QÜESTIIÓ, Vol 20, 3 pp. 485-501, 1996

UNA VARIANTE DEL ALGORITMO DE AHUJA-ORLIN PARA PROBLEMAS DE FLUJO MÁXIMO: EXPERIENCIAS COMPUTACIONALES Y COMPARACIONES

SEDEÑO NODA, A.A.*

GONZÁLEZ MARTÍN, C.*

Universidad de La Laguna

En este trabajo se introduce una variante del algoritmo de escalado de Ahuja y Orlin, con la misma complejidad computacional teórica, para resolver problemas de flujo máximo en redes sin circuitos. Como se constata en las experiencias computacionales que hemos realizado sobre problemas generados aleatoriamente, en el noventa por ciento de los casos el tiempo de CPU del nuevo procedimiento es significativamente inferior.

A variant of Ahuja-Orlin's algorithm for maximum flow problems: comparison and computational experiences

Keywords: Problema de flujo máximo, complejidad computacional, etiqueta distancia, experiencias computacionales.

* Departamento de Estadística, Investigación Operativa y Computación (DEIOC). Universidad de La Laguna. 38271 La Laguna, Tenerife (España).

- Article rebut el setembre de 1995.
- Acceptat el juliol de 1996.

1. INTRODUCCIÓN

El problema de flujo máximo es uno de los problemas más relevantes del análisis de redes y ha sido estudiado exhaustivamente (ver, por ejemplo, Ahuja, Magnanti y Orlin (1989b, 1993), Nicoloso y Simeone (1992), Murty (1992), Nemhauser y Wolsey (1988),...). Fue introducido por Fulkerson y Dantzig (1955) y resuelto por Ford y Fulkerson (1956) mediante su conocido algoritmo de caminos incrementales.

A partir de entonces, se han desarrollado un gran número de procedimientos que, a pesar de la eficiencia práctica del algoritmo anterior, tienden a mejorar la complejidad computacional teórica. Dichos procedimientos aportan nuevas ideas, entre las cuales caben destacarse dos: el concepto de *caminos incrementales mínimos* y el de *preflujo*. Un algoritmo que utiliza ambas ideas es el de Ahuja y Orlin (1989a), incorporando además un procedimiento de escala del exceso de flujo. En este trabajo mostramos que la herramienta etiqueta distancia utilizada en este algoritmo conduce, dependiendo de la morfología de la red, a un comportamiento empírico cercano al estimado para la complejidad computacional del caso peor. Esto implica, en muchos casos, un mayor consumo en el tiempo de ejecución de éste frente a los algoritmos clásicos de Ford y Fulkerson (1956), Dinic (1970), Edmonds y Karp (1972), Malhotra, Kumar y Maheshwari (1978).

Por estas razones, hemos centrado nuestro interés en estudiar algunas modificaciones del procedimiento de Ahuja y Orlin, en la pretensión de conseguir mejoras en su eficiencia práctica. En este trabajo estudiamos una de esas variantes para los casos de redes sin circuitos. Las redes sin circuitos aparecen en multitud de aplicaciones como, por ejemplo, los problemas de flujos en redes dinámicas, problemas de planificación con máquinas paralelas (especializadas o no especializadas), sistemas de abastecimientos de aguas donde la fuerza motriz del bien que circula es de origen natural (gravedad), redes de telecomunicación jerarquizadas, una vez establecido el sentido de la transmisión de datos (origen-destino), etc...

Hacemos énfasis en un análisis comparativo del comportamiento empírico de este nuevo algoritmo frente al procedimiento introducido previamente.

2. FORMALIZACIÓN DEL PROBLEMA Y CONCEPTOS BÁSICOS

Dada una red dirigida conexa y sin circuitos $N = (V,A)$, con $V = \{1,\ldots,n\}$ el conjunto de vértices y $A = \{(i,j)/i \in V', j \in V'', i \neq j, V', V'' \subseteq V\}$ el conjunto de arcos, denotaremos por n el cardinal de V y por m el cardinal de A. Sean $s,t \in V$, dos vértices distinguidos de la red denominados respectivamente, *fuente* y *sumidero*. Para cualquier vértice i definimos $\text{Pred}(i) = \{j \in V/(j,i) \in A\}$, al conjunto de vértices

predecesores de i, y $\mathrm{Suc}(i) = \{j \in V/(i,j) \in A\}$, al conjunto de vértices sucesores de i. Asociado con cada arco $(i,j) \in A$ se tiene una capacidad u_{ij}, que indica el límite superior de la cantidad de flujo que puede soportar el arco y l_{ij} la menor cantidad de flujo que debe pasar necesariamente por el mismo arco. Sin pérdida de generalidad, podemos suponer que l_{ij} es igual a cero para cada arco. Denotaremos por U a $\max\{u_{ij}/(i,j) \in A\}$. En el problema de Flujo Máximo se desea enviar la mayor cantidad de flujo de la fuente al sumidero satisfaciendo las restricciones de capacidad y las de conservación de flujo en los nodos. En lenguaje matemático, este problema se puede plantear en los siguientes términos:

$$\max f$$

$$\text{(1)} \qquad \text{sujeto a:} \quad \sum_{j \in \mathrm{Suc}(i)} x_{ij} - \sum_{j \in \mathrm{Pred}(i)} x_{ji} = \begin{cases} f & \text{si } i = s \\ 0 & i \in V - \{s,t\} \\ -f & \text{si } i = t \end{cases}$$

$$\text{(2)} \qquad 0 \leqslant x_{ij} \leqslant u_{ij}, \quad (i,j) \in A$$

Un flujo factible en N es un vector $x = \{x_{ij}/(i,j) \in A\}$, que satisface (1) y (2), donde x_{ij} denota las unidades de flujo que circulan de i a j sobre el arco (i,j), y f es el valor del flujo de s a t.

Un *preflujo* x es una función $x : A \longrightarrow \mathbb{R}^+$ satisface (2) y la siguiente relajación de (1):

$$e(i) = \sum_{j \in \mathrm{Pred}(i)} x_{ji} - \sum_{j \in \mathrm{Suc}(i)} x_{ij} \geqslant 0, \quad i \in V - \{s,t\}$$

Dado un preflujo x, se define, para cada $i \in V - \{s,t\}$, $e(i)$ como el exceso del nodo i. Un nodo con exceso positivo es llamado nodo *activo*.

Dado cualquier par de nodos $i, j \in V$ definimos la *capacidad residual* del arco (i,j) con respecto a un preflujo x, como $r_{ij} = u_{ij} - x_{ij} + x_{ji}$ (debemos entender que si $(i,j) \neq A$, entonces $u_{ij} = 0$). La capacidad residual de (i,j) representa la cantidad máxima de flujo adicional que puede ser enviada desde el nodo i al j usando los arcos (i,j) y (j,i) de A (si existen ambos). Llamaremos *red residual* o *incremental* a aquella que consiste únicamente de los arcos con capacidades residuales positivas.

Para cada nodo i, definimos $A(i)$ como el conjunto de arcos de la red residual que salen de i.

Una función distancia $d : V \longrightarrow \mathbb{Z}^+$ para un preflujo x es una función del conjunto de nodos a los enteros no negativos. Diremos que d es *válida*, si satisface las dos condiciones siguientes:

a) $d(t) = 0$

b) $d(i) \leqslant d(j) + 1 \quad \forall (i,j) \in A,\ r_{ij} > 0$

Por inducción se demuestra que $d(i)$ es una cota inferior de la longitud del camino más corto de i a t en la red residual, entendiendo por longitud de un camino el número de arcos del mismo. Si para cada i, la etiqueta distancia $d(i)$ es igual a la longitud del camino de longitud mínima entre i y t en la red residual, entonces la denominaremos etiqueta distancia *exacta*.

Un arco (i,j) en la red residual es llamado *admisible* si satisface que $d(i) = d(j) + 1$.

3. ALGORITMO DE AHUJA Y ORLIN

En cada paso, este algoritmo mantiene un preflujo y envía flujo hacia el sumidero, desde los nodos más cercanos con exceso positivo, a través de arcos admisibles. Para estimar qué nodo activo está más cerca del sumidero y qué arcos de los que salen de éste son admisibles, se utiliza la etiqueta distancia. Este procedimiento usa un escalado del exceso. La idea básica consiste en enviar flujo desde nodos activos con exceso suficientemente grande a nodos con excesos suficientemente pequeños, sin que los excesos de estos últimos se hagan demasiado grandes. El algoritmo recibe por ello el nombre de *algoritmo de escala del exceso*. El algoritmo realiza $K = [\log U] + 1$ iteraciones de escalado. Para cada iteración de escalado, se define el *exceso dominante* como el menor entero Δ potencia de dos que satisface que $e(i) \leqslant \Delta,\ \forall i \in \mathbb{N},\ i \neq s,t$. En cada iteración de escalado se consideran nodos con exceso mayor que $\Delta/2$; y entre todos estos, se seleccionan uno de menor etiqueta distancia. Una vez seleccionado, se envia un flujo igual a $\delta = \min\{e(i),\ r_{ij},\ \Delta - e(j)\}$, si el arco (i,j) es admisible y se actualizan convenientemente los excesos de los nodos i y j. Un envío es *saturante* si $\delta = r_{ij}$, es decir, si la cantidad de flujo enviado a través del arco (i,j) coincide con su residuo; en otro caso, es un envío *no saturante*. Una iteración de escalado termina cuando no hay nodos con excesos mayores que $\Delta/2$ y una nueva iteración empieza con Δ igual a $\Delta/2$. Después de $\log U$ iteraciones de escalado todos los nodos tendrán un exceso igual a cero y obtendremos un flujo máximo. Para seleccionar un nodo activo con un exceso mayor que $\Delta/2$ y con la menor etiqueta distancia, se mantienen unas listas o colas $L(r) = \{i \in \mathbb{N} : e(i) > \Delta/2 \text{ y } d(i) = r\}$ para $r = 1, \ldots, 2n-1$. Un esquema del algoritmo es el siguiente:

Algoritmo de Ahuja-Orlin (1989a)

begin

Para cada arco $(s, j) \in A \Rightarrow x_{sj} = u_{sj}$, $x_{ij} = 0$ el resto;

$d(t) := 0$, determina las etiquetas distancias iniciales $d(i)$ mediante un recorrido en amplitud desde el nodo t; $d(s) := n$;

$K = [\log U] + 1$;

for $k = 1$ to K do (bucle logarítmico)

begin

$\Delta := 2^{K-k}$;

for cada nodo do $i \in V$ **do if** $e(i) > \Delta/2$ **then** Suma i a $L(d(i))$;

level:= 1;

while *level* $< 2n$ **do**

begin

if L *(level)* = $\emptyset$ **then** *level*:= *level* + 1

else

begin

Selecciona un nodo $i \in L$ *(level)*;

if hay un arco admisible $(i, j) \in A(i)$ **then**

begin

Envía $\delta = \min\{e(i), r_{ij}, \Delta - e(j)\}$ de i a j;

Actualiza las capacidades residuales y $e(i)$, $e(j)$;

if $e(i) \leqslant \Delta/2$ **then** Borra i de L *(Level)*;

if $(e(j) > \Delta/2)$ **and** $(j <> s,t)$ **then**

Suma j a L *(Level*-1*)*, *Level*:=*Level*-1;

end

else

begin

Borra i de L *(Level)*;

$d(i) = \min\{d(j) + 1 : (i, j) \in A(i)$ y $r_{ij} > 0\}$

Suma i a $L(d(i))$;

end

end

end

end

end;

La Complejidad del algoritmo de Ahuja y Orlin es $O(nm + n^2 \log U)$, donde $n^2 \log U$ es el número de envíos no saturantes y nm es el número de envíos saturantes. El número de veces que se actualizan las etiquetas distancias de los nodos es del orden de $O(n^2)$.

En la experiencia práctica sobre problemas generados aleatoriamente, se constata un comportamiento dispar respecto del tiempo de ejecución de este algoritmo. De-

pendiendo del problema particular, este puede ser resuelto de forma rápida o emplear un número de iteraciones cercana a las estimadas para el peor caso. Esto es así ya que, para el primer caso, el número de actualizaciones de la etiqueta distancia es $O(n)$, mientras que en el segundo es del orden de $O(n^2)$. Conviene recordar que cada vez que se necesita actualizar la etiqueta distancia de un nodo i, es porque este es un nodo activo y se debe disminuir su exceso de flujo. Este exceso de flujo es contribución del envío de flujo a través de arcos que habrán sido saturados o no. El tener que deshacerse de este exceso implica el envío de nuevos flujos a través de arcos que emanen de i, lo que implica un incremento obvio en el número de envíos saturantes y no saturantes. Si la etiqueta distancia de un nodo necesita ser actualizada $O(n)$ veces, es por que se tiene que enviar flujo desde este nodo $O(n)$ veces. Si esto ocurre para $O(n)$ nodos, el algoritmo necesita un número de iteraciones cercana a la de su complejidad teórica. Dicho de otro modo, es fácil que este algoritmo realice $O(nm + n^2 \log U)$ iteraciones incluso cuando la red de partida no sea complicada. Además, incluso habiendo encontrado el valor del flujo máximo, se necesitan realizar un número de iteraciones adicionales para eliminar los excesos de los nodos que no alcanzan a t.

Estimamos que este comportamiento es debido al uso de la herramienta etiqueta distancia, cuya forma de actuar muestra una dependencia clara con la morfología de la red. Dicha herramienta hace imprevisible el comportamiento de estos algoritmos frente a varios problemas con la misma especificación de los parámetros que lo definen, es decir, frente al número de nodos, número de arcos y capacidad máxima. Un análisis detallado de estas consideraciones aparece en González Martín, González Sierra y Sedeño Noda (1995).

A continuación se muestra la tabla 1 en la que se aprecia el comportamiento del algoritmo de Ahuja y Orlin para problemas particulares con la misma especificación.

Tabla 1

Valores de Ret y CPU para el algoritmo de Ahuja y Orlin

n	m	U	RET	CPU	n	m	U	RET	CPU
100	2000	32000	9493	1.17	100	2000	32000	38	0,02
100	2000	1E+09	9357	1.24	100	2000	1E+09	41	0,02
100	2000	2E+09	9372	1.10	100	2000	2E+09	67	0,03
200	7000	32000	38672	7.45	200	7000	32000	60	0,03
200	7000	1E+09	38777	6.82	200	7000	1E+09	68	0,05
200	7000	2E+09	38484	8.90	200	7000	2E+09	51	0,04
300	20000	32000	88122	23.55	300	20000	32000	95	0,05
300	20000	1E+09	88163	25.12	300	20000	1E+09	125	0,12
300	20000	2E+09	87875	26.7	300	20000	2E+09	131	0.12
400	50000	32000	157701	70	400	50000	32000	208	0.22
400	50000	1E+09	157859	71.88	400	50000	1E+09	218	0.03
400	50000	2E+09	157356	73	400	50000	2E+09	266	0.35

El algoritmo ha sido instrumentado en Pascal estándar y ha sido ejecutado sobre una estación de trabajo HP9000 serie 715/33. Los problemas han sido generados aleatoriamente recibiendo como entrada la especificación de parámetros antes reseñada. En esta tabla se muestra el número de nodos (n), de aristas (m) y el mayor valor de la capacidad (U) de la red generada aleatoriamente y, a continuación, el número de actualizaciones de la etiqueta distancia (RET) y el tiempo empleado por el algoritmo en segundos (CPU). Se ve claramente que para la misma especificación, el algoritmo se comporta de manera diferente.

Ahuja y Orlin (1989a) en la presentación de su algoritmo reconocen que, en la práctica, un cuello de botella potencial de éste es el número de actualizaciones de la etiqueta distancia. En particular, el algoritmo identifica que la red residual no contiene ningún camino del nodo i a t sólo cuando $d(i)$ es mayor que $n-2$. Evidentemente un nodo desconectado del sumidero con un exceso positivo, debe enviar este exceso de vuelta a la fuente. Pero para que esto ocurra, la etiqueta distancia de este nodo debe ser incrementada hasta un valor mayor que $n-2$ y, en el peor de los casos, se incrementa en una unidad cada vez. Lo ideal sería que la etiqueta distancia de un nodo activo no fuera actualizada mientras sea posible enviar flujo desde este nodo hacia el sumidero, es decir, mientras t sea alcanzable desde este nodo. Dicho de otro modo, se hace necesario relajar la condición de que para enviar flujo a través de un arco (i, j) las etiquetas distancias sean $d(i) = d(j) + 1$ y $r_{ij} > 0$, de tal manera que sólo sea necesario actualizar la etiqueta distancia de un nodo cuando su exceso sea positivo y este nodo no alcance a t. El problema es que para poder relajar esta condición, los valores de las etiquetas distancias deben ser los adecuados. Una posible solución es que para todo nodo conectado al sumidero el valor de $d(i)$ sea el número de arcos del camino más largo de i a t en la red incremental. Así, la condición se cambiaría por $d(i) > d(j)$ y $r_{ij} > 0$ y nos evitaríamos tener que actualizar la etiqueta $d(i)$ hasta que $d(i) = d(j) + 1$. La cuestión es que para poder realizarlo, debemos ser capaces de ordenar el conjunto de nodos de acuerdo con la cercanía de estos al sumidero y este orden sólo se consigue de manera total si la red no contiene circuitos.

4. VARIANTE DEL ALGORITMO DE AHUJA Y ORLIN PARA REDES SIN CIRCUITOS

Si la red no contiene circuitos, es posible ordenar el conjunto de nodos de acuerdo con la cercanía de estos a t. La función *orden*, ord : $N \longrightarrow \mathbb{Z}^+$, es una función definida en el conjunto de nodos y que toma valores en los enteros no negativos tal que ord(t) = 0 y ord(i) es el número de arcos del camino más largo de i a t. Los valores de esta función pueden calcularse en $O(m)$ mediante un recorrido en profundidad, empezando en el nodo t. Obviamente si N contiene circuitos no es posible tal orden. Además, un $i \in \mathbb{N}$, $i \neq t$ tal que ord(i) = 0 implica que no hay camino de i a t. Proponemos utilizar la función orden en lugar de la función

etiqueta distancia permitiendo enviar flujo desde un nodo i a un nodo j siempre que $\text{ord}(i) > \text{ord}(j)$. En cada etapa consideraremos el nodo activo i de menor $\text{ord}(i)$ y el arco (i,j) de $A(i)$ tal que j sea el de menor etiqueta distancia. De esta manera solo tendremos que actualizar la etiqueta distancia de un nodo activo cuando haya que devolver este exceso hacia la fuente. El algoritmo se desarrolla según el siguiente esquema:

Algoritmo para el caso de N sin circuitos

begin

$d(t) := 0, d(s) := n$; determina el orden $\text{ord}(i)$ $\forall i \neq s,t$ mediante un recorrido en profundidad desde el nodo t;

$d(i) = \text{ord}(i)$;

Para cada arco $(s,j) \in A : d(j) > 0$ *or* $j = t \Rightarrow x_{sj} = u_{sj}, x_{ij} = 0$ el resto;

$K = [\log U] + 1$;

for $k = 1$ to K **do** (bucle logarítmico)

begin

$\Delta := 2^{K-k}$

for cada nodo $i \in V$ **do if** $e(i) > \Delta/2$ **then** Suma i a $L(d(i))$;

level:= 1;

while *level* $< 2n$ **do**

begin

if L *(level)* = $\emptyset$ **then** *level*:= *level* + 1

else

begin

Selecciona un nodo $i \in L$ *(level)*;

if $\exists j : d(j) = \min_{p \in \text{suc}(i)} \{d(p) : d(i) > d(p)$ y $(d(p) > 0$ o $p = t)\}$

and $r_{ij} > 0$ **then**

begin

Envía $\delta = \min\{e(i), r_{ij}, \Delta - e(j)\}$ **de** i **a** j**;**

Actualiza las capacidades residuales y $e(i), e(j)$**;**

if $e(i) \leqslant \Delta/2$ **then Borra** i **de** *L (Level)***;**

if $(e(j) > \Delta/2)$ **and** $(j <> s,t)$ **then Suma** j **a** $L(d(j))$**,**

Level*:=d*(j)**;**

end

else

begin

Borra i **de** *L (Level)***;**

$d(i) = \min\{d(j)+1 : (i,j) \in A(i)$ **y** $r_{ij} > 0$ **y** $(d(j) >= d(i))\}$

Suma i **a** $L(d(i))$**;**

end

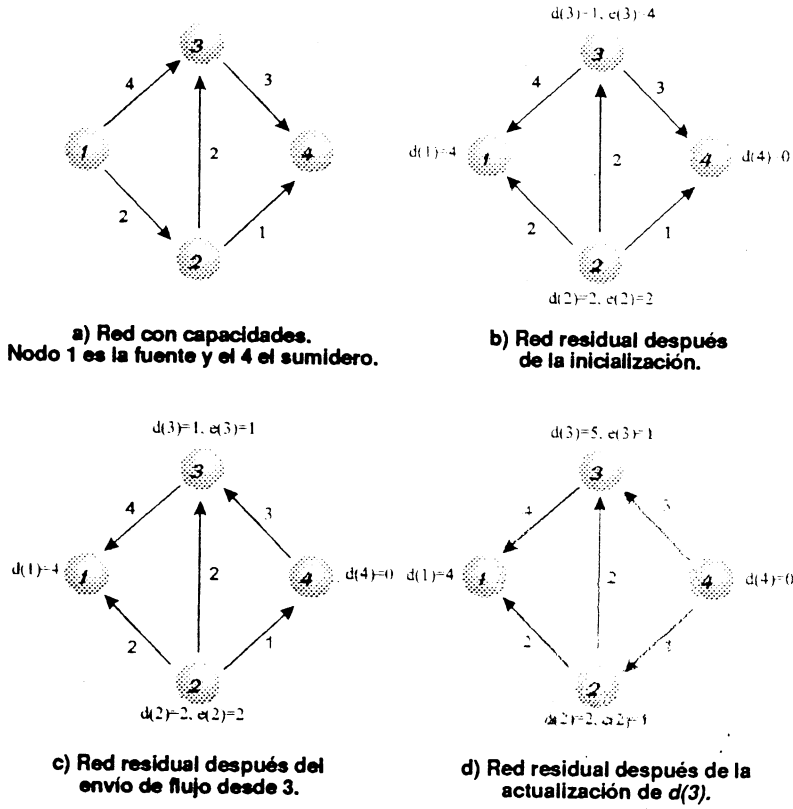
end

end

end

end;

Sólo se actualizará la etiqueta distancia de un nodo cuando no haya camino desde este al sumidero. La figura 1 ilustra los pasos del algoritmo. En La figura 1 d) se puede observar que una vez que el nodo 3 no alcanza a t, su valor de etiqueta distancia es mayor que la del nodo fuente, de tal manera que en el siguiente envío desde este nodo, nos podemos deshacer de su exceso. Además, no se enviará flujo de 2 a 3, a través del arco (2, 3), ya que la etiqueta distancia del 2 es menor que la del 3. Esto es así, ya que el orden en el que se envían flujos es siempre a través de los nodos más cercanos a t, y en este caso 3 está muy alejado de t. En el Algoritmo de Ahuja y Orlin se realizan una serie de pasos equivalentes al de la figura anterior, pero efectuando actualizaciones de la etiqueta distancia adicionales y, por lo tanto, un número mayor de envíos de flujo.



a) Red con capacidades. Nodo 1 es la fuente y el 4 el sumidero.

b) Red residual después de la inicialización.

c) Red residual después del envío de flujo desde 3.

d) Red residual después de la actualización de *d(3)*.

Figura 1. ***Pasos del algoritmo con la nueva etiqueta distancia***

5. ASPECTOS COMPUTACIONALES

La complejidad de este algoritmo es la misma que el de Ahuja y Orlin, es decir, $O(nm+n^2\log U)$. Esto es evidente, ya que si bien la etiqueta distancia $d(i)$ representa la longitud del camino más largo de i a t, el valor de ésta vuelve a estar acotado por $2n$ y cada envío no saturante desde un nodo i a un j es al menos de $\Delta/2$ unidades. Por lo tanto el número de envíos no saturantes vuelve a estar acotado por $8n^2$ (ver Ahuja y Orlin (1989a)). Por otro lado, la condición de enviar flujo de un nodo i a un nodo j es $d(i) > d(j)$, lo cual reduce en la práctica el número de actualizaciones de las etiquetas distancia. Conviene recordar que cada vez que se actualizaba la etiqueta distancia de algún nodo en el algoritmo de Ahuja y Orlin era porque los arcos que salían de este nodo habían sido saturados. Además, por ser este nodo activo, necesita realizar nuevos envíos para deshacerse de sus excesos. Por ello, disminuir el número de actualizaciones de d equivale a disminuir el número de envíos, tanto saturantes como no saturantes. Estos comentarios se pueden constatar en la práctica.

Más adelante se muestran los resultados obtenidos a la hora de resolver una serie de problemas generados aleatoriamente. Los algoritmos han sido instrumentados en Pascal estándar y han sido ejecutados sobre una HP9000 serie 715/33. Como medida del esfuerzo de ejecución de los dos algoritmos (A&O= Ahuja y Orlin y VA&O= variante de Ahuja y Orlin) hemos tomado los siguientes valores: el tiempo de CPU(seg), el número de envíos saturantes (SAT), el número de envíos no saturantes (NSAT) y el número de actualizaciones de las etiquetas distancia (RET).

El estudio práctico se ha llevado a cabo generando aleatoriamente 150 problemas con las siguiente especificaciones para los nodos n, las aristas m y la capacidad máxima U. El rango de los nodos se estableció en 400, 600 y 800. Para cada valor del rango de nodos, el rango de aristas se estableció en $5n, 10n, 15n, 20n, 25n, 30n, 35n, 40n, 45n, 50n$. Para cada valor de n y m, se generaron 5 problemas cuyas capacidades fueron tomadas de forma aleatoria y uniforme en el intervalo $[1, U]$, con U tomando los valores $10^5, 10^6, 10^7, 10^8, 10^9$. De los 150 problemas, en 136 el método denominado VA&O mejora notablemente el tiempo de ejecución necesario para resolver un problema, ya que disminuye el número de envíos no saturantes, saturantes y el número de actualizaciones de la etiqueta distancia. En los 14 restantes ambos métodos son similares.

En la tabla 2, se muestran los tiempos de CPU en segundos, así como los valores de NSAT, SAT y RET promediados según los rangos de variación de las aristas y las capacidades máximas. Se observa que tanto los tiempos de CPU, como el número de envíos no saturantes, saturantes y el número de actualizaciones de la etiqueta distancia es bastante inferior en el método VA&O que en el A&O.

En la tabla 3, se muestran los tiempos de CPU en segundos, así como los valores de NSAT, SAT y RET promediados según los rangos de variación de la relación aristas/nodos,obteniendo así, el comportamiento de los métodos según la densidad de

la red. En esta tabla se vuelven a observar que los resultados obtenidos por el método VA&O mejoran los del A&O.

Tabla 2

Promedio para aristas y capacidad máxima

n	Algoritmo	CPU	NSAT	SAT	RET
400	A&O	19.55	86207	41427	107711
400	VA&O	4.95	19938	9560	24138
600	A&O	44.13	196744	78267	233445
600	VA&O	16.03	57996	26329	69495
800	A&O	89.54	397405	142229	454264
800	VA&O	27.50	103464	42069	118071

Tabla 3

Promedio según la densidad de la red

m/n	Algoritmo	CPU	NSAT	SAT	RET
5	A&O	30,49	253345	71685	247749
5	VA&O	7,56	61443	19184	56463
10	A&O	28,59	207936	63009	219200
10	VA&O	8,38	61496	21430	61962
15	A&O	29,58	186855	66106	206936
15	VA&O	10,56	64631	24578	69001
20	A&O	51,69	267268	95176	303507
20	VA&O	14,12	65010	25137	72555
25	A&O	52,83	245555	98958	290648
25	VA&O	16,53	62170	27285	73990
30	A&O	57,56	243672	94328	287650
30	VA&O	17,87	66006	31665	81735
35	A&O	50,69	196325	76843	240288
35	VA&O	16,28	57101	27725	71494
40	A&O	67,72	232683	102357	299528
40	VA&O	23,40	58414	30867	76662
45	A&O	66,71	191643	98229	249777
45	VA&O	20,81	49720	22178	63242
50	A&O	66,71	191643	98229	249777
50	VA&O	26,09	58667	29808	78574

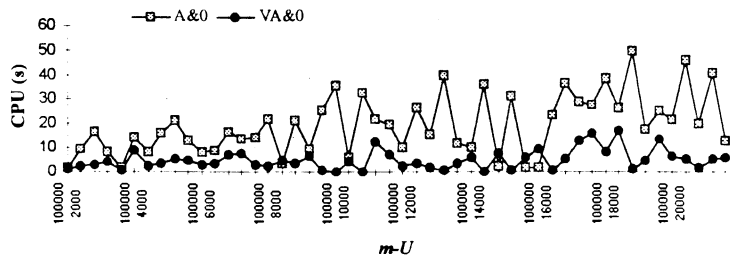


Figura 2. *Tiempos para redes de 400 Nodos*

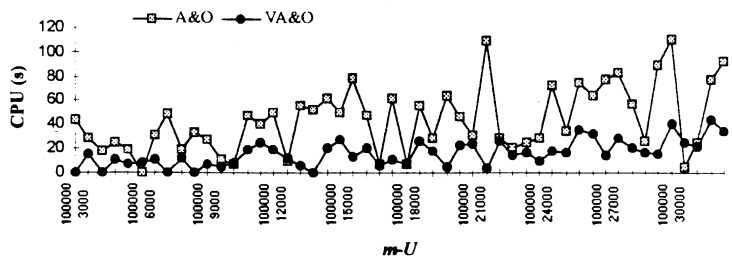


Figura 3. *Tiempos para redes de 600 Nodos*

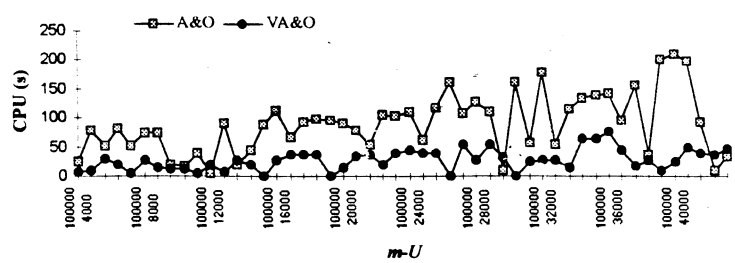


Figura 4. *Tiempos para redes de 800 Nodos*

En las figuras anteriores aparecen unas gráficas del tiempo de CPU, en segundos, empleado por cada algoritmo para cada problema en particular, agrupando según el número de nodos. La figura 2 se refiere a los problemas con 400 nodos, la figura 3 a los de 600 nodos y la figura 4 a los de 800 nodos. En estas figuras, se puede observar que el VA&O emplea menor tiempo de CPU que el A&O en la mayoría de los problemas. En aquellos, en que A&O es más rápido que VA&O, se aprecia que el tiempo de CPU es similar.

6. CONCLUSIONES

El método de Ahuja y Orlin y, en general, aquellos que utilizan la función etiqueta distancia son sensibles a la morfología de la red y, por lo tanto, a la numeración de los nodos. Esto hace que el comportamiento práctico de los mismos precise de un número de iteraciones similar a las de su cota del caso peor, aún cuando las redes no sean complicadas. Esto se pone de manifiesto en la tabla 1 y es tratado de manera rigurosa en el estudio estadístico contenido en González Martín, González Sierra y Sedeño Noda (1995).

El uso de unas etiquetas distancias adecuadas, o dicho de otro modo, el orden de consideración adecuado de los nodos de una red, proporciona una mayor robustez práctica a los métodos mencionados anteriormente. El orden o etiquetado presentado en este trabajo, válido para redes sin circuitos, mejora notablemente el comportamiento empírico de unos de estos algoritmos.

7. BIBLIOGRAFÍA

[1] **Ahuja, R.** and **Orlin, J.B.** (1989a). «A fast and simple algorithm for the maximum flow problem». *Operations Research*, **37**, 748–759.

[2] **Ahuja, R., Magnanti, T.** and **Orlin, J.B.** (1989b).«Network flows». En *Optimizacion. Handbooks in Operations Research and Management Science*. **Vol. 1**, 211–369. G.L. Nemhauser, A.H.G. Rinnooy Kan, M.J. Todd (eds.). North Holland.

[3] **Ahuja, R., Magnanti, T.** and **Orlin, J.B.** (1993). *Network Flows*. Prentice-Hall Inc.

[4] **Dinic, E.A.** (1970). «Algorithms for solution of aproblem of maximum flow in networks with power estimation». *Soviet Mathematical Doklady*, **11**, 1277–1280.

[5] **Edmonds, J.** and **Karp, R.M.** (1972). «Theoretical improvements in algorithmic efficiency of network flow problems». *Journal of ACM*, **19**, 248–264.

[6] **Ford, L.R.** and **Fulkerson, D.R.** (1956). «Maximal flow through a network». *Canadian Journal of Mathematics*, **8**, 399–404.

[7] **Goldberg, A.V.** and **Tarjan, R.E.** (1986). «A new approach to the maximum flow problem». *Proc. 18th ACM Symp. on the Theory of Computation*, 136–146.

[8] **González Martín, C., González Sierra, M.A.** and **Sedeño Noda, A.A.** (1995). *An algorithimic study of the maximum flow problem: a comparative statistical analysis*. Documento de trabajo.

[9] **Karzanov, A.V.** (1974). «Determining the maximal flow in a network by the method of preflows». *Soviet Mathematical Doklady*, **15**, 434–437.

[10] **Malhotra, V.M., Kumar, M.P.** and **Maheshwari, S.N.** (1978). «An $O(|V|^3)$ algorithm for finding maximum flows in networks». *Inform. Process. Lett.*, **7**, 277–278.

[11] **Murty, K.** (1992). *Network programming*. Prentice-Hall, Inc.

[12] **Nemhauser, G.** and **Wolsey, L.** (1988). *Integer and combinatorial optimization*. John Wiley and Sons, Inc.

[13] **Nicoloso, S.** and **Simeone, B.** (1992). *Classical and contemporary methods in network optimizacion*, part I: Network Flows.

ENGLISH SUMMARY

A VARIANT OF AHUJA-ORLIN'S ALGORITHM FOR MAXIMUM FLOW PROBLEMS: COMPARISON AND COMPUTATIONAL EXPERIENCES

SEDEÑO NODA, A.A.*

GONZÁLEZ MARTÍN, C.*

Universidad de La Laguna

In this paper, it is introduced a variant of Ahuja and Orlin's scaled algorithm, with an equivalent theoretical complexity, for solving maximum flow problems on networks without circuits. As it can be deduced from computational experiences made with randomly generated problems, in the ninety per cent of the cases, the CPU time of this new procedure is significantly less.

Keywords: Maximum flow problem; computational complexity; distance label; computational experiments.

* Departamento de Estadística, Investigación Operativa y Computación (DEIOC). Universidad de La Laguna. 38271. La Laguna, Tenerife (España).

- Received september 1995.

- Accepted july 1996.

In this paper, it is introduced a variant of Ahuja and Orlin's scaled algorithm, with an equivalent theoretical complexity, for solving maximum flow problems on networks without circuits. As it can be deduced from computational experiences made with randomly generated problems, in the ninety per cent of the cases, the CPU time of this new procedure is significantly less.

The Ahuja and Orlin's algorithm uses the *shortest augmenting path* idea presented by Edmonds and Karp, and the *Preflow* idea, introduced by Karzanov for solving the maximum flow problem. This algorithm keeps a preflow in each iteration and sends flow to the sink node from the nearest nodes with positive excess through admissible arcs. The distance label is used for estimating which active node is nearer the sink and which adjacent arcs of this one are admissible. This algorithm uses and excess scaled. The basic idea is to send flow from active nodes with excesses enough little, avoiding to create with big excess again. This algorithm is named excess scaled algorithm for this reason. The complexity of Ahuja and Orlin's algorithm is $O(nm+n^2 \log U)$, where $n^2 \log U$ is the number of non saturating sends and nm is the number of saturating sedns. The number of times that the distance labels are updated is $O(n^2)$ (1989a).

We have proved experimentaly that the behavior of this algorithm depends on the network morphology. Sometimes it needs much more time for solving a problem that others where the timeu sed is lower. Then, the algorithm makes a number of iterations near the estimated for the worst case according to the kind of network. We think that this behavior is caused for the use of the distance label tool, which depends clearly on the netsork morphology. This tool makes unforable the behavior of this algorithm with different problems the same specification: number of nodes, number of arcs and maximum capacity. The results are shown in González Martín, González Sierra and Sedeño Noda (1995).

Ahuja and Orlin (1989a), in the presentation of their algorithm, recognize that, in practice, a potential bottle neck is the number of updates of the label distance. Particularly, the algorithm detects that there is not path from node i to node t on the residual network only when the label distance of i is greater than $n-2$. Evidently, a disconnected node from the sink with a positive excess, musts send back it to the source. However, until this happens, the distance label of this node must be incremented until a value greater than $n-2$ and, in the worst case, it is increased in one unit each time. The best way would be that the distance label of an active node keeps its value while it was possible to send flow from this node to the sink; while t was reachable from this node. A solution is that, for each node i connected to the sink, the value of the distance label was the number of arcs of the longest path from i to t on the incremental network. Then, the condition of admissibility would be relaxed for decreasing the number of updates of the distance label. For realizing this, we must to order the set of nodes according to their nearness to the sink, and this order only is possible if the network has not circuits.

In this work we present a modification of the excess scaled algorithm, using a new distance function named *Order*, for networks without circuits. This one leads to an algorithm with the same complexity of Ahuja and Orlin's algorithm but with a practical behavior much better. We introduce also, an experimental comparison of the algorithms on randomly generated problems.