-RAIZ sin información adicional.

- CONECTAR(nivel)

EL ALGORITMO[4]

(1) Respuesta al despertado instantáneo

Ejecutar procedimiento Despertarse.

⟶ *Cuando un nodo en estado DORMIDO es despertado desde el exterior*

[4]Es el algoritmo del artículo original comentado; con los nombres de las variables y el pseudo-código en español.

(2) Procedimiento Despertarse

Sea m la arista adyacente de peso mínimo

```
begin
    AA(m) = MARCADA;
    ⟶ Elige su ASPM inmediatamente porque si estaba DORMIDO
      es un fragmento de un solo nodo que recien comienza a ejecutar el algoritmo.
    NIV = 0;
    EN = ENCONTRO;
    Contador_busq = 0
    ⟶ No tiene hijos
    send CONECTAR(0) sobre arista m
end
```

(3) Respuesta al recibir CONECTAR(L) sobre la arista j

```
begin
    if EN=DURMIENDO then Despertarse
    if L<NIV then begin
            AA(j) = MARCADA
            ⟶ Incluye arista en su fragmento
            send INICIAR(NIV,IDF,EN) sobre j
            ⟶ Envia INICIAR inmediatamente incluyendo su propio estado.
              Si EN=BUSCANDO los nodos del fragmento absorbido deberan buscar tambien
              Si EN=ENCONTRO los nodos del fragmento ya no buscan
            if EN=ENCONTRO then
              Contador-busq=Contador-busq+1
              ⟶ Debe buscar su ASPM en el fragmento tambien: Espera la respuesta de
                un hijo mas, la raiz del subarbol absorbido
            end
    else
    ⟶ Su nivel es igual del que recibe (menor no puede ser)
            if AA(j) = BASICA
              ⟶ No envio CONECTAR por ella: No hay acuerdo
              then
                  Poner mensaje recibido al final de la cola
                  ⟶ En espera de un acuerdo posterior o una absorcion
              else
                  send INICIAR(NIV+1,peso(j),BUSCANDO)
                  ⟶ Si esta MARCADA, hay acuerdo.
                    El estado luego de una fusion es siempre BUSCANDO
end
```

(4) Respuesta al recibir INICIAR(L,F,S) sobre la arista j

```
begin
      NIV=L; IDF=F; EN=S
      ⟶ Actualiza sus valores segun el fragmento al que ahora pertenece
      Apuntador_PADRE = j
      ⟶ A traves de j llegara a su padre que es justamente el nodo que le envia el INICIAR
      mejor_arista = nil; mejor_peso = ∞
      Para Todo i ≠ tal que AA(i) = MARCADA do begin
                        send INICIAR(L,F,S) sobre i
                        ⟶ Reenvia INICIAR por sus aristas incidentes que esten en el arbol
                        if S = BUSCANDO then
                          contador_busq = contador_busq +1
                          ⟶ Cada INICIAR que envia, incrementa el contador para
                              saber de cuantos hijos debe esperar REPORTE
      end
      if S = BUSCANDO then
        ejecutar procedimiento Test
        ⟶ LUEGO de enviar INICIAR a sus hijos, busca su propia ASPM
end
```

(5) Procedimiento Test

```
begin
      if Existen aristas adyacentes BASICAs then begin
        arista_observada = arista_adyacente de menor peso BASICA
        send TEST(NIV,IDF) sobre arista_observada
        end
      else begin
          arista_observada = nil
          ⟶ Existen aristas BASICAs
          ejecutar procedimiento Reporte
          end
end
```

(6) Respuesta al recibir TEST(L,F) sobre la arista j

```
begin
      if EN = DURMIENDO then
        ejecutar procedimiento Despertarse
```

```
      ⟶ Puede ser el primer mensaje que reciba
    if L > NIV
      then Poner mensaje recibido al final de la cola
      ⟶ Retrasa su respuesta pues es de menor nivel.
         Mientras no tenga un nivel mayor o igual a L, cada vez
         que tome este mensaje de la cola, lo mandara para atras.
         Notar que no lo procesa inmediatamente que su nivel suba
         a uno mayor o igual a L
      else
         if IDF ≠ F then
           send DIFERENTE sobre j
         else begin
              if AA(j) = BASICA then
                AA(j) = RECHAZADA
                ⟶ Al recibir TEST de otro nodo, conoce el fragmento del
                   mismo y por tanto sin necesidad de enviar
                   TEST por j, puede ya saber que la debe rechazar.
              if arista_observada ≠ j
                then send IGUAL sobre j
                ⟶ debe avisar que la rechaza.
                else ejecutar procedimiento Test
                ⟶ Cruce de mensajes TEST. Recibio TEST por la
                   misma arista que envio. Entonces, si tienen igual
                   identificador, la rechaza automaticamente sin avisar
                   (pues el otro nodo tambien detecta el cruce)
                   y busca la siguiente arista BASICA
         ⟶ En este caso (IDF=F), el TEST actua como IGUAL
      end
end
```

(7) Respuesta al recibir DIFERENTE sobre la arista j

```
begin
    arista_observada = nil
    ⟶ ya le respondieron y no necesita buscar mas
    if peso < mejor_peso then begin
      ⟶ Puede ser que antes de saber el peso de su propia ASPM
         haya recibido respuestas de algunos (talvez todos)
         de sus hijos que han modificado ya el valor de mejor_peso
      mejor_arista = j
      ⟶ Hasta el momento la arista de menor peso encontrada es su ASPM
```

```
        mejor_peso = peso(j)
        ⟶ Si su peso es menor que el encontrado hasta el momento,
            actualiza la variables correspondientes
    end
    ejecutar procedimiento Reporte
end
```

(8) Respuesta al recibir IGUAL sobre la arista j

```
begin
    if AA(j) = BASICA then
      AA(j) = RECHAZADA
      ⟶ Si envio un TEST fue porque j era BASICA en ese momento
          pero puede ser que antes de recibir el IGUAL haya
          recibido un CONECTAR por ella (Ver en el paso (11)) que este
          mensaje se envia por una arista sin revisar que se recibio por ella
    ejecutar procedimiento Test
    ⟶ Para buscar la siguiente BASICA
end
```

(9) Procedimiento Reporte

```
begin
    if contador_busq = 0 y arista_observada=nil then begin
      ⟶ Si ya recibio REPORTE de todos sus hijos y ya eligio su ASPM
      EN = ENCONTRO
      send REPORTE(mejor_peso) sobre Apuntador_PADRE
    end
end
```

(10) Respuesta al recibir REPORTE(P) sobre la arista j

```
begin
    if j ≠ Apuntador_PADRE then begin
      contador_busq = contador_busq -1
      ⟶ Un hijo menos de quien esperar respuesta
      if P < mejor_peso then begin
        mejor_peso = P
        mejor_arista = j
        ⟶ Cuidado! En mejor_arista no pone la arista de menor peso
            de todo el subarbol del cual es raiz, sino su
```

```
            arista adyacente por la que recibe el peso menor
            (La arista de menor peso esta en el subarbol cuya raiz
            es el nodo al otro extremo de j).
        end
      ejecutar procedimiento Reporte
      end
    else
        ⟶ El unico caso en que j = Apuntador_PADRE es cuando
          es un nodo extremo de la arista_memoria y recibe REPORTE
          del otro extremo.
        if EN=BUSCANDO then
          Poner mensaje recibido al final de la cola
          ⟶ Aun no sabe su ASPM, entonces no puede compararla con la del otro
        else
            if P > mejor_peso then
              ejecutar procedimiento Cambiar_raiz
              ⟶ Si la ASPM esta en su subarbol el se debe encargar
              de avisarle que fue elegida
            else if p=mejor_peso= ∞
                then HALT
                ⟶ Ya no hay aristas de salida
end
```

(11) Procedimiento Cambiar_Raiz

```
begin
    if AA(mejor_arista) = MARCADA then
      send CAMBIAR_RAIZ sobre mejor arista
      ⟶ Si mejor_arista esta MARCADA, forma parte del
        subarbol y por tanto por ella recibio el menor peso.
        Sin embargo no es la ASPM. Es una arista del camino
        al nodo extremo de la ASPM y por eso reenvia Cambiar_raiz por ella
    else begin
        ⟶ Es la ASPM
        send CONECTAR(NIV) por mejor_arista
        AA(mejor_arista) = MARCADA
    end
end
```

(12) Respuesta al recibir CAMBIAR_RAIZ

```
begin
```

```
        ejecutar procedimiento Cambiar_raiz
end
```

CAPITULO 7

ESTUDIO DEL COMPORTAMIENTO DEL ALGORITMO GHS

En este capítulo analizamos como se comporta el algoritmo GHS sobre el modelo estándar que describimos en el capítulo 2 y sobre otros menos generales.

Nuestro propósito es analizar los factores que influyen su desempeño al ejecutarse sobre redes que cumplen el modelo general y ver como se adapta a modelos más restringidos.

Presentamos algunos resultados con sus respectivas demostraciones formales y ejemplos aclaratorios cuyo desarrollo usa, en gran medida, la "digráfica de fragmentos" que ya conocemos.

7.1 GHS sobre el modelo general

Con el análisis del algoritmo de GHS realizado en el anterior capítulo obtuvimos una medida sobre su complejidad en el peor caso. $O(n \log n)$ es una cota superior del número de mensajes y del tiempo que consume cualquier ejecución. Pusimos énfasis en que el número de mensajes enviado es proporcional al tamaño y al número de fragmentos - ambos parámetros controlados por el nivel -. Lo que queda por determinar, y que de hecho es el propósito de esta sección, es cuales son los parámetros de los cuales depende la cantidad, tamaño y por tanto, el nivel, de los fragmentos que se forman a lo largo de una ejecución específica del algoritmo.

Determinar el máximo nivel que se puede alcanzar está directamente relacionado al número de fusiones que se aplica en la ejecución pues son estas operaciones las que incrementan el nivel de los fragmentos. A su vez, los tamaños dependen, no solo de las fusiones, sino también de las absorciones que se apliquen pues en ambos casos se incrementa el número de nodos de

los fragmentos. En otras palabras, nuestro objetivo se reduce a encontrar los parámetros dinámicos y estáticos que se deben conocer para poder establecer, o al menos estimar, el número de fusiones y absorciones que se aplicarán y, en el caso de las segundas, el orden de su aplicación.

Las Fusiones

Nuestro primer paso es ver que sucede con las fusiones.

Teorema 7.1.1 *Existe al menos una gráfica tal que al ejecutar GHS sobre ella, el AGM de la misma se forma aplicando solo fusiones*

Demostración Para la demostración basta dar un ejemplo de una ejecución que solo aplica fusiones (ver figura 7.1).

■

El siguiente teorema que presentamos equivale a afirmar que hallar el AGM de una gráfica aplicando fusiones depende solamente de la topología de la misma y de los pesos de sus aristas, sin importar el orden en que hayan despertado los nodos ni la velocidad con la que hayan viajado los mensajes. Para su demostración requerimos algunos lemas previos que son interesantes por si mismos.

Lema 7.1.2 *Sea $n_1 = 2^k$ el número de fragmentos de nivel j sobre los cuales se aplica fusiones para construir n_2 fragmentos de nivel $j+1$, entonces $n_2 = 2^k/2 = 2^{k-1}$*

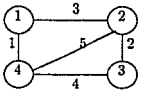
Demostración Tomemos la sub-gráfica de fragmentos $G_1 = (V_1, E_1)$ formada por los n_1 fragmentos de nivel j y sus ASPMs. Para que los n_1 fragmentos se fusionen , debe cumplirse que si la ASPM de un fragmento $f_i \in V_1$ es (f_i, f_j), la ASPM de f_j sea (f_j, f_i). Cada par de estas aristas que se cruzan en distintas direcciones equivalen a un acuerdo entre dos fragmentos de V_1 y por tanto dan lugar a un nuevo fragmento de nivel $j+1$. Por tanto $n_2 = |E_1|/2 = n_1/2 = 2^{k-1}$

■

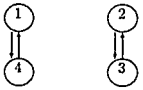
Lema 7.1.3 *Sea Λ una ejecución de GHS sobre una gráfica cualquiera $G = (V, E)$ y sea nf_l el número total de fusiones aplicadas a fragmentos de nivel l, entonces el número de fragmentos de nivel $l+1$ es $n_{l+1} = nf_l$*

Demostración $n_{l+1} \geq nf_l$ pues de cada fusión sobre dos fragmentos de nivel l, que posiblemente involucre la absorción de otros fragmentos, se crea un nuevo fragmento de nivel $l+1$. Si hay nf_l fusiones, se producen al menos nf_l fragmentos de nivel $l+1$.

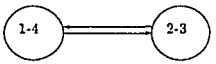
Por otra parte, la única manera de construir un fragmento de nivel $l+1$ es fusionando dos de nivel l. Por tanto $n_{l+1} = nf_l$ ■



Supongamos, para simplificar, que todos los nodos se despiertan al mismo tiempo.
Así, inicialmente cuando todos los nodos buscan su ASPM, la digráfica de fragmentos se ve como:



A su vez la digráfica de fragmentos resultante es:



Y el fragmento resultante final es:

Figura 7.1: Gráfica sobre la que se aplican solo fusiones

Lema 7.1.4 *Sea $G = (V, E)$ una gráfica tal que para cualquier ejecución Λ de GHS el máximo número de fusiones posibles entre fragmentos de nivel 0 es nf_0. Entonces, el máximo número de fusiones posibles de fragmentos de nivel l es $\lfloor nf_0/2^l \rfloor$.*

Demostración Haremos la demostración por inducción sobre el número de nivel l.

Por hipótesis, en el nivel 0 tenemos máximo $nf_0 = nf_0/2^0$ fusiones. Por el lema 7.1.3, habrá a lo sumo nf_0 fragmentos de nivel 1 sobre los cuales se puede aplicar a lo sumo $nf_0/2 = nf_0/2^1$ fusiones pues para cada una se necesita al menos dos fragmentos.

Supongamos, por hipótesis inductiva que el lema se cumple para el nivel l, es decir que el máximo número de fusiones posibles sobre fragmentos de nivel l es $\lfloor nf_0/2^l \rfloor$.

Veamos que sucede con el nivel $l+1$: Sea nf_l el número de fusiones aplicadas sobre fragmentos de nivel l, por hipótesis $nf_l \leq \lfloor nf_0/2^l \rfloor$ y por el lema 7.1.3 solo hay nf_l fragmentos de nivel $l+1$ sobre los cuales se puede aplicar a lo sumo $nf_l/2$ fusiones. Por tanto $nf_{l+1} \leq \frac{\lfloor nf_0/2^l \rfloor}{2}$ $= \lfloor \frac{nf_0}{2^{l+1}} \rfloor$ ∎

Lema 7.1.5 *Sea $G = (V, E)$ una gráfica tal que para cualquier ejecución Λ de GHS el máximo número de fusiones posibles entre fragmentos de nivel 0 es nf_0. Entonces, el máximo nivel al que un fragmento puede llegar en cualquier ejecución es $\lfloor \log nf_0 \rfloor + 1$.*

Demostración El nivel máximo es el nivel del último fragmento que se construye y que es el AGM completo, llamemos a este nivel MAX.

Se cumple que $MAX = l + 1$ con l el nivel de los dos fragmentos cuya fusión dio origen al AGM.

Aplicando el lema 7.1.4 y sabiendo que en el penúltimo nivel (l) se aplica una única fusión tenemos que:

$1 = \lfloor \frac{nf_0}{2^l} \rfloor$

$2^l = nf_0$

$l = \log(nf_0)$ ∎

Corolario 7.1.6 *Del lema 7.1.4 inferimos que si nf_0 es el número máximo de fusiones sobre fragmentos de nivel 0, entonces el número máximo de fusiones de cualquier nivel es : $\frac{nf_0}{2} + \frac{nf_0}{4} + \ldots + \frac{nf_0}{nf_0} = nf_0 \Sigma_{i=1}^{\log nf_0} \frac{1}{2^i} = nf_0 \times \frac{nf_0 - 1}{nf_0} = nf_0 - 1$.*

Lema 7.1.7 *Sean Λ' y Λ'' dos ejecuciones del algoritmo GHS sobre una gráfica G=(V,E) que aplican solo fusiones sobre todos los fragmentos de nivel menor a l. Entonces, en ambas se construyen en algún paso (posiblemente en distinto orden) los mismos fragmentos de nivel menor o igual a l.*

Demostración Demostraremos por reducción al absurdo.

Asumamos que Λ' y Λ'' son dos ejecuciones de GHS sobre G que aplican fusiones que no involucran absorciones sobre todos los fragmentos de nivel menor a l; y supongamos que construyen los mismos fragmentos de nivel q, $0 < q \leq l$ pero que al menos un fragmento f_{ab} de nivel q es construído por una ejecución y no por la otra. Sin pérdida de generalidad decimos que f_{ab} fue construído por alguna fusión aplicada en Λ'.

Por hipótesis, f_{ab} fue obtenido por la fusión (sin absorciones) de dos fragmentos f_a y f_b de nivel $q-1$. Para ello, se debe cumplir que las ASPMs de ambos fragmentos coincidan, es

decir:
$\text{ASPM}(f_a) = \text{ASPM}(f_b)$.

Supusimos que Λ' y Λ'' construyen los mismos fragmentos de niveles menores a q, en particular del nivel $q-1$. Por tanto f_a y f_b fueron construídos también en Λ''. Las ASPMs de los fragmentos son independientes de la ejecución pues todas las aristas tienen distinto peso y por tanto, en algún paso de Λ'' las ASPMs de f_a y f_b son encontradas posibilitando la fusión de ambos fragmentos y dando lugar al fragmento f_{ab} (pues asumimos que las fusiones sobre fragmentos de nivel menor a l no involucran absorciones). Esto contradice la suposición que f_{ab} no era construído por Λ''. ■

Teorema 7.1.8 *Sea G=(V,E) una gráfica tal que* $|V| = 2^x = n$ *y* Λ *una ejecución del algoritmo de GHS que utiliza solo fusiones al construir el AGM de G, entonces cualquier ejecución* $\Lambda \neq \Lambda'$ *de GHS sobre G también utiliza solo fusiones.*

Demostración Por hipótesis, Λ aplica solo fusiones hasta hallar el AGM de G. Esto es, a lo largo de Λ se aplican fusiones, que no involucran ninguna absorción, a todos los fragmentos de todos los niveles que ocurren en la ejecución.

Supongamos, por reducción al absurdo, que existe una ejecución $\Lambda' \neq \Lambda$ de GHS sobre G tal que aplica al menos una absorción. De todas las absorciones que se dieron en Λ, elegimos una que involucra la absorción de un fragmento f_w cuyo nivel r no sea mayor al de cualquiera de los otros fragmentos absorbidos durante toda la ejecución. Es decir, sobre los fragmentos cuyo nivel es menor que r se han aplicado solo fusiones. Sea P el paso de Λ' en el cual f_w es absorbido, en el momento de la aplicación de P se deben cumplir las siguientes condiciones :

a.1) Existen dos fragmentos f_x y f_y talque:

$$nivel(f_x) = nivel(f_y) = l$$
$$\text{y}$$
$$ASPM(f_x) = ASPM(f_y)$$

a.2)

$$nivel(f_w) = r \leq l$$
$$ASPM(f_w) = (w, xy) \text{ con } w \in f_w \text{ y } xy \in f_x \text{ (o bien } xy \in f_y)$$

Sin pérdida de generalidad, supongamos $xy \in f_x$.

Como elegimos f_w de tal forma que no existen fragmentos absorbidos de nivel menor a r; por el lema 7.1.7, Λ y Λ' construyeron, aunque en distintos paso y orden, los mismos fragmentos de niveles 0,1,2...,r y por tanto f_w es construido también por Λ.

Por hipótesis Λ aplica solo fusiones a los fragmentos de todos los niveles, en particular fusiona f_w con un fragmento f_z. Para ello, al momento de aplicar el paso P' de la fusión de f_w y f_z se debe cumplir :

b.1)

$$ASPM(f_w) = ASPM(f_z) = (w', z), w' \in f_w, z \in f_z$$

y

$$nivel(f_w) = nivel(f_z)$$

La ASPM de f_w es independiente de la ejecución, por tanto:

$$(w, xy) = (w', z) \Rightarrow w = w' \; y \; xy = z$$

Nuevamente por el lema 7.1.7, f_z fue construida en Λ' y xy formaba parte de ella. Pero, al momento de aplicar el paso P, xy pertenecía a f_x y $nivel(f_x) \geq r = nivel(f_w)$. A partir de esto hay dos casos a considerar:

1) Si $nivel(f_x) = r = nivel(f_z)$ entonces $f_x = f_z$ pues un nodo cualquiera solo pertenece a un fragmento de un nivel dado.

Entonces, $ASPM(f_x) = (w, xy) = ASPM(f_w)$ y $f_w \neq f_y$ contradiciendo la suposición de que $ASPM(f_x) = ASPM(f_y) \neq ASPM(f_w)$.

2) Si $nivel(f_x) > r = nivel(f_z)$ entonces f_z debió fusionarse con otro fragmento de nivel r, digamos f'_z en un paso anterior a P en Λ'. Pero para que esa fusión hubiese tenido lugar, $ASPM(f_z) = ASPM(f'_z) \neq ASPM(f_w)$ y por tanto f_w y f_z no se hubieran podido fusionar en Λ contradiciendo nuestra suposición. ■

Este último teorema es de particular importancia pues, como se establece seguidamente, el peor caso $O(n \log n)$ del número de mensajes que envía el algoritmo se alcanza al aplicar solo fusiones. Intuitivamente podemos ver que se tienen $\log n$ niveles cuando los fragmentos crecen en cada etapa exactamente al doble de su tamaño anterior y en cada nivel existen n nodos lo que sucede justamente con solo fusiones pues con éstas los nodos no 'se saltan' niveles, pasando exactamente de un nivel l a uno $l + 1$.

Como primer paso, vamos a establecer formalmente que si se aplican solo fusiones al ejecutar GHS sobre una gráfica con n nodos, cada nodo debe pasar por todos los niveles que se formen durante la ejecución. Es decir, ningún nodo luego de pertenecer a un fragmento de nivel l, pasa inmediatamente a pertenecer a uno de nivel $l + k$, con $k > 1$, sin pasar previamente por fragmentos de nivel $l + 1, \ldots, l + (k - 1)$

Lema 7.1.9 *Sea $G = (V, E), |V| = n$ una gráfica cualquiera y Λ una ejecución que aplica solo fusiones sobre G para hallar su AGM. Entonces No.nodos(nivel j) = n, $\forall j \in 1, 2 \ldots nivelmaximo$.*

Demostración Supongamos, por reducción al absurdo, que existe una ejecución Λ de GHS sobre G tal que hasta el nivel l todos los nodos han formado parte de fragmentos de nivel $0, 1, \ldots, l - 1, l$ sucesivamente y que existe al menos un nodo v tal que no forma parte de ningún fragmento de nivel $l + 1, \ldots, l + k - 1$ *con* $k > 1$.

v forma parte de un fragmento f_v de nivel l y por hipótesis Λ solo aplica fusiones, entonces f_v tiene que fusionarse con otro fragmento f_x de nivel l y la fusión produce un fragmento f_{vx} de nivel $l+1$ que contiene todos los nodos de f_v y de f_x y que, por tanto, contiene a v. Llegamos a una contradicción porque habíamos supuesto que v no pertenecía a ningún fragmento de nivel $l + 1$. ■

En cuanto al nivel máximo que se puede conseguir aplicando solo fusiones, se deduce del Corolario 7.1.5 que éste es :
$\lfloor \log n f_0 \rfloor + 1 = \lfloor \log n/2 \rfloor + 1 = \lfloor \log n - \log 2 \rfloor + 1 = \lfloor \log n - 1 \rfloor + 1 = \lfloor \log n \rfloor$.

Como ya mencionamos, el máximo nivel alcanzado depende de las fusiones que se puedan aplicar, por tanto nos interesa estimar el número de fusiones que tengan lugar en cualquier ejecución de GHS sobre una red contando con la mínima información posible sobre la misma. El siguiente teorema nos da una cota superior simplemente conociendo la topología de la red y aun sin conocer los pesos de las aristas a priori, dato que normalmente es más variable que la topología.

Teorema 7.1.10 *Sea G=(V,E) una gráfica y Λ una ejecución cualquiera de GHS sobre ella, entonces las fusiones sobre el nivel 0 que aplique Λ se hacen sobre un conjunto independiente de aristas $E' \subset E$.*

Demostración La haremos por reducción al absurdo.

Supongamos que existe una ejecución Λ' sobre una gráfica G=(V,E) tal que las fusiones de fragmentos de nivel 0 se realizan sobre un conjunto E_{ni} de aristas no independientes entre si. Entonces en E_{ni} existe al menos un par de aristas que comparten un nodo. Supongamos, sin pérdida de generalidad, que son de la forma (u, v) y $(v, w), u \neq w$. Sin embargo, si por la arista (u, v) se realiza una fusión es porque las ASPMs de u y v coinciden y son justamente (u, v), por tanto la ASPM de v no puede coincidir con la de w pues $u \neq w$ contradiciendo nuestra suposición. ■

Este último lema nos permite aprovechar los muchos resultados y algoritmos desarrollados para resolver el conocido problema de calcular la cardinalidad de un conjunto independiente maximal de aristas de una gráfica ("Maximum Cardinality Matching Problem" o MCMP) [BUS] para calcular la máxima cantidad posible de fusiones sobre fragmentos de nivel 0 y, por consiguiente, el máximo número de fusiones de fragmentos de cualquier nivel usando el teorema 7.1.10.[1]

Las Absorciones

En cuanto a las absorciones, el siguiente teorema establece un resultado muy interesante: La aplicación de una absorción a través de una arista depende únicamente de la topología de la red y de los pesos asignados a las líneas de comunicación.

Teorema 7.1.11 *Sea G=(V,E) una gráfica sobre la que se aplica una ejecución Λ de GHS, y sea e_1 una arista sobre la que se realiza una absorción en tal ejecución; entonces en cualquier ejecución Λ' sobre G, se realizará una absorción a través de e_1.*

[1] El tema del MCMP está fuera de los alcances de este trabajo pero consideramos importante dejar sentada la relación.

Demostración Supongamos, por reducción al absurdo que existe una ejecución $\Lambda' \neq \Lambda$ en la que no se realiza una absorción a traves de e_1. Hay dos casos a considerar:

1) Que en Λ' no se realize tampoco una fusión a través de e_1 lo que es imposible pues el AGM es único y las aristas que lo forman son aquellas por las que se aplicó una fusión o una absorción.

2) Que en Λ' se aplique una fusión por e_1. Sea $e_1 = (u, v)$, por hipótesis en algún paso P de Λ el fragmento al que pertenece u (f_u) se absorbe al fragmento al que pertenece v (f_v) o viceversa.
Supongamos, sin pérdida de generalidad que f_u se absorbe a f_v. Para que esto pueda suceder, al momento de ejecutar P, se debe cumplir que:

a.1) $ASPM(f_u) = (u, v)$ y

a.2.1) Si $nivel(f_u) = nivel(f_v)$, $ASPM(f_v) \neq (u, v)$

o bien

a.2.2) $nivel(f_u) < nivel(f_v)$

Por otro lado, para que en Λ' se realiza una fusión a través de (u, v), al momento de ejecutar alguno de sus pasos, digamos P', se debe cumplir lo siguiente:

b.1) $ASPM(f_u) = (u, v) = ASPM(f_v)$

y

b.2) $nivel(f_u) = nivel(f_v)$

Claramente se ve que si se cumple b.1) no se puede cumplir a.2.2) y que b.1) y b.2) no permiten que a.2.1) sea cierto. ■

Ya que el AGM es único y que justamente las aristas que lo forman son aquellas por las que se realizó una fusión o bien una absorción en algún paso del algoritmo, del teorema 7.1.11 se puede deducir un corolario aun más poderoso que el teorema 7.1.8.

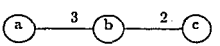
Corolario 7.1.12 *Sea G=(V,E) una gráfica sobre la que se aplica una ejecución Λ de GHS, y sea e_1 una arista sobre la que se realiza una fusión en tal ejecución; entonces en cualquier ejecución Λ' sobre G, se realizará una fusión a través de e_1.*

Damos a continuación un ejemplo sencillo para que esta característica se vea más claramente.

Ejemplo 1:

Veamos dos ejecuciones diferentes sobre esta gráfica.

Ejecución I



1. b despierta y envía un pedido de fusión a c.
2. c es despertado por el mensaje de b y responde fusionándose con el.
3. Luego a despierta y se absorbe al fragmento bc de nivel 1.

Ejecución II

1. a despierta, envía un pedido de fusión a b y queda esperando por la respuesta.
2. b es despertado por el mensaje de a y envía pedido de fusión a c.
3. c despierta y responde al pedido de b fusionándose con él y absorbiendo a a.

En ambas ejecuciones se realizaron la misma absorción y la misma fusión aunque no formaron exactamente los mismos fragmentos debido al distinto orden en que se aplicaron las operaciones. En la ejecución I fueron a,b,c,bc y abc; en la ejecución II no se formó bc.

La pregunta obvia que surge en este punto es cómo influye la aplicación de absorciones en el desempeño de GHS.

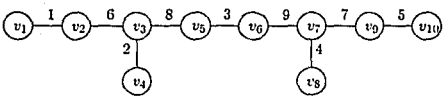
Vimos que con las fusiones los nodos van de un nivel al siguiente y eso es lo que hace que se tengan $\log n$ fases. Si $n = 2^x$, basta que exista una absorción para que no se llegue al nivel $\log n$ pues no hay suficientes fragmentos para aplicar las fusiones necesarias. Sin embargo, si $n \neq 2^x$, el máximo nivel al que se puede llegar es $\lfloor \log n \rfloor$ y se puede alcanzar aun cuando se absorban unos pocos nodos, más exactamente m si $n = 2^x + m$.

Ejemplo 2:

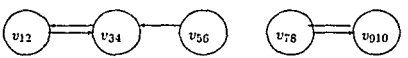
Sea G=(V,E) una gráfica con $|V| = 10 = 2^3 + 2$ con la topología mostrada en la figura 7.2

Notemos que en este ejemplo se tiene además el peor caso del número de nodos en cada fase. En cada una de ellas hay n=10 nodos debido a que la absorción se dio de un fragmento de nivel 1 a uno de nivel 2 y los nodos absorbidos no se saltó ningún nivel, si hubiera despertado después que el resto de los nodos, si hubiera saltado algunos.

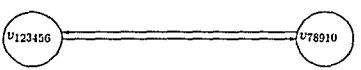
Esta observación ilustra un aspecto importante de las absorciones. Si bien su aplicación no depende de una ejecución en particular (por el teorema 7.1.11), el orden en que se aplican puede variar e influir en la cantidad de nodos en cada fase.



Supongamos, para simplificar, que se realiza una ejecución síncrona sobre G donde además todos los nodos se despiertan simultáneamente. Los fragmentos que se forman a lo largo de la ejecución son los siguientes:



Fase 1



Fase 2

Fase 3

Figura 7.2: Ejecución del ejemplo 2

Ahora bien, determinamos que el peor desempeño se da tanto aplicando solo fusiones en una red con un número de nodos igual a una potencia de dos o aplicando todas las fusiones posibles y solo absorciones sobre una proporción pequeña de nodos (m si $n = 2^z + m$). Lo que nos resta ver es si la aplicación de una cantidad más significativa de absorciones ayuda a reducir el número de niveles o bien el número de nodos por nivel.

Reducir el número de niveles está directamente relacionado a reducir el número de fusiones aplicadas. Al menos un par de nodos debe ponerse de acuerdo [2] y por lo tanto al menos una fusión tiene lugar. Veamos un ejemplo de mejor caso con una sola fusión (ver figura 7.3).

Dicho ejemplo se puede generalizar a una gráfica con cualquier topología y número de nodos. Solo se requiere que los pesos estén repartidos de tal manera que exista un par de nodos u y v tal que,

[2] Recordemos que la digráfica tiene al menos un ciclo.

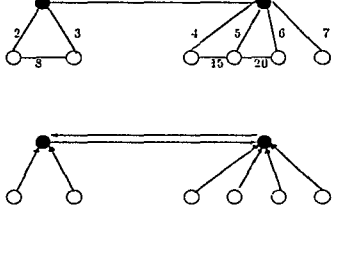


Figura 7.3: Ejemplo 3: Una gráfica donde se aplica una sola fusión. Mejor Caso

$$peso(u,v) = \min_{e \in E}(peso(e)) \text{ y}$$

$\forall x \neq u(v) \in V$

$$peso(x,u) = \min_{y \in V}(peso(x,y)) \text{ o bien } peso(x,v) = \min_{y \in V}(peso(x,y))$$

Es decir, las ASPMs de todos los nodos restantes apuntan a u o a v. Así no se podrán poner de acuerdo entre ellos y esperarán que u y v formen un fragmento de nivel 1 que los absorba.

La otra posibilidad sería reducir el número de nodos en algunos niveles al ser absorbidos por fragmentos de mayor nivel; pero en ese caso estaríamos dejando de usar esos nodos en fusiones. Si son más que los m definidos anteriormente ($n = 2^z + m$), el nivel máximo no podría ser $\lfloor \log n \rfloor$.

7.2 GHS en Modelos Menos Generales

Otro aspecto que resulta interesante analizar es como actua el algoritmo GHS sobre modelos menos generales que el presentado. En particular nos interesa ver su desempeño en dos variantes del modelo original cuyas características parecerían, en principio, ayudar a reducir la complejidad en número de mensajes e incluso en tiempo.

7.2.1 Topologías Sencillas

Inicialmente mantenemos todas las características del modelo general del capítulo 2 salvo la de tener cualquier topología. Nos limitaremos a gráficas que sean árboles o anillos.

GHS sobre árboles

La elección de esta topología no solo se debe a su simplicidad sino a que un árbol es en si el AGM y por tanto se esperaría que el algoritmo envíe una cantidad significativamente menor de mensajes al ejecutarse sobre ellos. Sin embargo esta situación no es aprovechada por el algoritmo GHS, no descubre que de inicio ya se tiene el AGM y sigue enviando $O(n \log n)$ mensajes. Esto se puede ver en el ejemplo que presentamos a continuación.

Ejemplo 4

Tomaremos una gráfica en forma de cadena (el árbol más simple) de, inicialmente, ocho nodos. Para simplificar su seguimiento supondremos una ejecución síncrona donde todos los procesadores despiertan y empiezan a ejecutar el algoritmo simultáneamente.

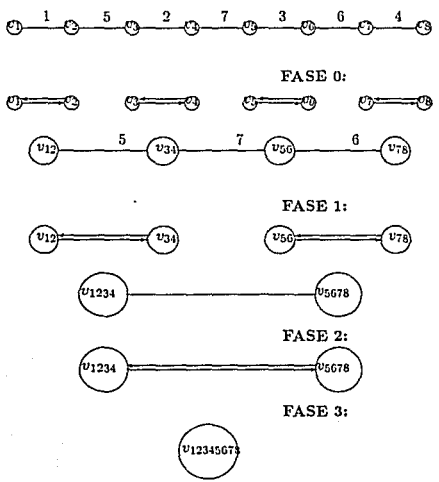


Figura 7.4: GHS sobre una cadena

Podemos notar que el ejemplo anterior (Figura 7.4) está incluído dentro de los peores casos que habíamos señalado anteriormente: La aplicación de solo fusiones a lo largo de toda la

ejecución. Por el lema 7.1.8 sabemos que cualquier ejecución aplica solo fusiones ya que esto depende únicamente de la topología de la red y de como estén repartidos los pesos en las aristas.

Tomando en cuenta esta observación, el ejemplo es fácilmente generalizable a cualquier cadena con $n = 2^z$ nodos donde z es un entero cualquiera, fijándonos cuidadosamente en las ubicaciones de los pesos. Si consideramos las aristas ordenadas según su posición de izquierda a derecha, la idea es poner primero los $n/2$ pesos menores una arista si, una no, es decir en la primera, tercera, quinta, etc. (los nodos con subíndice impar con la arista de menor peso a la derecha y los con subíndice par a la derecha). Luego, se aplica el mismo razonamiento en las aristas que aun no tienen peso asignado ignorando las aristas que ya lo tienen, es decir se pondrán los siguientes menores pesos a las aristas primera, tercera, quinta, etc. del conjunto de aristas que aun no tienen asignado un peso. El proceso se sigue hasta que todas las aristas tengan un peso asociado.

Observando la figura 7.4 notamos que este intercalamiento de pesos asegura que cada par de nodos compartan la misma ASPM , en la siguiente etapa sucede lo mismo con cada par de fragmentos de dos nodos y así sucesivamente hasta tener dos fragmentos de $n/2$ nodos cuyas ASPMs coinciden.

GHS sobre anillos

Una generalización natural de las cadenas analizadas en el anterior punto es agregarles una arista que vaya del primer al último nodo convirtiéndola en un anillo. El procedimiento de asignación de pesos es exactamente igual que antes sobre las aristas que formaban la cadena; a la arista adicionada se le pone un peso mayor que a todas las restantes para que seguro no se la incluya en el AGM y, por tanto, no se aplique ninguna fusión ni absorción a través de ella.

Ejemplo 5:

Presentamos un ejemplo de anillo donde, a partir de la fase 0, se siguen los mismos pasos que en la ejecución del ejemplo 4.

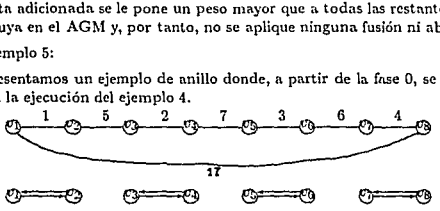


Figura 7.5: Ejemplo 5: Generalización del ejemplo 4 a anillos

7.2.2 Red Síncrona: Reduce la Complejidad de tiempo?

Nuestro modelo original incluye comunicación asíncrona. El desempeño de varios algoritmos asíncronos se beneficia de la sincronía que brinda el tener un reloj global en una red. Es este

el caso del algoritmo GHS?

Con el lema 7.1.8 demostramos que si una ejecución de GHS sobre una gráfica específica aplica solo fusiones, entonces cualquier ejecución hace lo propio. Por tanto, cualquier ejecución, síncrona o no, sobre la cadena del ejemplo 4 requería el envío de $O(n \log n)$ mensajes. Sin embargo, es muy probable que el tiempo si se modifique, en particular es importante el orden en que se despierten los nodos. A continuación damos un ejemplo de una ejecución sobre la cadena del ejemplo 4 (figura 7.4) donde se llega al peor caso de tiempo $O(n \log n)$ a pesar de ser una topología muy sencilla y de que la ejecución es síncrona.

Ejemplo 6:

Supongamos que sobre la gráfica del ejemplo 4 se aplica una ejecución de GHS donde el único procesador que despierta inicialmente es v_1.

Para simplificar el análisis hemos agrupado los pasos, que podrían considerarse como atómicos en una ejecución, en dos"acciones" más significativas que definimos a continuación y cuyos nombres son lo suficientemente explícitos como para evitar grandes explicaciones.

1) Construir fragmento: Envío de mensajes para que los fragmentos que lo vayan a componer se pongan de acuerdo (absorciones y fusiones)

2) Despertar fragmento: El primer mensaje que recibe un fragmento y ocaciona que este empieze a tomar parte en el algoritmo

Podemos ahora describir la ejecución básicamente en términos de estas dos acciones :

- El nodo v_1 se despierta y despierta (Enviando CONECTAR) a v_2
- v_2 se despierta y se construye el fragmento v_{12} de dos nodos
- v_{12} despierta a v_3 (enviando un TEST)
- v_3 despierta a v_4 (enviando CONECTAR)
- v_4 se despierta y se construye el fragmento v_{34} de dos nodos
- v_3 responde al TEST de v_{12} y se construye el fragmento v_{1234} de cuatro nodos
- v_{1234} despierta a v_5 (enviando un TEST)
- v_5 despierta a v_6 (enviando CONECTAR)
- v_6 se despierta y se construye el fragmento v_{56} de dos nodos
- v_{56} despierta a v_7 (enviando un TEST)
- v_7 despierta a v_8 (enviando CONECTAR)
- v_8 se despierta y se construye el fragmento v_{78} de dos nodos
- v_7 responde al TEST de v_{56} y se construye el fragmento v_{5678} de cuatro nodos. - v_5 responde al TEST de v_{1234} y se construye el fragmento $v_{12345678}$ que contiene a los ocho nodos de la gráfica.

Esquemáticamente, se pueden agrupar estas acciones que se ejecutan una tras de la otra (ninguna se realiza paralelamente a otra) de la siguiente forma:

1- Se construye fragmento de dos nodos.
2- Se construye fragmento de dos nodoc.

3- Se construye fragmento de cuatro nodos.
4- Se construye fragmento de dos nodos.
5- Se construye fragmento de dos nodos.
6- Se construye fragmento de cuatro nodos.
7- Se construye fragmento de ocho nodos.

Este esquema puede fácilmente generalizarse a una ejecución de las mismas características pero con 16, 32,...2^z nodos. Así como se requieren la construcción de dos fragmentos de tamaño dos (paso 1 y 2 o paso 4 y 5) para inmediatamente después formar uno de cuatro nodos (paso 3 o paso 6), o dos fragmentos de cuatro nodos para seguidamente formar uno de ocho; se requerirán dos fragmentos de ocho nodos (es decir dos secuencias de acciones como las que se ven en los pasos del 1 al 7) para formar un fragmento de tamaño 16 y asi sucesivamente.

Como la construcción de los fragmentos se realiza en forma secuencial, el tiempo que tarda esta ejecución en hallar el AGM es la suma del tiempo que consume cada uno de los fragmentos construídos a lo largo de la ejecución. Si agrupamos lo que tardan los fragmentos de nivel 2, luego los de nivel 4, etc. podemos expresar el tiempo total de la ejecución como:

Tiempo total ejecución =

$\Sigma_{niveles:1..\log n} \Sigma_{fragmentos\ de\ nivel\ i}$ *tiempo para hallar ASPM en frag. de nivel i*

Dado que la ejecución aplica solo fusiones, sabemos que un fragmento de nivel i tiene 2^i nodos y que hay $\frac{n}{2^i}$ fragmentos de nivel i; entonces podemos calcular fácilmente el tiempo total:

Tiempo total ejecución =

Número(fragmentos nivel 1) * tiempo(Hallar ASPM de un fragmento de nivel 1) + Número(fragmentos nivel 2) * tiempo(Hallar ASPM de un fragmento de nivel 2) + ... =

$n/2[(1+1/2)2+1]+n/4[(1+1/2)4+1]+n/8[(1+1/2)*+1]...+n/n[1*n] = n(1+1/2)+n/2+n(1+1/2)+n/4+...+n = (n+n+n+.....+n)+(n/2+n/2+n/2+...+n/2)+n/2+n/4+n/8+...+2 = n\log n + n/2\log n + \Sigma_{i=1}^{\log n} = (n+n/2)\log n + (n-1)$

Queda claro que el pobre desempeño de la ejecución se debe a que, al despertarse únicamente un nodo extremo de la cadena, la construcción de los fragmentos se realiza en forma secuencial.

CAPITULO 8

NUEVAS APROXIMACIONES Y MEJORAS

Como hemos venido mencionando, se considera el algoritmo de GHS como un gran aporte en el tratamiento del problema de hallar árboles generadores mínimos y en el diseño de algoritmos distribuidos en general. En los diez años que ya han transcurrido desde su aparición no se han dado a conocer aproximaciones totalmente distintas y significativamente mejores al mencionado problema. Sin embargo han surgido propuestas que mejoran el algoritmo original bajando la complejidad del tiempo o bien modificándolo para que la presencia de fallas en las líneas de transmisión y/o en los nodos no impliquen la reconstrucción total del AGM.

En este capítulo presentamos tres de ellas. No pretendemos hacer un análisis exhaustivo de las mismas pero si explicar sus ideas básicas y las estructuras generales de los algoritmos correspondientes. Por otro lado, aclaramos que no nos detendremos en los detalles del algoritmo GHS que son usados en ellos por considerarlos ya suficientemente discutidos.

8.1 El Algoritmo de tiempo lineal de Awerbuch

8.1.1 El problema

Recordemos que una de las características primordiales de GHS, y que le permite ser tan eficiente es la de unir fragmentos de cierto tamaño exclusivamente a fragmentos de mayor o igual tamaño. Esto requiere que, de alguna manera, se conozca o al menos se pueda acotar el número de nodos en un fragmento. Saber el número exacto de nodos en cada subárbol es muy difícil y contarlos consumiría un gran número de mensajes y unidades de tiempo. En lugar de ello se decide buscar maneras de obtener una estimación lo más exacta posible.

Una forma de hacerlo es la que se aplica en el algoritmo de GHS donde se aumenta el nivel de los fragmentos involucrados en una fusión. Así el nivel juega el papel de cota inferior del

logaritmo del tamaño del subárbol. Sin embargo puede suceder que el nivel de un fragmento esté muy por debajo del logaritmo de su tamaño. Por ejemplo cuando el fragmento ha aumentado de tamaño gracias a la absorción de muchos fragmentos de menor nivel pero relativamente grandes.

Un caso peor aún es cuando se tiene una cadena larga de subárboles de un mismo nivel l esperando cada uno fusionarse con el siguiente. La fusión se da solo entre los dos últimos fragmentos de la cadena (que son los que se ponen de acuerdo en su arista de salida de peso mínimo) y los restantes se absorben. Así todos los fragmentos de la cadena forman uno solo de nivel $(l+1)$ aún cuando la longitud de la cadena fuese muy grande. Cualquier fragmento F de nivel $k > l$ vecino a algún fragmento de la cadena, es decir que uno de los nodos de F -llamémoslo a- tenga una arista de salida adyacente a un nodo b de algún fragmento de la cadena (ver figura 8.1) y que esté buscando su ASPM deberá retrasar la selección de la misma hasta que todos los árboles a los cuales podría llegar su ASPM tengan nivel mayor o igual a k. En particular, debe esperar que el nivel del nodo b sea mayor a k. A su vez, b debe esperar que los dos últimos fragmentos de la cadena se fusionen y que le envíen su nuevo nivel. En el peor de los casos, b puede pertenecer al primer fragmento de la cadena y, a pesar de haber esperado tanto $(\Omega(No.\ arboles\ cadena * 2^l))$, el aumento puede no ser suficiente $(l+1$ sigue siendo menor a k) y F seguirá esperando por un fragmento que, muy probablemente, tiene un tamaño bastante mayor al suyo.

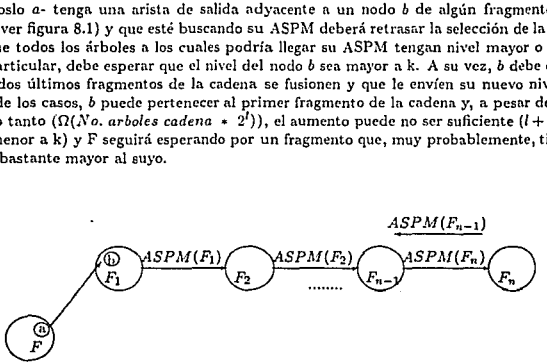


Figura 8.1: Cadenas de esperas

Este ejemplo ilustra una de las razones principales por la que la complejidad de tiempo del algoritmo de GHS no es lineal [1] : La espera que debe realizar un árbol debido a cada árbol vecino puede ser excesivamente grande ya que es posible que el nivel no refleje adecuadamente el tamaño real de los fragmentos y es muy probable que sean muchos los fragmentos que estén esperando por un mismo fragmento F muy grande que tarda mucho (tiempo proporcional a su tamaño) en incrementar su nivel. Adicionalmente, los vecinos que encuentren que su ASPM va a un nodo de F no subirán tantos niveles como para compensar el tiempo perdido.

[1] Recordemos que tiene complejidad de tiempo $O(n \log n)$.

Podemos usar un ejemplo que incluye Awerbuch en su trabajo para ver numéricamente que es lo que sucede: Consideremos que la cadena larga de esperas consiste de 2^{10} subárboles de nivel 10. Eventualmente el nivel del árbol resultante subirá a 11 pero esta actualización llevará tiempo lineal en el tamaño del árbol, es decir $\Omega(2^{20})$ [2]. Durante ese tiempo, un árbol T_{11} de nivel 11 vecino a un árbol T_{10} de nivel 10 que pertenece a la cadena está esperando que T_{10} alcance un nivel mayor o igual a 11. Una vez que la actualización de la cadena termina, T_{11} puede seguir buscando su arista de salida de peso mínimo. Desde el punto de vista de la búsqueda de esa arista en T_{11}, la construcción de la cadena de 2^{10} árboles de nivel 10 causa una pérdida de $O(2^{20})$ unidades de tiempo.

8.1.2 Los resultados

Awerbuch [AWE] emplea una técnica de actualización del tamaño de los fragmentos que le permite una estimación más exacta del mismo y así reduce el tiempo obteniendo a:

Complejidad de tiempo(Algo. Awerbuch) $= O(n)$

En cuanto al número de mensajes, se requieren algunos más que en GHS pero no aumenta su complejidad, es decir:

Complejidad de mensajes(Algo. Awerbuch) $= O(e + n \log n)$

8.1.3 La Idea Principal

Este algoritmo, en lugar de incrementar el nivel sólo al aplicar fusiones, lo va actualizando localmente estimando el número real de nodos en cada subárbol. Utiliza mecanismos que permiten detectar si el tamaño de un fragmento es demasiado grande para el nivel que tiene. En caso que así fuese, incrementa el nivel del fragmento.

Al asegurar que el nivel de un fragmento refleje acertadamente su tamaño, se previene que árboles pequeños esperen por árboles grandes.

8.1.4 Actualización del nivel de un fragmento

Para la estimación del número de nodos queda claro que el algoritmo requiere de cierta información que permita, a cada fragmento, actualizar su nivel periódicamente sin necesidad de estar contando todo el tiempo el número de nodos.

Utiliza dos nuevos mecanismos llamados Actualización de la Raiz (Root-Update o R-U) y Chequeo de Distancia (Test-distance o T-D):

[2] El tamaño del árbol final es del orden del número de nodos de cada subárbol por el número de subárboles que lo componen. Hay 2^{10} subárboles y al ser de nivel 10, cada uno tiene $\Omega(2^{10})$ nodos.

R-U incrementa el nivel de la 'raiz' [3] y de todo un fragmento F si después de cierto tiempo F aún no encontró su ASPM. La idea subyacente es que si la raiz detecta que el mensaje INICIAR ha tenido que avanzar una gran distancia al difundirse por F o si algún nodo se da cuenta que ya hay muchas aristas en el fragmento, entonces la propagación de ese mensaje INICIAR es interrumpida y se inicia la de uno nuevo con el nivel de la raiz incrementado.

T-D incrementa el nivel de los nodos de un subárbol F_1 de un fragmento F si la raiz de F_1 está suficientemente lejos de la raiz de F. El 'suficientemente lejos' se refiere a que si la raiz de F_1 está a cierta distancia de la de F, entonces se detecta que el tamaño de F es más grande de lo que se espera para un fragmento de su nivel actual.

El objetivo principal de usar estos dos procedimientos es lograr que el periodo de tiempo en el cual l es el menor nivel en la red esté acotado superiormente por $O(2^l)$.

8.1.5 El Algoritmo

Damos a continuación una descripción esquemática del algoritmo de Awerbuch a un nivel de detalle equivalente al que presenta el autor en su artículo aunque agregando aclaraciones que nos parecen pertinentes y resumiendo los pasos que son equivalentes en el algoritmo GHS.

El algoritmo consta de dos fases:

I- Fase de Conteo Esta fase cuenta el número de nodos de la red (n) y la única razón de su inclusión en el algoritmo es que la segunda fase, que es en realidad la que construye el AGM, necesita conocer n.

Construye un árbol generador cualquiera usando la idea de niveles de GHS, sin embargo los ajusta de acuerdo a la altura de los subárboles así como al grado y número de los nodos en ellos. Como no se construye un árbol generador mínimo, los árboles no esperan casi nunca unos por otros.

El algoritmo en sí es muy similar al de GHS: En cada etapa se tiene un bosque generador y los subárboles se van uniendo hasta formar un solo árbol final. Cada subárbol tiene un nivel asociado.

En el transcurso del algoritmo, cada nodo busca sus aristas incidentes de salida según el orden de sus pesos (de menor a mayor) hasta que encuentra una que vaya a un árbol de mayor o igual nivel. Si durante esta búsqueda, el nodo detecta una arista que va a un árbol de menor nivel, el árbol de mayor nivel inicia la 'invasión' del árbol pequeño a través de esa arista. La invasión consiste simplemente en que el árbol grande le pasa su nivel e identificador al pequeño[4].

[3] Awerbuch define la raiz de un fragmento como el nodo extremo de la arista_memoria que tiene mayor identificador.

[4] Puede suceder que un árbol sea invadido al mismo tiempo por varios árboles de mayor nivel a través de distintas aristas. En ese caso se permite que cada árbol invasor capture un pedazo disjunto del árbol más pequeño.

Paralelamente a las posibles invasiones realizadas por sus nodos, un árbol sigue buscando su ASPM global que vaya a un árbol de mayor o igual nivel. Una vez que la encuentra, intenta fusionarse a través de ella.

Para realizar su tarea, el algoritmo de conteo tiene tres procedimientos básicos:

1) Búsqueda de arista (Link-Search o L-S): El árbol se expande conquistando árboles vecinos de menor nivel hasta que se encuentra con un árbol de mayor o igual nivel. Si no lo encuentra el algoritmo termina. Si encuentra al menos uno, elige la arista de menor peso que vaya a otro árbol de igual nivel y se produce una fusión utilizando el procedimiento Matrimonio que describiremos posteriormente. [5]

2) Actualización de nivel (Level-Update o L-U): Actualiza el nivel del árbol resultante de una fusión, incrementándolo hasta el logaritmo de su cardinalidad. Puede ser invocado también durante la ejecución de L-S, cuando se detecta que el árbol tiene altura o grado demasiado grandes en relación a su nivel.

Tiene éxito a no ser que el árbol sea invadido al mismo tiempo por otro árbol de mayor nivel.

En caso de que tenga éxito, se llama al procedimiento L-S.

3) Matrimonio(M): Fusiona dos árboles del mismo nivel que tengan la misma arista de salida de peso mínimo. Cuando un árbol no puede ponerse de acuerdo con ningún otro, el algoritmo termina.

Estos procedimientos serán ejecutados en algún orden secuencial por los árboles que aún no hayan sido invadidos por otros hasta terminar con un solo árbol generador de toda la gráfica.

II- Fase del AGM Esta fase a su vez se divide en dos subfases o pasos que pasamos a describir a continuación.

II.a- Se ejecuta un algoritmo idéntico a GHS que termina cuando todos los árboles llegan al tamaño de $\Omega(\frac{n}{\log n})$. En el momento que un fragmento pasa a nivel $\log(\frac{n}{\log n})$ sabe que tiene al menos $\frac{n}{\log n}$ nodos y por tanto pasa a la fase 2. La idea es asegurar que los fragmentos que pasan de la fase 1 a la 2 son relativamente pequeños respecto al número total de nodos (n). Esta característica es indispensable para disminuir el tiempo de ejecución del algoritmo.

II.b- En cada fragmento F que esté ya en la segunda fase, se siguen los siguientes pasos:

i. El nodo raiz de F difunde el mensaje INICIAR igual que en GHS.

ii. Cada nodo que recibe INICIAR de la raiz de F busca su ASPM igual que en GHS.

iii. La información obtenida por cada nodo es enviada a la raiz en la misma forma convergente que en GHS.

[5] No se está construyendo un árbol de peso mínimo pero se requiere algún criterio de selección para decidir entre la distintas aristas de salida

iv. La raiz elige la ASPM global (v,w) ($v \in F, w \in G$) de F y envía Change-Root como lo hace también GHS.

v. El nodo v recibe Change-Root.

Si (v,w) es una arista por la cual se fusionarán F y otro fragmento G (la arista de salida de peso mínimo de ambos es (v,w)) e $identificador(v) > identificador(w)$, v será la raiz del árbol resultante de la fusión.

En otro caso, es decir si $identificador(w) > identificador(v)$ o bien si la arista (v,w) es una arista por la cual F se absorberá a G, v se 'cuelga' a w por esa arista, F se vuelve subárbol del árbol resultante y la raiz de ese subárbol es v. Cuando w envía un mensaje INICIAR a v (Ya sea porque F y G se fusionaron a través de (v,w) y por tanto w es la raiz del árbol resultante o bien porque F se absorbió a G y el árbol resultante tiene otra raiz que eventualmente le envía INICIAR a w quien a su vez, por ser su padre, se lo reenvía a v) y v se encarga de que se difunda por F para que sus nodos participen también en la elección de la ASPM de todo el árbol. Sin embargo aquí está la mayor diferencia respecto al algoritmo de GHS. En lugar de que v mande INICIAR a sus hijos inmediatamente después de recibirlo de w, antes de hacerlo v itera el procedimiento T-D. Si T-D tiene éxito entonces el tamaño del árbol completo es mayor que $2^{nivel(v)+1}$ y por tanto se debe incrementar el nivel de v y de todos los demás nodos de F en 1. Luego v vuelve a ejecutar T-D aumentando el nivel de los nodos de F en 1 hasta que T-D ya no tenga éxito.

El funcionamiento de T-D al ser invocado es el siguiente: El nodo v envía un *token de exploración* a su padre w. El token lleva un contador que se inicializa en $2^{nivel(v)+1}$. A la llegada del token a w, éste resta al contador el número de hijos que tiene y lo reenvía a su padre que hará lo mismo y así sucesivamente mientras el contador sea positivo y el nodo receptor no sea la raiz.
Eventualmente, a través de los nodos padre, o bien el token llega a la raiz con *contador* > 0 o bien se cumple que *contador* < 0 antes de llegar a la raiz.

En el primer caso el token 'muere' y T-D falla.
En el segundo se le envía un reconocimiento -Acknowlegment- a v. y cuando v lo recibe, difunde un mensaje especial por F para que cada uno de sus nodos incremente su nivel en 1. Luego de esta difusión, T-D se declara exitoso y se reinicia.
Es claro que cuando T-D falla, o sea cuando el token no llega a la raiz, v está a una distancia a lo sumo $2^{nivel(v)+1}$ de la misma. Recíprocamente T-D tendrá éxito para todo nodo a una distancia de la raiz mayor que $2^{nivel(v)+1}$.

Ya que cada nodo que recibe el token tiene grado al menos 2, el token puede viajar una longitud a lo sumo $2^{nivel(v)+1}$. Notemos que en este procedimiento no se toma en cuenta todos los nodos del árbol sino exclusivamente los hijos de los nodos en el camino de v a la raiz , se está haciendo una aproximación (Ver figura 8.2).

vi. Independientemente de lo anterior, cuando el mensaje INICIAR ha avanzado una distancia mayor a $2^{nivel(raiz)+1}$ o si alguno de los nodos del fragmento detecta más de

$2^{nivel(raiz)+1}$ aristas internas al buscar su candidato local; se activa el procedimiento R-U: Se interrumpe el proceso de selección, se incrementa el nivel de la raiz en uno y el nuevo nivel es difundido a través del nuevo mensaje INICIAR que se envía sobre el árbol para reiniciar el proceso de selección.

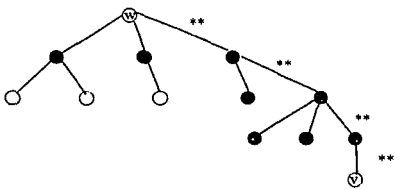


Figura 8.2: El token sube por las aristas marcadas con ** contando los nodos incluídos en ese camino y a los hijos de éstos (los llenos).

8.1.6 Correctez y Complejidades

En esta sección demostraremos la correctez y las complejidades de mensajes y tiempo del algoritmo completo de Awerbuch. Sin embargo sólo se darán los términos correspondientes a la fase de conteo sin incluir sus demostraciones porque en realidad nos interesa analizar la correctez y el desempeño de la segunda fase.

Teorema 8.1.1 *El algoritmo de Awerbuch es correcto.*

Demostración [Esquema] La fase de conteo es correcta y termina dando como resultado el número de nodos en la red.
La fase GHS eventualmente termina. Al final de la subfase II.a, los fragmentos resultantes son fragmentos del AGM de la gráfica total (por GHS).
La segunda subfase es muy similar al algoritmo GHS, únicamente mantiene los niveles más actualizados. R-U y T-D modifican solamente la asignación de niveles a los nodos, los fragmentos se siguen construyendo con la misma filosofía que en GHS. Termina con el AGM de la gráfica. ■

Teorema 8.1.2 La complejidad de mensajes *del algoritmo es* $O(e + n \log n)$

Demostración De la fase de conteo: $O(e + n \log n)$.

De la subfase II.a: $O(e + n \log n)$ por GHS.

De la subfase II.b: Comparando con GHS los únicos mensajes adicionales son los enviados por R-U y T-D.

R-U: Agrega el envío de mensajes INICIAR a los nodos de un fragmento un número constante de veces, tantas veces como se detecte que es necesario incrementar el nivel de la raiz y, consiguientemente, el nivel de todos los nodos.

T-D: Se adicionan los tokens de exploración y los reconocimientos.
Sea el token_exploración_final en un subárbol el token que llega a la raiz y sea el mensaje que lleva ese token el mensaje_exploración_final. El número total de mensajes de exploración no finales no puede exceder por más de una constante al número total de transmisiones de mensajes de exploración final. Concretamente, cada token viaja por un camino que es el doble de largo que su predecesor. La longitud de estos caminos forma una progresión geométrica cuya suma está dominada por el último término:
$2^{nivel(v)+1} + 2^{nivel(v)+2} + 2^{nivel(v)+3} + \ldots + 2^{nivel(v)+k} = 2^{nivel(v)} * (\Sigma_{i \leq k} 2^i) \leq 2^{nivel(v)+(k+1)}$ o sea el término que corresponde al mensaje_exploración_final. Por tanto es suficiente demostrar que el número de mensajes_exploración_final es $O(n \log n)$.

Construímos un árbol dirigido llamado 'Arbol de fusiones' AF=(EF,VF) donde:
VF=$\{v_a$ tal que existe un árbol a durante la fase 2 correspondiente a $v_a\}$ y
EF=$\{e = (v_{a_1}, v_{a_2})$ si a_1 resulta de fusionar a_2 con algun(os) otro(s) arbol(es)$\}$
Por tanto, las hojas de AF son los árboles iniciales de la fase II.b -los que resultan inmediatamente al terminar la fase II.a- y el padre de un vértice que representa el árbol a es el vértice correspondiente a un árbol que resulta de fusionar a con otros árboles.

Notemos que si un token de un árbol a pasa por un nodo i en la gráfica original, i no puede recibir token de ningún árbol a_1 antecesor de a en AF ya que cuando a manda su mensaje_exploración (o sea cuando recibe INICIAR de su padre), ya se volvió parte o subárbol de otro más grande, éste es antecesor suyo en AF y contiene a i (Ver figura 8.3). Por otro lado, un nodo recibe a lo sumo un mensaje_exploración_final de cada árbol en la subfase II.b pues un árbol envía uno solo de estos mensajes. De ahí se ve que el número total de mensaje_exploración_final es menor o igual que la máxima cardinalidad de un subconjunto de nodos en el AF tal que ningún nodo en el subconjunto sea antecesor de otro.
Obviamente cada hoja en el árbol tiene un único padre pero un padre puede tener más de una hoja (al menos una) por lo que el subconjunto más grande posible es el formado por las hojas que corresponden a los árboles iniciales de la subfase II.b. Ya que en esta subfase cada árbol tiene al menos $\frac{n}{\log n}$ nodos y los árboles en la fase II.b son disjuntos en nodos; el número de tales árboles es a lo sumo $\log n$. Por tanto el número total de mensaje_exploración_final recibidos por un nodo es menor o igual a $\log n$, es decir:

$$No.\ total\ mensaje\_exploracion\_final = O(n \log n)$$

Viendo las complejidades de comunicación de ambas fases concluimos que:

La Complejidad en mensajes del algoritmo Awerbuch es $O(e + n \log n)$

■

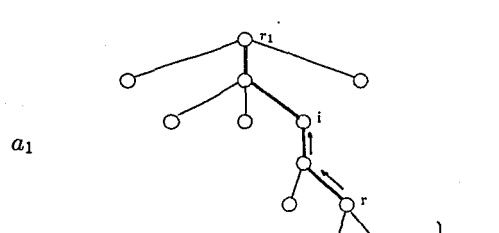


Figura 8.3: r raiz del árbol a y r_1 la de a_1

Para los siguientes dos lemas vamos a definir:
A_l= Primera vez (tiempo) en la que l es el menor nivel de la gráfica y
C_l= Primera vez (tiempo) luego de A_l en que cada nodo a una distancia menor a $2 * 2^{l+1}$ de la raiz del fragmento al cual pertenece tenga nivel $l + 1$ al menos.

Lema 8.1.3 $C_l - A_l = O(2^{l+1})$

Demostración Luego de A_l, los mensajes TEST enviados por un árbol de nivel l son respondidos inmediatamente pues el resto de los árboles tienen nivel mayor o igual a l (ver definición de A_l). Es fácil ver que, o bien el tiempo en la selección es $O(2^{l+1})$ o el nivel de la raiz se incrementa (pues se incrementa justamente si el INICIAR tomó más tiempo que ese en propagarse). En caso que se llegue a seleccionar la arista de menor peso, si ésta no es una arista memoria (por la cual se realizó una fusión), la raiz deja de serlo; si es una arista memoria, la raiz sigue siéndolo pero su nivel se ha incrementado en uno. En todos los casos, en el tiempo $A_l + O(2^{l+1})$ todas las raices tienen nivel mayor a $l + 1$.
Adicionalmente se requiere tiempo $O(2^{l+1})$ para propagar el aumento del nivel de las raices a todos los nodos internos cuyas distancias a sus raices no excede $2 * 2l + 1$. ■

Lema 8.1.4 $A_{l+1} - C_l = O(2^{l+1})$

Demostración Considere un nodo v de un subárbol T con profundidad D, $2^d \leq D < 2^{d+1}$ y que tiene nivel s con $s \geq d$ cuando se vuelve un subárbol.

Sea H la distancia de la raiz de T a la raiz r del árbol completo.

Si $s \geq l+1$, entonces $nivel(v)$ ya es $l+1$ al menos y obviamente en el tiempo C_l tiene nivel mayor o igual a $l+1$.

Si $s < l+1$ y $H < 2^{l+1}$ entonces v está a lo sumo a distancia $D + H \leq 2^s + 2^{l+1} \leq 2 * 2^{l+1}$ de r y su nivel es al menos $l+1$ en el tiempo C_l (por definición de C_l).

Por tanto solo queda ver lo que sucede si $s < l+1$ y $H > 2^{l+1}$. Como $s < l+1$, primero T debe volverse un subárbol antes del tiempo C_l. Luego cada procedimiento T-D de nivel q, $s < q \leq l+1$ termina exitosamente en tiempo $O(2^{q+1})$ luego de su invocación y el nivel de todos los nodos de T llega a q de donde el tiempo total consumido por los T-D $\leq \Sigma_{q=s+1}^{l+1} O(2^{q+1}) = O(2^{l+1})$. Entonces en el tiempo $C_l + O(2^{l+1})$ todos los nodos tienen nivel mayor o igual a $l+1$. ∎

Para el siguiente lema y el posterior teorema haremos dos definiciones:

Γ_l=longitud intervalo tiempo en el cual el menor nivel en la red es l.

S_l=tamaño del árbol más grande de nivel l.

Lema 8.1.5 $S_l \geq c * \Gamma_l$, *c una constante*

Demostración La demostración de este se halla en la referencia [GAF]. ∎

Teorema 8.1.6 *La complejidad de tiempo del algoritmo de Awerbuch es O(n)*

Demostración Fase de conteo: O(n). Nuevamente se excluye la demostración correspondiente a esta fase por las razones ya expuestas.

Fase del AGM:
Subfase II.a:

Sea α el conjunto de todos los niveles cuyo árbol más grande tiene un tamaño menor a $\frac{n}{\log n}$, es decir:

$$\alpha = \{nivel\ l\ tal\ que\ S_l < \tfrac{n}{\log n}\}$$

y sea β el conjunto de niveles cuyo árbol más grande tiene al menos $\frac{n}{\log n}$ nodos, es decir:

$$\beta = \{nivel\ l\ tal\ que\ S_l \geq \tfrac{n}{\log n}\}$$

Los intervalos de tiempo en los cuales el menor nivel en la red es 1,2,etc. son disjuntos. El nivel 2 empieza a ser el menor nivel en la red cuando 1 deja de serlo, el nivel 3 cuando el nivel 2 deja de serlo y así sucesivamente. Así el tiempo total que tarda la primera subfase es la suma de los intervalos de tiempo que se queda en cada nivel antes de pasar a la segunda fase.

Usando el lema 8.1.5, se puede acotar la complejidad de tiempo de la primera subfase con los tamaños máximos de árboles, pues ese dato si se puede estimar:

$$\Sigma_{l\in\alpha}\Gamma_l + \Sigma_{l\in\beta}\Gamma_l \leq 1/c(\Sigma_{l\in\alpha}S_l + \Sigma_{l\in\beta}S_l)$$

Ya que los árboles que alcanzaron un tamaño mayor a $\frac{n}{\log n}$ no participan más en la subfase II.a, para todo par de niveles en β, los árboles más grandes de ambos deben ser disjuntos en nodos [6] y por tanto la suma de todos los nodos de los árboles más grandes de todos los niveles del conjunto β es menor o igual al número total de nodos, es decir:

$$\Sigma_{l\in\beta}S_l \leq n \qquad (1)$$

Por otro lado todos los árboles de los niveles en el conjunto α tienen a lo sumo $\frac{n}{\log n}$ nodos. Dado que no puede existir un fragmento con nivel mayor al logaritmo del número de sus nodos, los niveles que pueden estar en α son 0, 1, 2,...$\log(\frac{n}{\log n})$, es decir:

$$\Sigma_{l\in\alpha}S_l \leq \Sigma_{l=0}^{\log n/\log\log n} n/\log n \leq n \qquad (2)$$

(1) y (2) implican que:

$$Tiempo(Subfase\ II.a) = O(n)$$

Subfase II.b: Para ver el tiempo que requiere esta fase, nuevamente hay que definir cada período Γ_l en el cual l es el menor nivel en la red y sumar estos sobre todos los niveles (tiempo en el que ya no hay árboles de nivel 1 + tiempo en el que ya no hay árboles de nivel 2 +.....+ tiempo en el que ya no hay árboles de nivel log(n)).

Los lemas 8.1.3 y 8.1.4 implican que:

$$\Gamma_l = A_{l+1} - A_l = (A_{l+1} - C_l) + (C_l - A_l) = O(2^{l+1})$$

de donde:

$$tiempo(Subfase\ II.b) = \Sigma_{l=0}^{\log n}\Gamma_l = O(2^{\log n}) = O(n)$$

■

8.1.7 Una nota interesante

Normalmente seguir la ejecución de un algoritmo distribuído sobre una red específica es una tarea difícil sobre todo cuando se supone una red asíncrona. El algoritmo de Awerbuch presenta dos dificultades adicionales:

1- Distintos fragmentos pueden estar corriendo, en un mismo instante, distintos algoritmos dependiendo de la subfase en la que se encuentren.

2- Distintos nodos de un mismo fragmento que está en la subfase II.b pueden estar corriendo distintos procedimientos, T-D o R-U, en un mismo momento. Esto sin mencionar las dificultades propias de cada uno de estos procedimientos como el incremento de nivel de los nodos a distintos tiempos y las posibles interrupciones del proceso de selección de la ASPM en R-U.

A pesar de ello deseamos analizar, de forma esquemática, la ejecución de Awerbuch sobre una gráfica que nos interesa en particular dado el análisis que hicimos en el anterior capítulo.

[6] Ya no se fusionarán entre ellos en la primera subfase

Awerbuch fundamenta las bondades de su algoritmo en la manera en que actualiza el nivel de los nodos para que aquel refleje más apropiadamente el tamaño de los fragmentos argumentando que incrementar el nivel sólo cuando se fusionan dos fragmentos es insuficiente ante la posibilidad de que una fusión implique la absorción de varios otros fragmentos. Sin embargo, en el anterior capítulo dimos un ejemplo que mostraba que se podía llegar al peor caso de tiempo del algoritmo GHS ($n \log n$) aplicando solo fusiones y por tanto manteniendo en todo momento el tamaño de los fragmentos de nivel l en exactamente 2^l nodos.

La duda que surgió inmediatamente fue como podía el algoritmo de Awerbuch disminuir el tiempo de ejecución en una gráfica como la del ejemplo 2 del capítulo 7.

Recordemos que las especificaciones básicas del ejemplo mencionado eran las siguientes:

i) La topología de la red es la de la figura 8.4

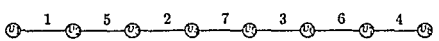


Figura 8.4: En este caso $n = 8$, $\log n = 3$, $\frac{n}{\log n} < 3$

ii) El procesador v_1 inicia solo el algoritmo. Los otros se despiertan al recibir un mensaje.

A primera vista podríamos pensar que dado que los fragmentos que se forman tienen un nivel exactamente igual al logaritmo de su tamaño, tanto T-D como R-U nunca tendrían éxito. En ningun caso un nodo de cualquier fragmento está a una distancia mayor que $2^{nivel(raiz)+1}$ de la raiz del fragmento ni tiene más de $2^{nivel(raiz)+1}$ aristas incidentes internas.

Sin embargo, recordemos que el algoritmo de Awerbuch aplica inicialmente una fase de conteo. Por tanto al comenzar la segunda fase, todos los nodos están despiertos y no se puede dar una ejecución como la que supusimos (Que solo v_1 despierte inicialmente).

En esta gráfica particular, si se tienen todos los nodos despiertos, no hay esperas e incluso al algoritmo de GHS le lleva tiempo O(n) hallar el AGM. Los fragmentos de nivel 1 se forman al mismo tiempo, los de nivel 2 también y así sucesivamente. Cada fragmento tarda un tiempo proporcional a su tamaño. Los fragmentos de nivel 1 tienen dos nodos, los de nivel 2 tienen cuatro, etc.

Generalizando el ejemplo a una cadena de n nodos con repartición equivalente de pesos, el tiempo que consume la ejecución de GHS (y de Awerbuch) es la suma de lo que tarda en formar los fragmentos de nivel 1, de nivel 2 , etc. hasta el nivel $\log n$: $\Sigma_{i=1}^{\log n} 2^i = O(n)$

8.2 El Algoritmo sublineal de Garay, Kutten y Peleg

8.2.1 El Problema

Nuevamente la motivación para el desarrollo de este algoritmo es la de reducir la complejidad de tiempo del algoritmo original de GHS pues, como ya sabemos, ésta no es óptima.

Sin embargo, en este caso, Juan Garay, Shay Kutten y David Peleg [GKP] modifican la definición de lo que en general se considera como una cota óptima del tiempo de un algoritmo distribuído: $O(n)$ donde n es el número de procesadores en una red. La justificación de considerar tal cota como óptima es básicamente que existen gráficas de n nodos donde esta cota es la mejor posible (un ejemplo típico es una cadena). Este tipo de optimalidad es *existencial* y a los autores les interesa trabajar con optimalidades *universales* con las cuales el algoritmo propuesto resuelve el problema óptimamente sobre cada instancia. Una característica muy importante de la optimalidad universal es que permite identificar precisamente los parámetros del problema que son los responsables de su complejidad lo que permite obtener cotas más 'finas' y exactas.

Los autores mencionados están interesados en mostrar la posibilidad de encontrar cotas más apropiadas para problemas de naturaleza esencialmente global -aquellos que no se limitan a soluciones locales- que requieren enviar al menos un mensaje por toda la red. Se concentran en desarrollar un algoritmo para el problema del AGM por considerarlo clásico y de gran interés práctico.

8.2.2 Los resultados

Uno de los parámetros que, de manera natural puede tomarse en cuenta para tratar de hallar cotas más exactas que $O(n)$ es el diámetro de una gráfica. En el caso del AGM, dado que el árbol puede tener una altura considerablemente mayor que el diámetro de la red, el problema de rebajar la clásica cota $O(n)$ es particularmente difícil.

Sin embargo los autores usan técnicas que les permiten reducir la cota supuestamente óptima obtenida por Awerbuch y obtener un algoritmo -lo llamaremos GKP- con una complejidad de tiempo sublineal en n. En concreto:

$$Complejidad\ tiempo(GKP) = O(n^{0.614} + Diam(G))$$

donde Diam(G) es el diámetro de la gráfica G, que se define como la distancia máxima entre dos nodos de G sin tomar en cuenta los pesos, es decir la distancia medida solo en términos del número de aristas en un camino.

En cuanto a la *Complejidad de mensajes* los autores no están interesados en ella y por tanto no la evaluan ni la tratan de controlar aunque queda claro que es mayor a la obtenida en GHS y mantenida por Awerbuch.

Cabe destacar que estos resultados son obtenidos sobre un modelo que difiere del estándar del capítulo 2 en una característica importante: Se asume una red síncrona. Si quisiéramos

aplicar el algoritmo GKP sobre una red asíncrona sería necesario usar un sincronizador [7] que es una metodología que permite a cualquier algoritmo síncrono correr en una red asíncrona. La complejidad en mensajes del sincronizador es $O(n^2)$ y por tanto se incrementaría la complejidad inicial de GKP.

8.2.3 La diferencia con GHS: Balance de Fragmentos

Este algoritmo usa como base las ideas de GHS. Sin embargo, más allá de las diferencias de naturaleza implementativa, se distingue de aquel algoritmo en la definición del balance de los fragmentos. En lugar de requerir mantener el balance en términos del número y del tamaño de los fragmentos, se lo requiere en términos del número y del diámetro de los mismos. Fusionará los fragmentos de manera que este balance se siga respetando en cada una de las fases. Esta característica es imprescindible para que el tiempo del algoritmo ya no dependa linealmente del número de nodos de la gráfica.

Sin embargo, mantener ese balance hasta hallar el AGM completo no es posible pues lo más probable es que éste sea mucho más profundo que el diámetro de la gráfica. Su altura es O(n) y por tanto un algoritmo que se basa en enviar mensajes a través de las aristas del árbol hasta que éste esté totalmente construído requiere tiempo O(n). Para mejorar el tiempo se necesita una aproximación diferente. En el algoritmo de GKP se ejecuta el algoritmo similar a GHS mienstras no haya problemas con mantener el balance de los fragmentos. Luego se cambia a otro algoritmo para terminar de elegir las aristas del AGM total.

8.2.4 La Idea General

EL algoritmo GKP está formado por tres partes:

La primera llamada GHS-CONTROLADO es la versión controlada de GHS que se encarga de construir los fragmentos balanceados. Al finalizar esta parte, se obtiene fragmentos de diámetros pequeños que formarán parte del AGM final y que cumplen los criterios de balance expuestos en la anterior sección.

En la segunda parte, se observan las aristas de salida de los fragmentos que resultaron de la primera parte (y de los que se vayan formando). Es de esperar que estas aristas formen ciclos y por tanto se eliminan las de mayor peso de cada ciclo, aristas que seguro no están en el AGM final. Para que la complejidad de tiempo no aumente, esta etapa se ejecuta sólo hasta cierto punto, específicamente se ponen restricciones a la longitud de los ciclos a eliminar.

En la tercera y última etapa, se utiliza un árbol generador BFS (Breath First Search) para realizar la comunicación necesaria para quitar las últimas aristas que no forman parte del AGM pero que no han sido eliminadas en la anterior etapa.

[7] Para mayor información sobre los sincronizadores consultar la referencia [AWR].

8.2.5 Encontrar un conjunto dominante pequeño de un árbol

En la primera etapa, para producir los fragmentos balanceados en términos de su número y de su diámetro, se computa en cada fase un conjunto dominante pequeño (CDP) sobre la 'supergráfica de fragmentos'. Ya que los nodos en tal gráfica corresponden a los fragmentos existentes en un momento determinado y las aristas dirigidas son las aristas de salida de menor peso que van de un fragmento a otro, un conjunto dominante en este caso se refiere al conjunto de fragmentos tal que los restantes están 'conectados' a alguno de ellos a través de su ASPM. Por pequeño entendemos que la cardinalidad del conjunto no excede alguna cota predeterminada. En nuestro caso, definiremos esa cota en breve.

De acuerdo a ese conjunto se tomará la decisión de que fragmentos fusionar. Como se verá posteriormente, en el algoritmo se unen fragmentos que no están en un CDP a fragmentos que si lo están. Las características del CDP aseguran dos cosas: Al ser conjunto dominante todos los fragmentos que no están en él se fusionan a alguno que si está [8] y no se forman cadenas largas de fusiones porque se fusionan solo fragmentos adyacentes. Al ser pequeño, se acota el número de fragmentos que se forman pues, como se verá, se forma un nuevo fragmento por cada elemento en el CDP.

Aunque la construcción de un CDP es sencilla, hemos preferido separar la descripción del procedimiento que lo computa del algoritmo completo para que el estudio del mismo sea más simple. Tal procedimiento -llamado Conj-dom-peq (en el artículo original: Small-Dom-set)- usa un subprocedimiento para computar el conjunto independiente maximal del árbol [9].

Sea:

$$L(v) = \begin{cases} 0 & \text{si } v \text{ es hoja} \\ 1 + \min_{u \in hijos(v)}(L(u)) & \text{sino} \end{cases}$$

es decir $L(v)$ es la longitud del camino más corto de una hoja a v,

y sea:

$$L_i = \{v \ tal \ que \ L(v) = i\}$$

La aplicación del algoritmo Conj-dom-peq sobre un árbol $T = (V, E)$ de la gráfica de fragmentos (que es un bosque) consta de los siguientes pasos:

i. Marcar los nodos $v \in V(T)$ tal que $L(v) = 0, 1, 2$. Es decir los nodos que sean hoja o estén a una distancia menor a dos de alguna hoja.

ii. Seleccionar un conjunto independiente maximal Q sobre el conjunto de nodos no marcados $R = V(T) - \{L_0 \cup L_1 \cup L_2\}$

iii. Construir el Conjunto dominante pequeño que está formado por los nodos que están en el conjunto independiente maximal elegido y los nodos que están a distancia 1 de una hoja, es decir: $CDP = Q \cup L_1$.

[8] Todo fragmento está conectado a uno de ellos

[9] Ver definiciones capítulo 2.

Notemos que, a pesar que este procedimiento se aplica a los árboles sobre el bosque de fragmentos, realmente es ejecutado sobre la gráfica original. Por tanto el procedimiento requiere que cada fragmento sea simulado por un nodo representativo, por ejemplo la raiz.

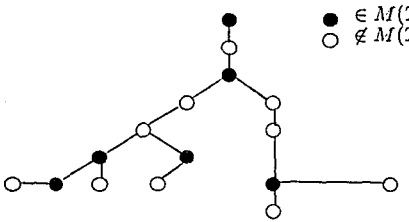


Figura 8.5: Un conjunto dominante pequeño M(T) sobre el árbol T

El propósito de este procedimiento es construir un conjunto dominante pequeño, es decir un conjunto CDP que cumpla con las siguientes dos condiciones:

1. CDP es un conjunto de vértices dominante de T
2. CDP debe ser *pequeño*. Para los propósitos del algoritmo es suficiente que $|CDP| \leq \frac{|V_T|}{2}$

Se debe demostrar que el CDP construído por Conj-dom-peq las satisface.

Lema 8.2.1 *Sea CDP el conjunto formado al aplicar el procedimiento Conj-dom-peq sobre un árbol* $T = (V_T, E_T)$. *Entonces* $\forall v \notin CDP, \exists v_1 \in CDP$ *tal que* $(v, v_1) \in E_T$

Demostración i) Si $v \in R$ y $v \notin CDP$, entonces $v \notin Q$. Si $v \notin Q$ es porque existe un $v_1 \in Q$ adyacente a v porque sino Q no sería maximal y $Q \subset CDP$.

ii) Si $v \notin R$ y $v \notin CDP$, entonces $v \in L_0$ o $v \in L_2$. En ambos casos debe existir $v_1 \in L_1$ adyacente a v y $L_1 \subset CDP$. ∎

Lema 8.2.2 $|Q| \leq \frac{|R \cup L_2|}{2}$

Demostración Para todo nodo v en Q seleccionamos un v_1 distinto talque $v_1 \in (R \cup L_2) - Q$. Esto se hace designando como v_1 un hijo arbitrario de v y que cumple con la característica que se quiere de v_1 porque por definición los vértices de Q siempre tienen un hijo en $R \cup L_2$ (los nodos en Q pertenecen a $L_3, L_4, \ldots$). Además, puesto que cada nodo en T tiene un único padre, se garantiza que los nodos v_1 sean todos distintos.

De ahí: $|R \cup L_2| \geq 2 * |Q|$
pues por cada nodo $v \in Q$, existe al menos un $v_1 \in (R \cup L_2) - Q$ y ambos están en $|R \cup L_2|$.

■

Lema 8.2.3 $|CDP| \leq \frac{|V_T|}{2}$

Demostración Sabemos que $CDP = Q \cup L_1$ por construcción.

Claramente $|L_1| \leq |L_0|$ pues T es un árbol y cada hoja tiene un solo padre. (1)

Usando (1) :

$$|L_1| \leq |L_0|$$

$$2|L_1| \leq |L_0 \cup L_1|$$

$$|L_1| \leq \frac{|L_0 \cup L_1|}{2} \quad (2)$$

De (2) y el lema 8.2.2 obtenemos que:

$$|CDP| = |Q \cup L_1| \leq \frac{|R \cup L_2|}{2} + \frac{|L_0 \cup L_1|}{2} = \frac{|V_T|}{2}$$

■

8.2.6 El algoritmo

El algoritmo GKP consiste de tres partes:

PARTE I: GHS-CONTROLADO.- La versión modificada de GHS es más sencilla. No se limita el número de mensajes y se asume un algoritmo síncrono. En particular, no se preocupa de balancear el tamaño de los fragmentos usando las reglas para el control y crecimiento de los niveles del algoritmo original ya que los niveles son impuestos por la sincronización de fase (las fases están sincronizadas, es decir todos los fragmentos terminan de computar una fase antes de iniciar la siguiente).

Cada fase del algoritmo GHS-Controlado consiste de dos pasos:

1. Ejecuta una fase de GHS hasta el punto donde cada fragmento F ha elegido su arista de salida de peso mínimo y por tanto ya ha decidido con que otro fragmento se quiere unir. Con los fragmentos y sus ASPMs se construye la gráfica o bosque de fragmentos denotada por FF.

2. Rompe los árboles resultantes de FF en pequeños árboles con profundidad $O(1)$ y fusiona estos 'arbolitos' siguiendo el siguiente procedimiento: Se computa una implementación distribuída del procedimiento Conj-dom-peq aplicándolo sobre cada árbol T de FF. Llamemos $M(T_{FF})$ al conjunto dominante de un árbol T_{FF}. Para cada nodo que esté en T_{FF} y no esté en $M(T_{FF})$ se elije uno de sus nodos vecinos que si esté en $M(T_{FF})$, llamémosle F' y se fusionan F y F'. De esa manera, las fusiones que se realizan en cada fase forman 'estrellas' en FF pues cada nodo que no está en el conjunto dominante de su árbol elige uno solo de sus vecinos que si está. Por otro lado, un nodo de $M(T_{FF})$ puede ser elegido por varios que no estén en $M(T_{FF})$. Entonces las estrellas que se forman tendrán como centro un nodo en $M(T)$ al que le 'rodean' uno o más nodos que no están en el conjunto dominante. (ver figura 8.7)

   Así se previene fusiones sobre largas cadenas y se limita el diámetro del fragmento resultante lo que no sucedía en GHS donde cada árbol del bosque de fragmentos se podía convertir en un único fragmento.

   Cabe recalcar que como las aristas de salida de peso mínimo de cualquier fragmento están seguro en el AGM final, la nueva manera de fusionar también es correcta.

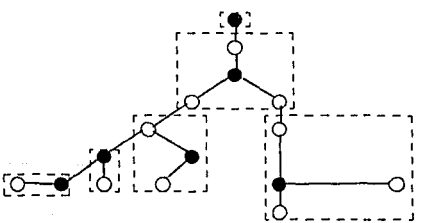


Figura 8.6: El árbol T de la figura anterior se rompe en pequeños árboles

El proceso recién descrito se ejecuta I fases, donde I se definirá posteriormente, empezando con la fase donde cada fragmento es un nodo solo. Cada una de dichas fases cumple con las siguientes propiedades:

1. El número de fragmentos es a lo sumo la mitad del número de fragmentos en la fase previa.
   Esto se ve claramente usando el lema 8.2.3. $|M| \leq \frac{|F|}{2}$ donde $|F|$ es el número de nodos

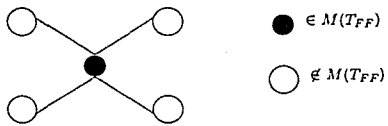


Figura 8.7: 'Estrellas'

(fragmentos) del bosque de fragmentos y cada estrella tiene uno y solo un nodo de M. Por tanto *numero de estrellas* $\leq \frac{|F|}{2}$ y hay tantos nuevos fragmentos como estrellas pues de cada estrella sale un nuevo fragmento.

2. El diámetro de cualquier fragmento F crece en un factor de a lo sumo 3 (Se acota diámetro) pues la distancia entre dos fragmentos en las estrellas formadas en el bosque de fragmentos luego de aplicar el paso 2 es a lo sumo dos (ver figura 8.7) y así el camino más largo incluye a lo sumo tres fragmentos.

Por tanto, luego de ejecutar la i-ésima fase se tiene:

- *No. fragmentos* $\leq \frac{n}{2^i}$ y
- *Diametro*$(F) \leq 3^i, \forall$ *fragmento* F

Luego de la I-ésima fase, es decir luego de la última fase en la que se aplica este algoritmo se cumple:

- *No. fragmentos* $\leq N$ y
- *Diametro*$(F) \leq d, \forall$ *fragmento* F, donde N y d se especifican después

PARTE II: ELIMINACION DE CICLOS DE LA GRAFICA DE FRAGMENTOS.- Como dijimos, la primera parte no termina con el AGM total sino con un número (menor a N) de fragmentos del mismo.

A partir de ellos se construye una supergráfica denotada por $F' = (V', E')$ con:
V'=conjunto de fragmentos resultantes de la parte I y
E'=conjunto de todas las aristas de salida de todos los fragmentos

Dado que se incluyen todas las aristas de salida, esta gráfica puede tener ciclos interfragmentos [10] que deben ser eliminados. Para ello se elimina la arista de mayor peso de cada uno pues seguro no está en el AGM.

En la raiz de cada fragmento F se va guardando información sobre el camino de menor peso de F a cualquier otro fragmento conectado a él.

[10] Los fragmentos son árboles, obviamente no habrá ciclos dentro de ellos

Esa información, al ser difundida por todo el subárbol, permite a cada uno de los nodos del fragmento detectar si es incidente a una arista que forma parte de un camino entre F y otro fragmento que no es el de menor peso [11]. Si el camino es de longitud 1, lo que está haciendo es detectar si es incidente a la arista de mayor peso en algún ciclo de longitud 2. En cualquiera de los dos casos, el nodo correspondiente marca inmediatamente la arista como que no está en el AGM.

F va detectando los ciclos en los que está incluído al intercambiar información con sus fragmentos vecinos y en orden de longitud, es decir primero detecta los ciclos de longitud dos, luego los de tres y así sucesivamente.

Se describe inicialmente el procedimiento de eliminar ciclos de longitud dos para luego generalizarlo a ciclos de mayor longitud.

Procedimiento para eliminar ciclos de longitud 2

En la gráfica de fragmentos, los ciclos de longitud 2 aparecen cuando (al menos) dos nodos de un fragmento F son adyacentes a dos nodos, posiblemente el mismo, de otro fragmento F_1 [12]. De estas aristas que conectan un par de fragmentos se eliminan todas menos la de menor peso siguiendo el siguiente procedimiento:

i. Cada nodo de F crea un registro $Camino_1(F_1)$, para todo fragmento F_1 adyacente a él en F', que contiene la siguiente información :
   - Una arista $e = (u, v), v \in F, u \in F_1$.
   - Los identificadores de u, v y F_1.
   - El peso de la arista e, denotado como $peso(e)$.

ii. Todos los nodos de F envían todos los registros formados en el anterior paso hacia la raiz de F en la siguiente forma convergente.

   Cada hoja h elige solo un registro $Camino_1(F_1)$ por cada fragmento F_1 adyacente a F por h. Si existen varias aristas $(h, f_1), f_1 \in F_1$, entonces h elige el registro correspondiente a la arista de menor peso y a las restantes las marca como NO-USABLE. Los registros así elegidos son enviados uno por uno empezando por el del fragmento con menor identificador y terminando con el de mayor.

   Cada nodo interno v almacena todos los registros que recibe de sus hijos y los suyos propios, si tiene varios registros para un mismo fragmento, elimina todos menos el que tenga la arista con menor peso.
   En cada paso, v envía el registro del fragmento cuyo identificador sea el de menor identificador que conoce hasta el momento. De hecho, los registros son ordenados (pipelined) por los identificadores de fragmentos. Si identificador(F_1) < identificador(F_2), los nodos a lo largo de F que reciben los dos registros envían $Camino_1(F_1)$ antes que $Camino_1(F_2)$.

[11] Si hay dos caminos de F a otro fragmento, cada pair de estos caminos forman un ciclo. Solo nos interesa quedarnos con el camino de menor peso entre todos ellos

[12] Recordemos que se incluyen todas las aristas de salida.

Cada nodo v empieza a enviar registros en tiempo $L(v)$, donde $L(v)$ es la función que definimos antes.

iii. La raiz de F va almacenando los registros que le llegan en una estructura $DS_1(F) = \bigcup_i(Camino_1(F_i)), F_i$ *adyacente a* F

iv. La raiz difunde $DS_1(F)$ para que cada nodo $v \in F$ adyacente a una arista de salida $e = (v,u)$ pueda decidir si esa arista es todavía candidata de quedarse en el árbol final o ya puede ser eliminado definitivamente.

Sea $e = (v,u), u \in F_1$: Si e está en el registro de $DS_1(F)$ correspondiente a F_1 puede que esté en el AGM final y por tanto v no la elimina. En caso contrario, v la marca como NO-USABLE.

Al término de este procedimiento se han eliminado (marcado con NO-USABLE) las aristas que seguro no están en el AGM. Las restantes, es decir las que tienen su registro incluído en DS_1 aun son candidatas para estar en el árbol final pues el orden de envío que se ha utilizado asegura que el registro $Camino_1(F_1)$ que está en DS_1 sea el de la arista de menor peso que una un nodo de F a uno de F_1 [13]. En el AGM total talvez F y F_1 no sean vecinos pero si lo son, seguro que será por la arista cuyo registro se ha mantenido en DS_1. Para que esto sea cierto, es básico que la forma en que vamos enviando los registros hacia la raiz asegure que un nodo envíe el registro de un fragmento F_1 después de haber recibido todos los registros para F_1 que le podían enviar sus descendientes y por tanto, envíe el correspondiente.

El siguiente lema sirve para demostrar formalmente que el algoritmo realmente cumple con esta característica:

Lema 8.2.4 *Cada nodo $v \in F$ envía a su padre exactamente un registro $Camino_1(F_1)$ por cada fragmento F_1 que es adyacente a nodos en el subárbol cuya raiz es v y estos registros son mandados hacia arriba en orden creciente de identificador.*

Demostración La prueba se basa en demostrar que en el tiempo $t = j + i$ un nodo v con $L(v) = j$ ya ha recibido de sus descendientes todos los registros del fragmento con el i-ésimo identificador del conjunto de los fragmentos vecinos a los nodos del subárbol con raiz v, ordenados en forma creciente. Se hará la demostración por doble inducción sobre i y j.

Base: $L(v) = j = 0$. Una hoja empieza a enviar registros a su padre en t=0 de menor a mayor identificador.

El primero , i=0, es enviado en $t = L(v) + i = 0 + 0 = 0$.

Ahora supongamos que se cumple que el registro del fragmento con el l-ésimo identificador ($i = l$) es enviado en $t = l$, obviamente se cumple que el del fragmento con el (l+1)-ésimo identificador es enviado en la siguiente unidad de tiempo, o sea en $t = l + 1$.

[13] Se han eliminado las aristas de mayor peso de todos los ciclos $F - F_1 - F$.

Paso inductivo: Para los nodos internos. Supongamos que se cumple para todo nodo v, con $L(v) = j \leq m$. Veremos que sucede con un nodo u con $L(u) = m+1$. Los nodos hijos de u, h_u, cumplen que $L(h_u) < m+1$.

Por hipótesis han enviado el registro con menor identificador en tiempo $t < m+1 (i = 0)$. Como la red es síncrona, seguro que en el tiempo $m+1$ u ha recibido todos estos registros de los hijos que debían enviárselos.

Para el (l+1)-ésimo identificador el razonamiento es equivalente. Por hipótesis sobre i, para $t = (m+1)+l = m+l+1$, u ya ha recibido todos los registros correspondientes al fragmento con l-ésimo identificador y por hipótesis sobre j en el mismo instante de tiempo $t = m+(l+1) = m+l+1$, todos sus hijos ya han recibido los fragmentos correspondientes del (l+1)-ésimo fragmento. Así, en el siguiente instante de tiempo, $t = (m+1)+(l+1)$, u recibe los registros del fragmento con (l+1)-ésimo identificador que aún no había recibido en $t = (m+1)+l$. ■

Procedimiento para eliminar los ciclos restantes

Al final del anterior procedimiento obtenemos los caminos de longitud igual a 1 con menor peso. Básicamente se repetirá estre procedimiento por l fases, eliminando en la i-ésima fase ($1 \leq i \leq l$) los ciclos interfragmentos de longitud $2*i-1$ y $2*i$ y manteniendo los caminos de menor peso a fragmentos que estén a distancia i en la gráfica de fragmentos F'. La eliminación, siguiendo con la misma idea que antes, se realiza desechando el camino que contiene la arista de mayor peso en los ciclos detectados.

El procedimiento que se sigue en un fragmento F en la i-ésima fase es el siguiente:

i. Los registros se envían a la raiz de manera similar a la del anterior procedimiento: Cuando dos caminos, de longitud menor o igual a i de F a otro fragmento F_1 compiten entre ellos, se desecha el que contiene la arista de mayor peso.

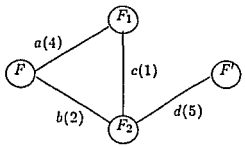
ii. Cuando la raiz recibe todos los registros de sus hijos, ensambla la estructura $DS_i(F)$ y la difunde por todo F identificando el nodo extremo en F del camino que queda a cada fragmento F_1 para que se marque como responsable del camino $F - F_1$.

iii. Cuando un nodo $v \in F$ recibe $DS_i(F)$ marca como NO-USABLE a sus aristas incidentes que estaban en alguno de los caminos eliminados (los que no están incluídos en $DS_i(F)$) y, si es responsable de algún fragmento F_1, envía la estructura $DS_i(F)$ completa por la primera arista (v,w) del camino a F_1.

iv. Ya que esto se realiza en todos los fragmentos, nodos en F adyacentes a nodos de fragmentos vecinos F_j reciben la estructura $DS_i(F_j)$ y consecuentemente pueden aprender los nuevos fragmentos que estén a distancia $i+1$ de F en F'.
Esta nueva información sirve para construir los registros $Camino_{i+1}(F)$ para todos los fragmentos recién conocidos, es decir no accesibles desde F por caminos de longitud

menor a $i+1$. Para los ya conocidos se elige el camino de longitud menor o igual a $i+1$ que no tenga la arista de menor peso en un ciclo, este camino puede ser el que ya se tenía de longitud menor a $i+1$ o el recién encontrado. (Ver figura 8.8).

Este proceso se reinicia para la siguiente fase hasta llegar a la fase $l = \log n$.



registro	F_1	F_2	F'
$DS_1(F)$	a(4)	b(2)	-
$DS_2(F)$	bc(3)	b(2)	-
$DS_3(F)$	bc(3)	b(2)	bd(6)

Figura 8.8: Un ejemplo

PARTE III: ELIMINACION GLOBAL DE ARISTAS.- Al término de la parte II, se ha eliminado todos los ciclos de longitud menor o igual $2l$ (con l como la definimos anteriormente) en la gráfica de fragmentos F', al haber eliminado todas las aristas de mayor peso de cada uno de ellos. Sin embargo quedan aún los ciclos de mayor longitud. Se debe proceder, entonces, a reducir el número total de aristas para mantener sólo las que forman el AGM.

El método es como sigue:

1. Construir un árbol BFS, denotado por B, sobre la gráfica original G.
2. La raiz de cada fragmento F obtenido en la parte I envía la lista de sus aristas externas aún no eliminadas a través de las aristas de B hacia su raiz.
3. La eliminación final de las aristas de mayor peso en ciclos de longitud mayor a 2l se realiza centralizadamente en la raiz de B.
4. La raiz de B difunde el AGM resultante a todos los nodos usando las aristas del árbol B

Notemos que la eliminación de las últimas aristas se realiza de forma centralizada pero usando comuninación sobre un árbol generador BFS debido a que la profundidad del mismo es O(diámetro(G)).

8.2.7 Correctez y Complejidades

El análisis de la correctez y de la complejidad de tiempo se hará parte por parte. Recordemos que la complejidad de mensajes no interesa en este caso.

En las demostraciones se hace uso de tres lemas para los cuales definiremos F'_i la gráfica obtenida de F' luego de la i-ésima iteración de la parte II

Lema 8.2.5 *Sea F_1 un fragmento tal que la distancia$_{F'}(F, F_1) \leq i$[14], entonces F'_i contiene un único camino que conecta F con F_1 y $DS_i(F)$ tiene el correspondiente registro $Camino_i(F, F_1)$.*

Demostración Se hace la demostración por inducción sobre i.
Base: $i = 1$. F_1 se construye aplicando el primer procedimiento. Se ve que el proceso de selección de las aristas que van a fragmentos vecinos asegura que la raiz se queda al final con una sola arista, la de mínimo peso entre F y cada uno de sus fragmentos vecinos. DS_1 contiene justamente las aristas que recibe la raiz y al difundir DS_1 todos los nodos de F eliminan sus aristas de salida excepto las que están en algún registro de DS_1.

Paso inductivo: Supongamos que se cumple para i, veremos que sucede para $i + 1$. Nuevamente hay que fijarse en el proceso de selección de los caminos que al final llegan a la raiz. Los caminos que van a fragmentos nuevos (no accesibles con caminos de longitud menor o igual a i) son comparados por cada nodo v que sólo envía a su padre el de menor peso entre los caminos accesibles por nodos del subárbol con raiz v de manera que al final la raiz se quede con el de menor peso. Para los fragmentos ya accedidos anteriormente la raiz también se queda con un solo camino, el que ya se tenía de longitud menor a $i + 1$ (es uno solo por hipótesis inductiva) o el de menor peso entre los nuevos caminos de longitud $i + 1$. ∎

Lema 8.2.6 *La longitud del ciclo más chico en F'_i es mayor que $2i$. Es más, el conjunto de aristas $\{e$ tal que $e \in F', e \in F'_i\}$ coincide con el conjunto $\{e$ tal que e arista de mayor peso en algun ciclo de longitud $2i$ o menos en $F'\}$*

Demostración Por inducción sobre i.
Base: $i = 1$. Como se vio en el lema 8.2.5 nos quedamos con un solo camino entre F y cualquier fragmento vecino F_1. Este camino está formado por la arista de menor peso (e) entre F y F_1 y por tanto se elimina toda arista e_1 tal que $e_1 = (u, v), u \in F, v \in F_1$ que es la arista de mayor peso en un ciclo de longitud $2i = 2$ (formado por las aristas e y e_1).

Paso inductivo: Suponiendo que se cumple para i, en F'_{i+1} ya no están las aristas de mayor peso en ciclos de longitud menor o igual a $2i$ en F' y, como se ve en el tercer paso del procedimiento , en la (i+1)-ésima fase se elimina la arista de mayor peso de los ciclos de longitud $2(j + 1)$. ∎

[14] Es decir la distancia entre F y F_1 en la supergráfica F'

Teorema 8.2.7 *El algoritmo GKP es correcto*

Demostración Parte I: Nos interesa asegurar que al final de la misma se cumple que:
- $No.fragmentos \leq N$
- $Diametro(F) \leq d, \forall F$

donde N y d sean lo suficientemente pequeños como para mantener el balance que nos interesa.

Recordemos que el diámetro del fragmento crece en un factor de 3 a lo sumo y que en cada fase de GHS-CONTROLADO, el número de fragmentos es a lo sumo la mitad del número de fragmentos de la anterior fase. Adoptando $N = \frac{n}{2^l}$ y $d = 3^l$ se cumple ambas condiciones.

Parte II: La correctez de la parte II se deduce usando los lemas 8.2.5 y 8.2.6. Fijando $l = \log n$, la parte II termina habiendo eliminado los ciclos de longitud menor o igual a $2 * \log n$

Parte III: La correctitud de esta última parte y del resultado final del algoritmo sale del hecho de haber eliminado en todo momento las aristas de mayor peso de cada uno de los ciclos en la gráfica(en la parte II de los de longitud menor o igual a 2logn y en la parte III de los restantes) que seguro no están en el AGM, de manera que al término de la parte III las únicas aristas que no eliminamos son las que forman el árbol generador de peso mínimo. ∎

Lema 8.2.8 *Cuando el algoritmo GHS-CONTROLADO es activado para I fases, toma tiempo* $O(3^I * 2^{\sqrt{\log n}})$

Demostración El primer paso en el procedimiento Conj-dom-peq es marcar los nodos de L_0, L_1, L_2. Cada hoja F_h en el bosque de fragmentos se identifica a sí misma como tal simplemente recorriendo el fragmento F_h correspondiente en la gráfica original en tiempo $O(diametro(F_h))$ y luego se puede marcar cada uno fragmento F de L_1 y L_2 en un tiempo $O(diametro(F))$.

En el segundo paso, la computación del conjunto maximal independiente de forma síncrona y aplicando un algoritmo como el de [REF PS EN GKP] en una gráfica con n nodos toma un tiempo $O(2^{\sqrt{\log n}})$. Como en el caso del algoritmo Conj-dom-peq los nodos del bosque de fragmentos son en realidad árboles, cualquier operación sobre el bosque implica recorrer los fragmentos en la gráfica original y por tanto el tiempo que consume la construcción del Conjunto Independiente Maximal es $O(2^{\sqrt{\log n}}) * Diametro(F)$.

Se ve que, en cada fase i, $1 \leq i \leq I$, GHS-CONTROLADO toma *tiempo* $\leq Diametro(F) * O(2^{\sqrt{\log n}}) \leq 3^i * O(2^{\sqrt{\log n}})$ por la primera propiedad que cumple cada fase j.

De donde:

$tiempo\ total\ GHS-CONTROLADO = \Sigma_{i \leq I}(3^i * O(2^{\sqrt{\log n}})) = 3^I * O(2^{\sqrt{\log n}})$ ∎

Lema 8.2.9 *El tiempo total de ejecución de la parte II de GKP es de* $O(\frac{n}{2^l} + 3^l) * \log n$

Demostración La i-ésima fase de eliminación de ciclos involucra el intercambio de estructuras DS_{i-1} por todas las aristas de salida, un proceso convergente para formar la DS_i y la difusión desde la raiz de la estructura ya completa.

El intercambio de las DS_{i-1} se hace al momento que la estructura va difundiéndose en el fragmento.
Como vimos en el lema 8.2.4, un nodo v talque $L(v) = j$ recibe el fragmento con i-ésimo identificador entre los fragmentos vecinos a los nodos del subárbol con raiz v en tiempo $t = j + i$. Si r es la raiz de un fragmento F, $L(r) \leq diam(F)$, por tanto a lo sumo en el tiempo $t = diam(F) + i$, r ya recibió el registro del fragmento con i-ésimo identificador y, como el número de fragmentos es a lo sumo N, r recibe los registros de todos los fragmentos vecinos a F en O(diam(F)+N). De igual manera, en el proceso de difusión cada nodo envía a sus hijos DS_I mandando los registros de los fragmentos ($O(N)$), uno tras otro. Nuevamente se requiere tiempo $O(diam(F) + N)$.
Sin embargo, notemos que cada registro $Camino_i(F)$ en la i-ésima iteración consiste de a lo sumo i aristas y por tanto tiene longitud $i \log n$. Como aceptamos mensajes de longitud $\log n$ entonces tenemos tiempo $O(i * diametro(F) + N)$ para la i-ésima iteración y la parte II completa toma $O((Diametro(F) + N) \log n)$.

Sabiendo que $N = \frac{n}{2^I}$ y que el diámetro de cualquier fragmento es a lo sumo 3^I y además fijando $I = \log n$: Entonces:

$$tiempo\ parte\ II = O(\log n(\tfrac{n}{2^I} + 3^I))$$

■

Lema 8.2.10 *La parte III lleva un tiempo total de a lo sumo* $\frac{n}{2^I} + Diametro(G)$

Demostración El número total de aristas que deben enviarse hasta la raiz del árbol B es O(N). Como B es un árbol BFS su profundidad es O(diam(G)) y por tanto enviar todas las aristas a la raiz requiere tiempo $O(N + diam(G))$.

Difundir el AGM final desde la raiz lleva también tiempo $O(N + diam(G))$ ■

Teorema 8.2.11 *La complejidad de tiempo del algoritmo GKP es* $O(diam(G) + n^{0.614})$

Demostración La complejidad de tiempo total es optimizada eligiendo I tal que $3^I = \frac{n}{2^I}$ esto es $I = \frac{\ln n}{\ln 6}$.
De donde: $3^I = \frac{n}{2^I} = n^{\frac{\ln 2}{\ln 6}} = n^{0.614}$

Entonces:
Complejidad tiempo GKP =
$diam(G) + n^{0.614} * O(2^{\sqrt{\log n}}) + O(n^{0.614} * \log n) = O(diam(G) + n^{0.614})$ ■

8.3 El Algoritmo de Park, Masukara, Hagilara e Ickura

En esta sección hacemos una breve descripción del algoritmo presentado en el artículo de Park, Masukara, Hagilara y Tokura [PAR], (algoritmo AP-PMHT) para resolver el llamado 'Problema de Actualizar el Arbol Generador Mínimo' - en inglés 'UMP' o 'Updating MST Problem'-

8.3.1 El problema

En una red real, la topología de la red cambia a menudo debido a fallas - y posteriores recuperaciones- en nodos y aristas; de ahí el interés de manejar eficientemente las posibles consecuencias de tales cambios.

El UMP consiste en reconstruir el AGM de una red luego de que varias líneas y/o nodos han sido eliminados y/o adicionados. Se asume que no se dan cambios de topología durante la ejecución del algoritmo.

El interés de resolver el UMP surge de la suposición natural de que el AGM - T_1 - de una red G_1 resultante de cambios en la topología de la red original G coincide en gran medida con el AGM de G - T - y que por tanto podría aprovecharse una parte de T para hallar T_1 sin tener que calcularlo desde el principio utilizando un algoritmo como el de Awerbuch, por ejemplo.

8.3.2 Los resultados

En principio, AP-PMHT reconstruye el AGM de una red después de que varias líneas han sido borradas y/o adicionadas. Sin embargo, los mismos autores le hacen una sencilla ampliación para que maneje también el caso de eliminación y/o adición de procesadores.

Al obtener la nueva solución conociendo la anterior, el algoritmo AP-PMHT mejora la complejidad en número de mensajes en comparación a aplicar Awerbuch nuevamente desde el inicio.

En cuanto al tiempo, la complejidad de AP-PMHT es levemente inferior a la de Awerbuch. Sin embargo la diferencia no es significativa a no ser en casos donde el número de aristas eliminadas y adicionadas es muy grande.

En concreto:

$$Numero\ mensajes(AP-PMHT) = O\ (m + n\ log(t+f)\ )$$

y

$$tiempo(AP-PMHT) = O\ (n + n\ log(t+f)\ )$$

con

n = número de procesadores en la red

t = número de aristas adicionadas
f = número de aristas eliminadas del anterior A
Y si e_1 es el número aristas resultante:

$$m = \begin{cases} t+n & \text{f=0} \\ e_1 & \text{si no} \end{cases}$$

Aclaramos que los resultados fueron obtenidos asumiendo que se trabaja sobre una red asíncrona que sigue el modelo estándar presentado en el capítulo 2.

8.3.3 La complicación de agregar aristas

Un problema más sencillo sería reconstruir el AGM después de que un conjunto de aristas ha sido eliminado. En ese caso quedan varios fragmentos del AGM original - **T** - que a su vez son subárboles del AGM nuevo - $\mathbf{T_1}$ -. Para construir $\mathbf{T_1}$ completo, se puede aplicar un algoritmo muy similar a GHS a partir de los fragmentos resultantes.

Sin embargo, tratar con la adición de aristas agrega cierta complicación a la solución del problema. Las aristas eliminadas siguen dividiendo el anterior AGM en varios fragmentos pero éstos pueden no ser fragmentos del nuevo AGM debido a la presencia de las nuevas aristas .En el nuevo AGM, por supuesto, ya no estarán las aristas borradas pero no se sabe a priori lo que sucederá con las adicionadas, éstas pueden tener pesos tales que sean incluídas en $\mathbf{T_1}$ en lugar de algunas que estaban en T.

8.3.4 La idea principal

La idea central del algoritmo es encontrar los fragmentos del anterior AGM que seguro estarán en el nuevo habiendo adicionado y eliminado aristas. A partir de ellos se reconstruye $\mathbf{T_1}$ a usando un algoritmo muy similar al de GHS.

El algoritmo AP-PMHT consta de dos partes:

En una primera se forman los fragmentos que seguro estarán en el nuevo AGM. Como se señaló en la anterior sección, las aristas de **T** (AGM original) que se eliminaron, partieron a T en fragmentos. Pero aun resta eliminar algunas aristas que no están en el nuevo AGM. El criterio que se sigue es quitar las aristas que podrían ser las de mayor peso de los ciclos que posiblemente se formaron con las aristas adicionadas y las que quedaron de **T**. Estas aristas reciben el nombre de "Aristas de Partición" y de ahí que hayamos decidido llamar **AP** a esta parte del algoritmo.

En la segunda, se aplica la versión modificada del algoritmo de GHS sobre los fragmentos que resultaron de la primera etapa. Los autores denominan a este algoritmo **PMHT**.

8.3.5 Encontrar los fragmentos iniciales

Como señalamos, la primera parte del algoritmo es la de identificar fragmentos lo más grandes posible a partir de los cuales reconstruye el nuevo AGM. Para ello, el algoritmo AP-PMIIT debe contar con la siguiente información:

- El AGM original (T)
- Aristas de T eliminadas (A_1)
- Aristas adicionadas (A_2)

Comienza con los fragmentos $F_{T-A_1} = \{F_i, i = 1...t\}$ de T que se obtienen de eliminar las aristas A_1. Como mencionamos, éstos aún pueden ser excesivamente grandes. Por tanto, para obtener fragmentos (posiblemente de menor tamaño) que seguro estarán en T_1 se quitan las aristas que podrían entrar en conflicto con las adicionadas, llamemos a este grupo de aristas A_3.

Para hallar las aristas que forman A_3 el razonamiento es el siguiente:

Una arista $(v_x, v_y) \in A_2$ tal que $v_x \in F_i$ y $v_y \in F_j$, $i <> j$ y $F_i, F_j \in F_{T-A_1}$ (es decir una arista que una a dos nodos de distintos fragmentos) no trae problemas pues será analizada al momento de unir los fragmentos. Pero puede suceder que ambos nodos incidentes a la arista adicionada (v_x, v_y) estén en un mismo fragmento $F_i \in F_{T-A_1}$ en cuyo caso forma un ciclo con las aristas del camino entre v_x y v_y que ya existía en F_i.

De cada uno de estos ciclos se debe eliminar la arista de mayor peso (ésta puede estar en A_2 o en $T - A_1$) para que quede un camino simple entre todos los nodos que lo forman. Sin embargo, recorrerlos todos uno a uno para hallar las aristas a eliminar es muy costoso. En lugar de ello, se recorren únicamente las aristas de los subárboles de $T - A_1$ para eliminar las aristas que podrían ser las de mayor peso en algún ciclo.

Al visualizar uno de tales ciclos en el subárbol F_i notamos que debe ser de la forma $v_x...v_{xy}...v_y$ donde v_{xy} es un antecesor común de v_x y v_y (ver figura 8.9).

En particular, si el ciclo es simple, v_{xy} es el antecesor común más cercano a v_x y v_y.

El algoritmo AP-PMIIT utiliza esta característica para detectar un ciclo cuando encuentra un nodo antecesor de los dos nodos incidentes a una arista adicionada y luego elige la arista que posiblemente es la de mayor peso en el ciclo.

A partir de estas aristas y las borradas se define formalmente el conjunto de 'aristas de partición' (A_p) para lo que requiere las siguientes definiciones:

Un *procesador incidente* es un procesador incidente a una arista adicionada o a una borrada.

Un *procesador puente (branch)* es un procesador que es el antecesor común más cercano a cualquier par de procesadores incidentes.

Un *procesador marcado* es un procesador incidente o un procesador branch.

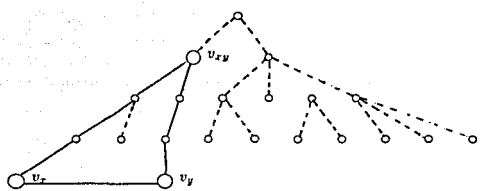


Figura 8.9: (v_x, v_y) es una arista adicionada

Camino Primario u-v. Es el camino de u a v en T, si u y v son procesadores marcados y no existe otro procesador marcado en ese camino.

Entonces una *Arista de partición* es la arista de mayor peso en un camino primario.

Es importante recalcar que esta definición de A_p incluye a las aristas borradas (u, v) pues cada una de ellas es por si sola un camino primario (u y v son procesadores marcados y (u, v) está en T) y no incluye a las aristas adicionadas pues éstas no están en T.

8.3.6 Dos lemas importantes

Presentamos en esta sección dos lemas que justifican la exclusión, al momento de construir T_1 y en caso que no se hayan eliminado aristas del AGM original T, de las aristas eliminadas que no formaban parte de él y en general de todas las aristas de la red original que no estaban en él. Esto implica que no se tengan que enviar mensajes por ellas a lo largo de todo el algoritmo AP-PMHT.

Lema 8.3.1 *Sea* G *la gráfica original y* T *su AGM. Si* G_1 *se obtiene de* G *exclusivamente eliminando aristas de* G-T, *entonces el AGM de* G_1, T_1, *coincide con* T.

Demostración Por reducción al absurdo supongamos que $T <> T_1$.

Las aristas de T siguen estando en G_1 (pues solo se han eliminado aristas fuera de él) y por tanto T es un árbol generador de G_1 con $peso(T) > peso(T_1)$ pues T_1 es el árbol generador mínimo de G_1.

Por otro lado, las aristas de T_1 están en G (no se han adicionado aristas) y por tanto T_1 es un árbol generador de G con $peso(T_1) > peso(T)$ pues T es el AGM de G.

Contradicción! pues no se puede cumplir que $peso(T) > peso(T_1)$ y $peso(T_1) > peso(T)$ al mismo tiempo. ■

Lema 8.3.2 *Sea* G *la gráfica original y* T *su AGM. Si* G_1 *se obtiene de* G *sin eliminar aristas de* T *y adicionando las aristas* A_2, *entonces el AGM de* G_1 *coincide con el AGM de* $T + A_2$.

Demostración Por reducción al absurdo, supongamos que en T_1 existe una arista e tal que $e \in G - T$, es decir que no está en T y tampoco en A_2.

Sea $T^- \subset T$ el conjunto de aristas que no están en T_1 y $T^+ = T - T^-$ es decir el conjunto de aristas de T que están en T_1.
a) El conjunto de aristas de A_2 incluídas en T_1 - denotado por A_2^1 - debe cumplir que sus pesos sean menores a los de las aristas de T^-.
b) Por otro lado e es de mayor peso que las aristas de T porque de lo contrario T no sería AGM de G.

Supongamos que formamos un árbol generador de G_1: $T_2 = T_1 - e + t$, donde $t \in T^-$ une los dos fragmentos de T_1 que quedaron separados a eliminar e (tal t existe por que de lo contrario T no hubiera sido AGM de G).
Por a) $peso(e) > peso(a_2) \; \forall \; a_2 \in A_2^1$
Por b) $peso(e) > peso(x) \; \forall \; x \in T$, en particular de t.

Por tanto $peso(T_2) < peso(T_1)$ contradiciendo la suposición que T_1 era el AGM de G. ■

8.3.7 Esquema del algoritmo

-Fase 1: Se ve si alguna de las aristas borradas era de T o no. Si no existe ninguna, se ignoran las aristas de G-T en las siguientes tres fases. (Usando el lema 8.3.2)

-Fase 2: Se elige un líder para cada componente de $T - A_1$

-Fase 3: Se encuentra las aristas de partición y se cambia su estado al de aristas que no están en el árbol. En esta fase las aristas de partición se encuentran en cada componente de la siguiente forma:

3.1: Marcar los nodos que sean branch o incidentes en una forma convergente de las hojas a la raiz.
i) Cada hoja decide si es un procesador incidente o no (viendo si es incidente al menos a una arista adicionada o a una eliminada) y envía un mensaje a su padre con esta información.
ii) Cuando un procesador interno recibió mensajes de todos sus hijos, decide si es un procesador marcado o no. Es marcado si y solo si
a) Es un procesador incidente o
b) Tiene dos hijos que le informan que existe un procesador incidente entre sus descendientes.
Luego envía un mensaje a su padre comunicándole si él a su vez tiene algún descendiente marcado o no.

3.2: Un nodo interno marcado, es el extremo 'superior' de los caminos primarios entre él y cada uno de sus descendientes marcados más cercanos.

Para encontrar la arista de partición en un camino primario, el nodo superior envía un mensaje a cada uno de sus hijos que le informaron que tenían un nodo incidente entre sus descendientes. El mensaje, que incluye el máximo peso de una arista en el camino ya recorrido, se retransmite hasta encontrar un nodo marcado que es el otro extremo del camino primario.
Cuando el mensaje llega al otro extremo, se envía otro de regreso hasta el primer nodo incidente a la arista de partición (se conoce su peso) para que ese mismo nodo se encargue de cambiarla como arista que ya no está en el árbol.

-Fase 4: Aplicar el algoritmo PMHT a la red G_1 con configuración inicial $T - A_p$

8.3.8 Correctez y complejidades

Lema 8.3.3 *Cada fragmento de $T - A_p$ es un fragmento del nuevo AGM T_1.*

Demostración Por reducción al absurdo, supongamos que existe un fragmento F_1 en $T - A_p$ que no es un fragmento de T_1. Entonces contiene al menos una arista (u,v) que no está en T_1.

Como F_1 es un fragmento de T, (u, v) estaba en T y por tanto u es hijo de v en T o viceversa. Sin pérdida de generalidad, supongamos que v es hijo de u en T.
Al quitar (u, v) de T, T se divide en dos subárboles, uno de ellos tiene como raiz v (denotado como T_v) y el otro es $T - T_v$ (denotado como G_1).

(Caso 1) Existe una arista adicionada entre T_v y G_1, es decir una arista (x, y) tal que $x \in T_v$ y $y \in G_1$ (ver figura 8.10).

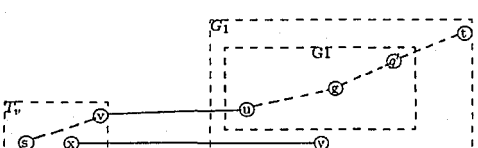


Figura 8.10: (x,y) arista adicionada

Es decir, en este caso, hay un nodo *incidente* a una arista adicionada en T_v (x) y en G_1 (y).

Al ser u antecesor de v, lo es de todos los nodos de T_v, en particular de x. O sea u es antecesor de un nodo *incidente* y como y también es un nodo *incidente*, existe un antecesor de u que es un procesador *marcado* (pues es antecesor de x y y). Sea t el antecesor *marcado* más cercano a u (t puede ser u mismo). Similarmente, sea s el descendiente *marcado* más próximo a v (s puede ser v mismo).

Por la definición de arista de partición, la arista de mayor peso en el camino $t - s$ es una arista de partición, llamémosla (g, g') con g hijo de g' en T. Ya que supusimos que $(u, v) \in T - A_p$ (no es arista de partición), entonces $(g, g') <> (u, v)$ y por tanto (g, g') está en G_1 o en T_v. Sin pérdida de generalidad supongamos que (g, g') está en el camino $u - t$, es decir en la porción del camino $s - t$ incluída en G_1. (ver figura 8.8
Al remover (u, v) y (g, g') de T, T es dividido en tres fragmentos: Una que contiene a u y g (denotada por $G1$), otra que contiene g' y una tercera que es T_v. Como t es el antecesor más cercano a u no existe un procesador marcado en el camino $u - t$ distinto a u y a t; y por tanto no hay ningún procesador marcado en $G1$ lo que implica que no existe una arista adicionada de la forma v_1, v_2 con $v_1 \in G1$. Este hecho implica que para cualquier arista $(z, z1)$ tal que $z \in G1$ y $z1 \in T - G1$ se cumple que $peso(z, z1) > peso(u, v)$ o $peso(z, z1) > peso(g, g')$, es decir que cualquier arista de salida del fragmento $G1$ tiene mayor peso que (u, v) o (g, g') pues estas fueron las que se eligieron como parte de T y al no tener aristas adicionadas en $G1$ esa elección no se modifica. Además, como (g, g') es la arista de mayor peso en el camino $t - s$, se cumple también que $peso(g, g') > peso(u, v)$. Por tanto, $peso(z, z1) > peso(u, v)$ (Aserción 1).

Cuando (u, v) se adiciona al nuevo AGM T_1 (por hipótesis supusimos que no estaba en él), forma un ciclo con el camino $u - v$ en T_1. Necesariamente en ese camino existe una arista (p, q) tal que $p \in G1$ y $q \in T - G1$ y por la (Aserción 1) sabemos que $peso(p, q) > peso(u, v)$ de donde $T_2 = T_1 - (p, q) + (u, v)$ es un árbol generador tal que $peso(T_2) > peso(T_1)$ contradiciendo la suposición de que T_1 era el AGM de R_1.

(Caso 2) No existen aristas adicionadas entre T_v y G_1. Ya que (u, v) es una arista de T se cumple que $peso(p, q) > peso(u, v)$ para cualquier arista entre T_v y G_1. Por tanto $T - (p, q) + (u, v)$ es un árbol generador de R_1 con peso menor al de T_1 contradiciendo la suposición que T_1 es el AGM de R_1. ∎

Teorema 8.3.4 *Si PMHT empieza con r fragmentos, entonces construye el AGM con:*
Complejidad de mensajes $= O(n \log r + e)$
Complejidad en tiempo $= O(n \log r + n)$

Demostración [Esquema] Tomando en cuenta que PMHT es un algoritmo muy similar al de GHS, podemos usar una demostración análoga a la usada para probar las complejidades de dicho algoritmo.

En la Complejidad de mensajes, los términos n y e surgen del mismo razonamiento aplicado en GHS. Lo que se modifica aquí es el término $\log r$ (en GHS teníamos $\log n$) que corresponde al

número de fases usadas por el algoritmo. Si vemos los r fragmentos como los r (super)vértices iniciales, entonces se encontrará el AGM de toda la red en $\log r$ fases.

La demostración para la complejidad del tiempo se hace siguiendo la misma idea. ■

Teorema 8.3.5 *El número de fragmentos en $T - A_p$ es $O(t + f)$ donde t es el número de aristas adicionadas y f es el número de aristas de T eliminadas*

Demostración Construímos la supergráfica $G1 = (V1, E1)$ donde
$V1 = \{u \text{ tal que } F_u \text{ es fragmento de } T - A_p\}$ (el conjunto de fragmentos que quedan luego de haber eliminado las aristas de partición) y
$E1 = \{(u, v) \text{ tal que existe una arista de particion } (s, t) \text{ y } s \in F_u, t \in F_v\}$. Es decir que s (respectivamente t) pertenezca al fragmento F_u (respectivamente F_v) en $T - A_p$.

G1 así definida tiene las siguientes características:

a) G1 es un árbol. Obviamente es conexa pues si dos nodos $v_1, v_2 \in V1$ es porque existía una arista de partición en la gráfica original R que unía a dos nodos tales que $s \in F_{v_1}$ y $t \in F_{v_2}$, y por la definición de G1, $(v_1, v_2) \in E1$. No contiene ciclos pues si contendría uno, las aristas de E1 que lo formarían corresponderían a un ciclo en R lo que no puede suceder porque $A_p \subset T$.

b) Cada fragmento correspondiente a un nodo de V1 contiene al menos un nodo marcado pues al ser un fragmento de $T - A_p$ debe tener al menos un nodo extremo de un camino primario.

c) Cada fragmento correspondiente a un nodo de G1 que es hoja contiene un nodo marcado que es un nodo incidente. Si hubiera un nodo branch en un fragmento (X) correspondiente a una hoja, este nodo tendría al menos dos descendientes marcados (en la gráfica original) y por tanto sería el extremo de al menos dos caminos primarios en cada uno de los cuales habría una arista de partición lo que implicaría que X correspondería a un nodo con dos hijos y las hojas no tienen hijos.

d) Cada fragmento correspondiente a un nodo interno en G1 tiene al menos dos hijos. El razonamiento que lleva a esta conclusión es equivalente al expuesto en c).

Habiendo f aristas borradas y t aristas adicionadas en R, el número de hojas en G1 es a lo sumo $2(t + f)$ (por c)). Por otro lado d) implica que si hay h hojas, el número de nodos internos es a lo sumo h-1.

De donde:

$$\begin{aligned} &\text{Número de fragmentos en } T - A_p = |V1| \\ &= \text{números de nodos internos en G1} + \text{número de hojas en G1} \\ &\leq 4(t + f) = O(t + f) \end{aligned}$$ ■

Teorema 8.3.6 Correctez. *El algoritmo AP-PMHT es correcto.*

Demostración Las fases 1, 2 y 3 terminan en un tiempo finito y las aristas de partición encontradas en la fase 3 son las correctas. Además, por el lema 8.3.3 cada componente conexa en $T - A_p$ es un fragmento del nuevo AGM entonces podemos aplicar PMHT (fase 4) sobre esas componentes para construir el nuevo AGM en tiempo finito. ■

Teorema 8.3.7 La complejidad de mensajes *del algoritmo es* $O(n \log(t+f) + m)$

Demostración Fase 1: Cada nodo debe ser notificado sobre si existen o no aristas del árbol T que hayan sido borradas. Si no existen, la comunicación puede hacerse por las aristas de T exclusivamente y por tanto la complejidad de mensajes de esta etapa será O(n). Si existe al menos una arista de T eliminada, entonces se debe usar otras aristas para la notificación y por tanto la complejidad de mensajes será O(e).
Fase 2: En un fragmento de f nodos se envían O(f) mensajes de las hojas hacia la raiz (ver algoritmo del capítulo 4, por ejemplo) para proclamarla como líder y en total el número de nodos en todos los fragmentos es n por lo que la complejidad de mensajes de la fase 2 es O(n).
Fase 3: En esta fase, se envía un número constante de mensajes a través de las aristas de T que no han sido eliminadas por tanto la complejidad de mensajes de esta fase es también O(n).
Fase 4: Del teorema 8.3.5, existen O(t + f) fragmentos al inicio de la fase 4. Usando el teorema 8.3.4, la complejidad de mensajes de la fase 4 es $O(n \log(t+f) + e)$, si $f <> 0$. Si $f = 0$, por el lema 8.3.2 se ignoran todas las aristas de R-T y no se envía mensajes a través de ellas. Entonces, en este caso la complejidad de mensajes es $O(n \log t + (n+t))$. ■

Teorema 8.3.8 La complejidad en tiempo *del algoritmo es* $O(n \log(t+f) + n)$

Demostración La complejidad de tiempo de las fases 1,2 y 3 es de $O(n)$.

Usando los teoremas 8.3.4 y 8.3.5 deducimos que la complejidad de tiempo de la fase 4 es $O(n \log(t+f) + n)$. ■

8.3.9 Extensión $(AP - PMHT)^1$: Adición y eliminación de nodos

Para obtener un algoritmo más general que soporte además la adición y eliminación de nodos, solo se necesita ampliar la definición de procesador incidente que ya se tenía de la siguiente manera:

Un procesador es *incidente* si es incidente a una arista adicionada o eliminada o bien es adyacente a un procesador adicionado o eliminado.

De manera que cuando un procesador, con grado g, es adicionado o eliminado causa a lo sumo g procesadores incidentes.

El resto del algoritmo AP-PMHT queda exactamente igual.

Las complejidades del algoritmo ampliado $(AP - PMHT)^1$ se modifican levemente a:

Complejidad en mensajes de $(AP - PMHT)^1 = O(n_1 \log(g + t + f) + r)$

Complejidad en tiempo de $(AP - PMHT)^1 = O(n_1 \log(g + t + f) + n_1)$

donde:

n_1 = número de nodos de R_1

$$r = \begin{cases} \text{n+g+t} & \text{si no existe procesadores o aristas borradas} \\ e_1\ (No.\ aristas\ de\ R_1) & \text{en otro caso} \end{cases}$$

g=suma de los grados de los procesadores adicionados o eliminados.

Conclusiones

Como se senaló desde un inicio, el propósito central de la presente tesis fue estudiar profunda y exhaustivamente el algoritmo para hallar árboles generadores mínimos propuesto por Gallager, Humblet y Spira. Creemos que hemos logrado, en buena medida, nuestro objetivo lo que nos permite sacar ciertas conclusiones sobre tal algoritmo.

A riesgo de sonar repetitivos, podemos anotar que a lo largo del desarrollo del trabajo, confirmamos nuestra idea de que el algoritmo GHS es verdaderamente interesante y significativo en el área de los Algoritmos Distribuídos, tanto por la forma en que resuelve el problema específico para el que fue diseñado como por las ideas subyacentes en su construcción. Las mejoras que han sido realizadas en posteriores trabajos no modifican sustancialmente las ideas básicas del algoritmo, simplemente emplean ciertos controles adicionales sobre el crecimiento de los fragmentos para mejorar el tiempo de ejecución o hacerlo tolerante a fallas.

A pesar de todo lo anterior, el algoritmo GHS tiene un punto débil y es que no es adaptativo. Constatamos que su comportamiento no se beneficia del hecho de que se tenga una gráfica muy sencilla, aun cuando ésta sea un árbol y, por tanto, el propio AGM.

Adicionalmente creemos que el algoritmo de Gallager, Humblet y Spira sirve para ejemplificar las ventajas y mejoras que se pueden obtener al proponer soluciones distribuídas en lugar de centralizadas a problemas no planteados en principio como distribuídos. Es muy posible que se requiera mayor cuidado y más controles para llegar a una solución correcta y eficiente partiendo de subsoluciones -o soluciones 'locales'- pero los resultados ameritan este esfuerzo adicional.

Otro aspecto que el algoritmo permite resaltar es la posibilidad de usar la -ya tan extendida- Teoría de Gráficas en el contexto de Redes de Computadoras al poder modelar estas últimas con gráficas. Contar con el marco teórico que nos facilite la construcción y evaluación de los algoritmos para el manejo de la comunicación sobre una red es de vital importancia, como en cualquier área de la computación, pero particularmente en ésta donde determinar si un algoritmo es correcto y produce siempre los resultados esperados es una tarea complicada dado que una determinada ejecución de un algoritmo dependerá de una serie de parámetros y circunstancias, cuyas posibles combinaciones son imposibles de conocer a priori.

En el aspecto personal, considero una gran experiencia haber trabajado en un problema tan rico en material de estudio. Aprendí, y en eso tuvo que ver mucho mi tutor, que a veces es mejor no seguir el impulso de abarcar mucho y hacer de nuestra tesis de maestría el trabajo más importante de nuestra vida, sino enfocarnos a un estudio más modesto pero, en la medida de nuestras posibilidades, "sacarle todo el jugo" como decimos los bolivianos.

Pasos Futuros

Hubiera sido interesante contar con el tiempo suficiente para desarrollar un nuevo algoritmo que posiblemente en el peor caso no sea tan bueno como GHS pero que se comporte mejor que aquel sobre gráficas sencillas, en particular gráficas similares a un árbol donde el criterio de 'similaridad' tendría que definirse cuidadosamente (por ejemplo, el número de ciclos independientes en la gráfica). Esta idea quedó como inquietud personal y tal vez en un futuro podamos desarrollarla.

BIBLIOGRAFIA

[AHO] Aho, A. V.; Hopcroft J. E. y Ullman, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[AFE] Afek, Y.; Gafni, A. *Time and Message Bounds for election syncronous and asyncronous in complete networks.* ACM, Symposium on Principles of Distributed Computing. Minaki, Canada, Agosto 1987.

[AWE] Awerbuch, Baruch. *Optimal Distributed Algorithms for Minimun-Weight Spanning Tree, Counting, Leader Election ans related Problems.* Proceedings of the 19th. ACM Symposium on Theory of Computing, 1987.

[AWR] Awerbuch, Baruch. *Complexity of Network Synchronization.* Journal of the Association for Computing Machinery, Vol. 2, No. 4. Octubre 1985.

[BUS] Busacker,Robert. y Saaty, Thomas. *Finite Graphs and Networks.* Mc Graw - Hill. 1965.

[EVE] Even, Shimon. *Graph algorithms.* Potomac, Md: Computer Science. 1989.

[GAF] Gafni, P. *Improvements in time complexities of two message-optimal algorithms.* Proceedings of PODC Conference. Minaki, Ontario, Canadá. Octubre 1985.

[GAR] Garey R., Michael; Johnson, David. *Computers and intractability.* W H Freeman and Company, New York.

[GHS] Gallager R. G.; Humblet P. A. y Spira P. M. *A Distributed Algorithm for Minimun-Weight Spanning Tree.* ACM Transactions on Programming Languages and Systems. Vol. 5, No. 1, Enero 1983, Pág. 66-77.

[GKP] Garay, Juan; Kutten, Shay y Peleg, David. *A Sub-Linear Time Distributed Algorithm for Minimun Spanning Trees.* Proccedings of 34th. IEEE Symposium on Fundations of Computer Science (FOCS). Noviembre, 1993

[HAR] Harary,Frank. *Graph Theory.* Addison-Wesley.

[JOH] Johnsonbaugh, Richard. *Matematicas Discretas.* Grupo Editorial Iberoamericana, 1988.

[KOR] Korach, E.; Kutten, S. y Moran, S. *A Modular Tecnique for the design or efficient distributed Leader Finding Algorithms.* ACM Symposium Principles of Distributed Computing. Minaki, Ontario, Canadá, Agosto 1985 (Versión revisada julio, 1989).

[LAM] Lamport, Leslie. *Time, Clocks and the Ordering of Events in a Distributed System.* Communications of the ACM, Vol. 21, No. 7. Julio, 1978.

[LYN] Lynch, Nancy y Goldman, Kenneth. *Distributed Algorithms.* Lectures Notes for 6.852 (MIT), Fall 1988.

[PAR] Park, Jungho; Masuzawa, Toshimitsu; Hagihora, Ken'chi y Tokura, Nobuki. *Distributed Algorithms for Reconstructing MST after Topology Change.* Faculty of Engineering Science, Osaka University.

[SEG] Segall, Adrian. *Distributed Networks Protocols.* IEEE Transaction on Information Theory. Vol II, N0. 1, Enero 1983.

[WEL] Welch, J. L. *Simulating Synchronous Processors.* Information and Computation. Agosto, 1985.