67

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

FLUJO EN REDES. UN ENFOQUE GENERICO

T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I O

P R E S E N T A

JONATHAN J. LOPEZ TELLO

DIRECTOR DE TESIS: M. en I. MARIA DE LUZ GASCA SOTO

DRA. MARÍA DE LOURDES ESTEVA PERALTA
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

Flujo en Redes. Un Enfoque Genérico

realizado por **JONATHAN JESÚS LÓPEZ TELLO**

con número de cuenta **092349720** quien cubrió los créditos de la carrera de:

Actuaría

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	**M. en I. María de Luz Gasca Soto**	
Propietario	**Dr. Abdón Sánchez Arroyo**	
Propietario	**M. en I. Nereo Elías Mata**	
Suplente	**Mat. Adrián Girard Islas**	
Suplente	**Mat. Laura Pastrana Ramírez**	Laura Pastrana R.

Consejo Departamental de Matemáticas

M. en C. José Antonio Flores Díaz

CONSEJO DEPARTAMENTAL DE MATEMÁTICAS

Agradezco antes que nada a Dios por darme una vida llena de bendiciones, de amor y armonía que hoy me permiten culminar este sueño tan largamente anhelado.

Gracias a mis padres, quienes me han forjado férreas convicciones y grandes valores humanos. Por su confianza, por la paciencia, por creer en mi y por darnos siempre, a sus hijos, todo el fruto de su esfuerzo.

Agradezco a mis hermanas, por su cariño y apoyo. Por ser un motivo más para querer ser un mejor ser humano.

Gracias a toda mi familia, por estar unida y por demostrarme en todo momento su cariño y respeto; por hacerme saber que siempre estarán a mi lado apoyándome, aconsejándome y alentándome.

Agradezco a Jennifer, por todo su amor, ternura y comprensión. Por acompañarme no sólo en los momentos más felices sino también en los más difíciles, siendo su cariño el mejor refugio donde poder tomar fuerzas para poder empezar de nuevo.

Gracias a Lucy Gasca, por impulsar continuamente mi esfuerzo. Por su paciencia; por haber hecho posible la realización de este trabajo.

Refiero mi agradecimiento a la UNAM y en especial a la Facultad de Ciencias por haberme dado una muy buena formación profesional.

Para todos ustedes, dedico este logro.

Muchas gracias.

Índice

I. Introducción

En la vida cotidiana las redes están presentes por todas partes. Redes eléctricas nos proporcionan luz en nuestras casas. Redes telefónicas nos permiten comunicar no sólo en nuestras comunidades locales, sino en todo el país y a cualquier parte del mundo. La red de carreteras, la red del metro, así como las redes de servicios de aerolíneas nos proveen la manera de viajar de un lugar a otro. Las redes de computadoras han cambiado la forma en que compartimos la información, la manera en que dirigimos negocios e incluso nuestra vida personal.

En algunos de estos ejemplos, y muchos mas, deseamos mover alguna entidad (como electricidad, señales, personas, vehículos, productos de consumo, datos, mensajes etcétera.) de un punto a otro y queremos hacer esto tan eficientemente como sea posible. Dichos ejemplos involucran problemas, modelados como un problema de flujo en redes, donde se pretende maximizar el envío de entidades de un punto a otro en la red. En particular, a esta clase se le denomina Problema de Flujo Máximo.

Dado un problema, un algoritmo genérico nos proporciona una estrategia general de solución para el problema, sin detallar algunos métodos involucrados en la solución. La clave de diseñar un algoritmo genérico consiste en identificar los métodos que generalizan la solución.

A partir de un algoritmo genérico, la especificación de sus métodos proporcionará una especialización del algoritmo. El objetivo de un algoritmo genérico es proporcionar, a partir de él, diferentes algoritmos (especializaciones) que solucionen un mismo problema con la estrategia general propuesta.

Este enfoque genérico facilita tanto la justificación como el diseño de algoritmos, además de que el cálculo del desempeño computacional de los algoritmos resulta menos complejo.

El problema de *flujo máximo* en redes consiste en enviar el mayor flujo posible a lo largo de una red. Los algoritmos que resuelven este problema se pueden clasificar en dos tipos:

(1) los algoritmos basados en rutas aumentantes y
(2) los algoritmos basados en la estrategia de empuje-preflujo

Para cada una de estas clasificaciones presentaremos un algoritmo genérico.

A lo largo de este trabajo iremos presentando también especializaciones de los algoritmos genéricos, las cuales justificaremos y calcularemos su desempeño computacional

mencionando además sus ventajas y desventajas.

El objetivo de este trabajo es que sirva como material de apoyo didáctico para cursos avanzados del Análisis de Redes tanto a nivel licenciatura como a nivel posgrado. Este documento está basado principalmente en los libros "Network flows: Theory, Algorithms, and Applications", Ahuja, R.K., Magnati, T.L. y Orlin, J.B. [1]; y "Applied and Algorithmic Graph Theory", Chartran, G. and Oellermann, R. [2]

El presente trabajo está organizado en la siguiente forma:

En el Capítulo 1 mostramos un panorama general del problema a estudiar.

En el Capítulo 2 se da una breve introducción al Análisis de algoritmos el cual es parte fundamental de este trabajo ya que con base en el análisis del peor de los casos se compara la eficiencia de los algoritmos presentados. En este mismo capítulo recordamos algunos conceptos y definiciones de Teoría de Redes las cuales nos ayudarán a la comprensión del texto. Una vez definidos estos conceptos, presentaremos el problema de Flujo Máximo y daremos los supuestos sobre los cuales vamos a trabajar.

En el Capítulo 3 se presenta uno de los teoremas fundamentales para la Teoría de flujo en redes, el Teorema de Flujo Máximo - Corte Mínimo. Durante la demostración de este teorema se deduce un algoritmo que resuelve el problema de flujo máximo, tal algoritmo es analizado y su complejidad es calculada.

En el Capítulo 4 se presenta el algoritmo genérico de Rutas Aumentantes y tres de sus especializaciones. Cada uno de estos algoritmos será descrito con detalle, además de ser analizados y de calcular su desempeño computacional.

En el Capítulo 5 mostramos el enfoque de los algoritmos de empuje - preflujo. Como inicio, presentamos y analizamos el algoritmo genérico de preflujo. Lo anterior nos llevará a la presentación de sus especializaciones las cuales también son detalladas, analizadas y para todas ellas se determina su desempeño computacional.

Al final y con base en este trabajo concluiremos que los algoritmos basados en el enfoque de preflujo son más eficientes que los algoritmos basados en rutas aumentantes.

Paralelamente a la elaboración de este documento se realizó, en una hoja de cálculo, la simulación de los algoritmos descritos durante este trabajo. Tal herramienta pretende mostrar cómo trabajan los algoritmos. En el Capítulo 6 presentamos tales simulaciones.

1. Panorama General

El problema de *flujo máximo* puede ser planteado de la siguiente forma: En una red con capacidades en los arcos, queremos enviar tanto flujo como sea posible entre dos nodos especiales, un nodo origen s y un nodo destino t, sin exceder la capacidad de cada arco. Existen varios algoritmos para resolver el problema de *flujo máximo* los cuales se dividen en dos tipos:

1. Algoritmos de *rutas aumentantes*: suponen la conservación de flujo en cada nodo de la red excepto en el nodo origen y el nodo destino. Estos algoritmos van enviando flujo a lo largo de rutas que van desde el nodo origen al nodo destino.

2. Algoritmos de *empuje - preflujo*: estos algoritmos sobre saturan la red de tal forma que algunos nodos tienen exceso en etapas intermedias. Estos algoritmos liberan flujo incrementalmente desde los nodos con exceso, ya sea hacia el nodo destino (hacia adelante) o al nodo origen (hacia atrás).

Para cada una de estas clasificaciones presentaremos un algoritmo genérico.

El algoritmo genérico de rutas aumentantes es el algoritmo más intuitivo para resolver el problema de flujo máximo, sin embargo, veremos que tiene limitaciones computacionales importantes. La complejidad de este algoritmo es poco atractiva en la práctica para problemas con capacidades muy grandes y, además para problemas con capacidades *irracionales* el algoritmo podría converger a una solución no óptima.
Con base en estas limitaciones mostraremos métodos que se han desarrollado con algunas adecuaciones que buscan mejorar el desempeño del algoritmo genérico de rutas aumentantes. Es posible mejorar el tiempo de ejecución del algoritmo genérico de rutas aumentantes. En general, consideraremos las siguientes ideas:

1. Aumentar flujo en grandes cantidades.

2. Limitar el tipo de rutas que pueden ser utilizadas en cada aumento.

Como veremos, el algoritmo genérico de rutas aumentantes puede llegar a ser muy lento porque podría ejecutar un gran número de aumentos enviando en cada uno de ellos un flujo de valor muy pequeño. Esto sugiere una estrategia para mejorar el algoritmo de rutas aumentantes: aumentar flujo a través de una ruta de tal forma que el aumento sea máximo, con esto el número de aumentos restantes sería relativamente pequeño. El *algoritmo de rutas aumentantes de máxima capacidad* utiliza esta idea: siempre aumenta flujo a lo largo de rutas con la máxima capacidad.

En este trabajo mostraremos una variación de este algoritmo cuyos aumentos de flujo son a través de rutas que permitan un aumento *suficientemente grande*, no necesariamente el máximo, el cual también se ejecuta en un tiempo del mismo orden que el anterior, además es más fácil de implementar. Este algoritmo es conocido como *algoritmo de capacidades escalables*.

Otra estrategia que discutiremos para mejorar el algoritmo de rutas aumentantes es una implementación que es totalmente independiente de la capacidad de los arcos. Por ejemplo, restringir la elección de rutas por las cuales aumentamos flujo, siempre aumentar flujo a través de una *ruta más corta* desde el nodo origen al nodo destino, definiendo ruta más corta como una ruta dirigida que consiste del menor número de arcos. Si aumentamos flujo por la ruta más corta, el tamaño de cualquier ruta más corta o se mantiene igual o se incrementa. Además, en m aumentos, donde m es el número de arcos en la red, el tamaño de la ruta más corta es seguro que se incremente. Ya que ninguna ruta tiene más de $n-1$ arcos, donde n es el número de nodos en la red, este resultado garantiza que el número de aumentos es a lo más $(n-1)m$. A este algoritmo lo llamamos: *algoritmo de rutas aumentantes más cortas*.

Una vez analizadas estas mejoras al algoritmo genérico de rutas aumentantes, explicaremos detalladamente el otro tipo de algoritmo, que es más *reciente*, conocido como *algoritmos de empuje - preflujo*. Este ha surgido como la técnica más poderosa tanto teórica como computacionalmente para la solución de problemas de flujo máximo. El *algoritmo de empuje - flujo* usa la siguiente idea:

Tener flexibilidad respecto al principio de conservación de flujo durante pasos intermedios del algoritmo, lo que implica que cada cambio de flujo no necesariamente significaría un aumento que inicia en el nodo origen y termina en el nodo destino.

Aunque este algoritmo identifica la ruta más corta, no envía flujo a lo largo de rutas del nodo origen al nodo destino, en su lugar, envía flujo por arcos individuales. Por lo que el algoritmo logra una rapidez superior a la obtenida por cualquiera de los algoritmos de rutas aumentantes.

2. Conceptos Básicos

En este capítulo veremos una introducción sobre el Análisis de algoritmos el cual es fundamental para este trabajo, también definiremos y explicaremos algunos conceptos básicos de la teoría de redes los cuales nos servirán para facilitar la comprensión del texto.

2.1. Análisis de Algoritmos

Un *algoritmo* es un conjunto de reglas o instrucciones bien definidas para resolver un problema paso a paso en un tiempo finito. Por un *problema* nos referimos a una cuestión a resolver; por ejemplo el problema de flujo máximo, el problema de ordenamiento, el problema de búsqueda, el problema de ruta más corta. Un problema generalmente posee parámetros por ejemplo, para el problema de búsqueda los parámetros son el elemento a buscar y la lista donde buscar.

Un *ejemplar* es un caso particular de los parámetros, es decir, son datos específicos para todos los parámetros del problema. Entonces, un *algoritmo* resuelve un problema P si cuando lo aplicamos a cualquier ejemplar de P, el algoritmo garantiza encontrar una solución para P.

Ejemplo. Consideremos el problema de encontrar el máximo elemento de un conjunto de números. A continuacuón definamos formalmente este problema.

> **Problema**: Encontrar el máximo.
> Dado un conjunto S de números reales, con S no vacío y finito. Encontrar $x \in S$ tal que para toda $y \in S$, $x \geq y$.
> **Parámetros.** S un conjunto no vacío y finito de números reales.
> Algunos ejemplares para este problema son:
>
> $$E_1 = \{3\dot{1}4, 2{,}27, 318, 932\}$$
>
> $$E_1 = \{500, 1{,}25, 1976, 30\}$$

Complejidad de los algoritmos

Generalmente estamos interesados en encontrar los algoritmos más "eficientes" para resolver un problema. En este trabajo nos basaremos en el *tiempo de ejecución* del algoritmo como la métrica para determinar su eficiencia.

El número de pasos que ejecuta un algoritmo podría ser diferente de un ejemplar

a otro del mismo problema. Por lo que un algoritmo podría resolver, algunos ejemplares "buenos" de manera rápida mientras que algunos otros ejemplares "malos" los resolvería tomándose más tiempo. Con esto, surge la pregunta: ¿cómo podemos seleccionar el mejor algoritmo de entre todos los algoritmos que resuelven un mismo problema? Para contestar a esta pregunta, en este trabajo, utilizaremos lo que en la literatura se denomina:

- *Análisis del peor de los casos.* Este análisis provee una cota superior para el número de pasos que un algoritmo puede necesitar para cualquier ejemplar del problema. En este análisis se cuenta el número máximo de pasos posibles; en consecuencia ofrece una garantía en el número de pasos que un algoritmo necesitará para cualquier ejemplar de un problema.

Utilizando este análisis tenemos que un algoritmo es mejor que otro si requiere menos número de pasos para resolver el problema en el peor de los casos. El análisis del peor de los casos tiene la desventaja de permitir que ejemplares "patológicos" determinen el tiempo de ejecución de un algoritmo, aún cuando estos ejemplares sean extremadamente raros en la práctica. A pesar de esta desventaja, este análisis es el más común para medir el tiempo ejecución de los algoritmos.

El tiempo que requiere un algoritmo para resolver un problema, al cual llamaremos también *tiempo de ejecución* del algoritmo, depende tanto de la naturaleza como del tamaño de los datos de entrada, ejemplares grandes requieren más tiempo. Diversos ejemplares del mismo tamaño pueden requerir diferente tiempo debido a la variedad entre sus datos.

Una *función de complejidad* para un algoritmo, es una función sobre el tamaño del ejemplar y especifica el tiempo máximo que necesita el algoritmo para resolver cualquier ejemplar de un problema de un tamaño dado. En otras palabras, la función de complejidad mide la razón de crecimiento del tiempo de solución en base al incremento en el tamaño del ejemplar. Nos referiremos a esta función de complejidad simplemente como la *complejidad o desempeño computacional* del algoritmo.

Para definir la complejidad de un algoritmo de manera más completa, describiremos la notación "O grande".

Definición 2.1 *Se dice que un algoritmo se ejecuta en tiempo $O(f(n))$ si para algún número real positivo C y un entero no negativo n_0, el tiempo requerido por el algoritmo, digamos $g(n)$, cumple:*

$$g(n) \leq C \cdot f(n) \qquad \text{para toda } n > n_0.$$

Se escribe $g(n)$ es $O(f(n))$ y se dice que $f(n)$ es una cota superior para $g(n)$.

Aunque hemos mencionado esta definición en términos de un simple parámetro n, que es el tamaño del ejemplar, podemos incorporar fácilmente la definición de más parámetros.

Por ejemplo, para una gráfica con n vértices, m arcos y costos sobre los arcos, el tiempo de ejecución de un algoritmo podría depender de m, n, U, donde U es una cota superior para el costo de cualquier arco y lo denotaríamos como $O(g(n, m, U))$.

La complejidad de un algoritmo es una cota superior sobre su tiempo de ejecución cuando los valores de n son muy grandes. Esta característica es justificable ya que nuestro interés está enfocado en el comportamiento de los algoritmos cuando el tamaño de la entrada de los datos es grande, porque esto determina los límites de la aplicabilidad del algoritmo en la práctica o la implantación del mismo en una computadora.

La notación O grande solamente indica el término más dominante en el tiempo de ejecución, porque para valores de n suficientemente grandes, los términos con una razón de crecimiento menor llegan a ser insignificantes. Por ejemplo, si el tiempo de ejecución de un algoritmo es $100n + n^2 + 0.0001n^3$, entonces para toda $n \geq 100$, el segundo término domina al primer término, y para toda $n \geq 10,000$, el tercer témino domina al segundo término. Así pues, la complejidad de este algoritmo es $O(n^3)$.

Algoritmos con tiempo de ejecución polinomial y exponencial

En años reciente se ha aceptado la idea de que un algoritmo es "bueno" si su complejidad para el peor de los casos está acotada por una función polinomial sobre los parámetros del problema, es decir, una función polinomial de n, m y $\log U$. Tales algoritmos son llamados *algoritmos de tiempo de ejecución polinomial.*

Algunos ejemplos de cotas de tiempo de ejecución polinomial son $O(n^2)$, $O(nm)$, $O(m + n \log U)$, $O(nm \log(n^2/m))$ y $O(nm + n^2 \log U)$. Notemos que $\log n$ esta acotado polinomialmente porque su razón de crecimiento es menor que n.

Se dice que un algoritmo tiene *tiempo de ejecución exponencial* si su tiempo de ejecución en el peor de los casos crece como una función que no puede ser polinomialmente acotada por el tamaño de la entrada. Algunos ejemplos de cotas de tiempo exponencial son $O(nU)$, $O(2^n)$ y $O(n!)$. Obsevemos que nU puede no ser acotada por una función polinomial de n y $\log U$, U podría ser una función exponencial en n.

Existen varias razones para preferir un algoritmo de tiempo de ejecución polinomial a uno de tiempo exponencial. Cualquier algoritmo de tiempo polinomial es asintótica-

mente superior a cualquier algoritmo de tiempo exponencial, aún en casos extremos. Por ejemplo, $n^{4,000}$ es menor que $n^{0,1 \log n}$ si y sólo si n es suficientemente grande, es decir, $n \geq 2^{40,001}$.

Las funciones de complejidad exponencial tienen una razón de crecimiento explosivo, en general, los algoritmos con esta complejidad son capaces de resolver solamente problemas de tamaño pequeño.

Una breve exploración de los efectos de mejora tecnológica en algoritmos es aún más relevante para entender el impacto de varias funciones de complejidad. Consideremos un algoritmo cuya complejidad es $O(n^2)$. Supongamos que el algoritmo es capaz de resolver un ejemplar de tamaño n_1 en 1 hora en una computadora con velocidad de s_1 instrucciones por segundo. Si incrementamos la velocidad de la computadora a s_2, entonces $(n_2/n_1)^2 = s_2/s_1$ especifica el tamaño n_2 del ejemplar que el algoritmo puede resolver en el mismo tiempo. En consecuencia, un incremento de 100 veces en la velocidad de la computadora nos podría permitir resolver un problema que fuera 10 veces mayor. Ahora consideremos un algoritmo con tiempo de ejecución exponencial de complejidad $O(2^n)$. Igual que antes, sean n_1 y n_2 el tamaño de los problemas resueltos con computadoras con velocidades s_1 y s_2 respectivamente, en un tiempo de 1 hora. Entonces $s_2/s_1 = 2^{n_2}/2^{n_1}$. Alternativamente, $n_2 = n_1 + \log(s_2/s_1)$. En este caso, un incremento de 100 veces en la velocidad de la computadora podría permitir resolver problemas que son solamente 7 *unidades* mayor en el mismo tiempo.

Complejidad	Valores de n			
	2	8	32	64
$\log n$	1 seg.	3 seg.	5 seg.	6 seg.
n	2 seg.	8 seg.	32 seg.	1.07 min.
$n \log n$	2 seg.	24 seg.	2.67 min.	6.4 min.
n^2	4 seg.	1.07 min.	17.07 min.	1.14 hrs.
n^3	8 seg.	8.53 min.	9.1 hrs.	3.03 días
2^n	4 seg.	4.27 min.	1.36 centurias	5.86×10^9 centurias
$n!$	2 seg.	4.2 hrs.	8.34×10^{25} centurias	4.02×10^{79} centurias

Figura 1: Comparación de funciones de complejidad. Tiempos aproximados [1]

La discusión anterior muestra que un incremento substancial en la velocidad de la computadora nos permite resolver problemas con algoritmos de tiempo polinomial que son mayores por un factor multiplicativo; mientras que para los algoritmos de tiempo exponencial solamente obtenemos mejoras del orden de una suma en el tamaño del ejemplar. Con lo que, la mejora en las capacidades de las computadoras puede tener solamente un impacto marginal en la solución de problemas de tiempo exponencial.

En resumen, nuestro objetivo es obtener algoritmos de tiempo polinomial y dentro de este dominio debemos buscar un algoritmo con la razón de crecimiento menor. Por ejemplo, preferimos $O(\log n)$ a $O(n^k)$ para $k > 0$, y preferimos $O(n^2)$ a $O(n^3)$.

La jerarquía de algunas funciones de complejidad se muestra a continuación:

$$O(1),\ O(\log n),\ O(n),\ O(n \log n),\ O(n^2),\ O(n^j),\ O(n^k),\ O(a^n),\ O(b^n).\ O(n!)$$

donde $k > j > 2$ y $b > a > 1$. Si una función de complejidad $f(n)$ está a la izquierda de otra $g(n)$ entonces la razón de crecimiento de $f(n)$ es menor que la de $g(n)$ para una n suficientemente grande.

En la Figura 1 se muestra el tiempo de ejecución (aproximado) de varias funciones de complejidad para algunos valores de n.

2.2. Teoría de Redes

En esta sección daremos algunas de las definiciones básicas de Teoría de Redes y otros conceptos relacionados que serán usados durante el resto del documento.

Gráfica. Una gráfica es una pareja de conjuntos $G = (N, A)$ donde N es un conjunto no vacío de elementos denominados nodos o vértices y A es un conjunto de pares no ordenados de nodos. La Figura 2 muestra una gráfica, en la cual $N = \{1, 2, 3, 4, 5, 6, 7\}$ y $A = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 6\}, \{3, 4\}, \{3, 5\}, \{4, 6\}, \{4, 5\}, \{5, 7\}, (6, 5], \{6, 7\}\}$.

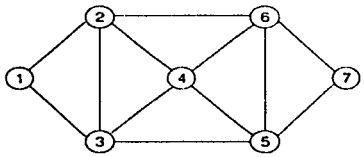

Figura 2: Ejemplo de una gráfica no dirigida.

Gráfica dirigida. Es una gráfica $G = (N, A)$ donde N es el conjunto de *nodos*, $N \neq \phi$ y $A \subseteq V \times V$, es decir, A es un conjunto de parejas ordenadas a las cuales llamaremos *arcos*. Una gráfica dirigida también es llamada digráfica. En la Figura 3 se muestra una gráfica donde el conjunto de nodos es $N = \{1, 2, 3, 4, 5, 6, 7\}$ y $A = \{(1, 2), (1, 3), (2, 3), (2, 4), (2, 6), (3, 4), (3, 5), (4, 6), (4, 5), (5, 7), (6, 5), (6, 7)\}$ es el conjunto de arcos.

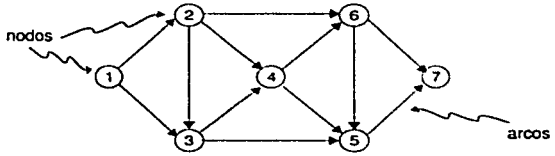

Figura 3: Ejemplo de una gráfica dirigida.

Red. Es una gráfica dirigida con funciones sobre los arcos o los nodos. Para este material consideraremos redes con una función de capacidad u sobre los arcos y dos nodos distinguidos denominados nodo *origen* s, y nodo *destino*, t.
A tal Red la denotamos $G = (N, A, u, s, t)$, con: $u : A \longrightarrow I\!R^+$, donde u_{ij} representa la capacidad del arco (i, j). La Figura 4 muestra un ejemplo de una Red, el valor asociado a cada arco se refiere a su capacidad u_{ij}.

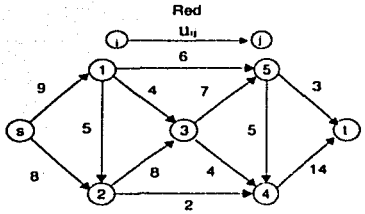

Figura 4: Ejemplo de una Red.

Extremos. Un arco (i, j) tiene dos *puntos extremos*, i y j. Nos referimos al nodo i como el nodo inicial del arco (i, j) y al nodo j como el nodo final. Un arco (i, j) es *incidente* a los nodos i y j. El arco (i, j) es un arco *saliente* del nodo i y es un arco *entrante* para el nodo j. Si un arco $(i, j) \in A$, decimos que el nodo j es *adyacente* al nodo i.

Lista de Adyacencia. La *lista de adyacencia de arcos* $A(i)$ de un nodo i es el conjunto de arcos que emanan de ese nodo, esto es $A(i) = \{(i, j) \in A : j \in N\}$. La *lista de adyacencia de nodos* $A(i)$ es el conjunto de nodos adyacentes a ese nodo; en este caso, $A(i) = \{j \in N : (i, j) \in A\}$.

En el ejemplo de la Figura 3 la lista de adyacencia de arcos para el nodo 2 es $A(2) = \{(2, 3), (2, 4), (2, 6)\}$ y la lista de adyacencia de nodos para el mismo nodo 2, está definida como $A(2) = \{3, 4, 6\}$.

Arcos paralelos. Los arcos paralelos son dos o más arcos con el mismo nodo inicial y el mismo nodo final.

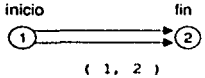

Figura 5: Ejemplo de arcos paralelos.

Subgráfica. Una gráfica $G' = (N', A')$ es una *subgráfica* de $G = (N, A)$ si $N' \subseteq N$ y $A' \subseteq A$. La Figura 6 ilustra esta definición.

Camino. Un *camino* en una gráfica dirigida $G = (N, A)$ es una subgráfica de G que consiste de una secuencia de nodos y arcos $i_1 - a_1 - i_2 - a_2 - \cdots i_{r-1} - a_r - i_r$ que satisface que para toda $1 \leq k \leq r - 1$, $a_k = (i_k, i_{k+1}) \in A$ o $a_k = (i_{k+1}, i_k) \in A$.

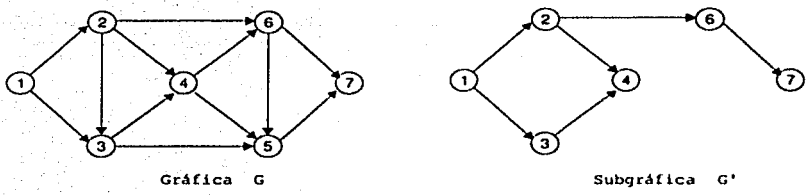

Figura 6: Subgráfica.

Alternativamente, podemos hacer referencia a un camino simplemente como una secuencia de nodos.

Para ilustrar esta definición, en la Figura 7 se muestran dos ejemplos de caminos, el $(a) : 1-2-4-3-5-6-7$ y el $(b) : 1-2-3-5-6-4-5-7$.

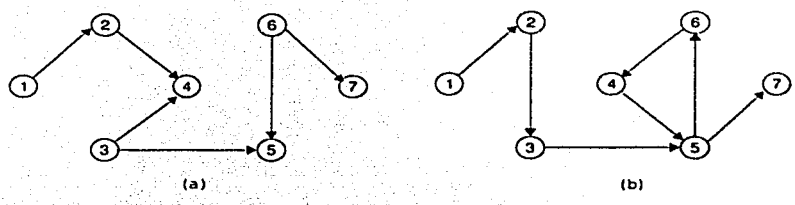

Figura 7: Ejemplos de caminos.

Camino dirigido. Un *camino dirigido* es una versión "orientada" de un camino en el sentido de que para dos nodos consecutivos i_k e i_{k+1} en el camino, $(i_k, i_{k+1}) \in A$. El camino de la Figura 7(a) no es dirigido mientras que el de la Figura 7(b) si lo es.

Ruta. Una *ruta* es un camino que no tiene repetición de nodos. El camino de la Figura 7(a) es una ruta, en cambio el de la Figura 7(b) no lo es porque repite el nodo 5.
Podemos particionar los arcos de una ruta en dos grupos: *arcos hacia adelante* y *arcos hacia atrás*. Un arco (i, j) en la ruta es *un arco hacia adelante* si la ruta pasa por el nodo i antes de pasar al nodo j, y es un *arco hacia atrás* en otro caso.

Ruta dirigida. Una *ruta dirigida* es un camino dirigido sin repetición de nodos. En otras palabras, una ruta dirigida no tiene *arcos hacia atrás*.
Ciclo. Un *ciclo* es una ruta $i_1 - i_2 - \ldots - i_r$ dónde $i_1 = i_r$.

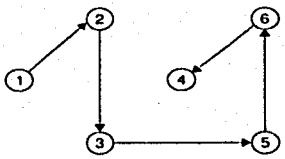

Figura 8: Ejemplo de una ruta dirigida.

Predecesores. Definimos el predecesor *pred(j)* para cada nodo j en la ruta, de la siguiente forma; si i, j son dos nodos consecutivos en la ruta, $pred(j) = i$. Por convención, el predecesor del nodo inicial es cero. Para la ruta dirigida de la Figura 8 tenemos que: $pred(1) = 0$, $pred(2) = 1$, $pred(3) = 2$, $pred(5) = 3$, $pred(6) = 5$, $pred(4) = 6$.

Cortadura. Una *cortadura* es una partición del conjunto de nodos N en dos partes, S y $\bar{S} = N \setminus S$. Cada cortadura define un conjunto de arcos que consiste en aquellos arcos que tienen un extremo en S y el otro en $\bar{S}$. Nos referiremos a este conjunto de arcos por la notación $[S, \bar{S}]$.

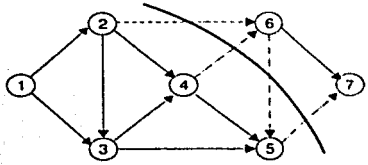

Figura 9: Ejemplo de una cortadura.

$$S = \{1, 2, 3, 4, 5\} \text{ y } [S, \bar{S}] = \{(2, 6), (4, 6), (6, 5), (5, 7)\}$$

Conectividad. Decimos que dos nodos i y j están *conectados* si la gráfica contiene al menos una ruta del nodo i al nodo j. Una gráfica es *conexa* si cada par de nodos esta conectado; en otro caso, la gráfica no es conexa.

Fuertemente Conexa. Una gráfica conexa es *fuertemente conexa* si contiene al menos una ruta *dirigida* entre cualquier par de nodos de la gráfica.

Árbol. Un *árbol* es una gráfica conexa que no tiene ciclos. La Figura 10 muestra dos ejemplos de árboles.

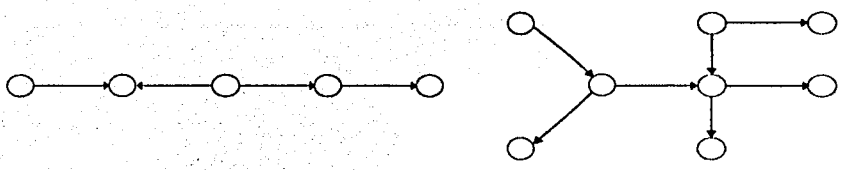

Figura 10: Ejemplo de dos árboles.

Bosque. Un bosque es una colección de árboles. En la Figura 11 se observa un ejemplo de un bosque.

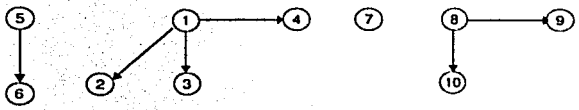

Figura 11: Ejemplo de bosque.

Árbol con raíz. Un árbol con raíz es un árbol con un nodo especial, llamado *raíz*, podemos ver a un árbol con raíz como si colgara de su raíz. La Figura 12 muestra un árbol con raíz; en este ejemplo, el nodo 1 es la raíz del árbol.

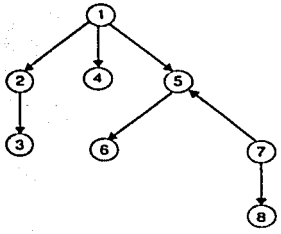

Figura 12: Ejemplo de árbol con raíz.

Algunas veces podemos ver los arcos de un árbol con raíz definiendo relaciones de predecesores-sucesores (o de padres- hijos). Por ejemplo, en la Figura 12, el nodo 5 es el predecesor del nodo 6 y 7, y el nodo 1 es el predecesor de los nodos 2, 4 y 5. Cada nodo i excepto el nodo raíz tiene un único predecesor, el cual es el siguiente nodo en la ruta que va de ese nodo i al nodo raíz, además dicha ruta es única. Los *descendientes*

de un nodo i consisten en el nodo mismo, sus sucesores y los sucesores de sus sucesores y así sucesivamente. Por ejemplo, en la Figura 12 el conjunto 5,6,7,8 es el conjunto de descendientes de l nodo 5. Decimos que un nodo es un *ancestro* de todos sus nodos descendientes. En la misma figura, el nodo 2 es un ancestro de él mismo y del nodo 3.

Representación de una Red. Una de las representaciones más populares de una red es la *matriz de Adyacencia de nodos*, o simplemente matriz de adyacencia, que almacena las redes como una matriz $\mathcal{H} = \{h_{ij}\}$. La matriz tiene un renglón y una columna para cada nodo, y la ij-ésima entrada h_{ij} es igual a 1 si $(i, j) \in A$ e igual a 0 en otro caso. En la Figura 13 tenemos la representación de la red de la Figura 9 como una matriz de Adyacencia de nodos.

$$\mathcal{H} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figura 13: Representación de una red con una matriz de Adyacencia.

Representación de una lista de Adyacencia. Anteriormente definimos una *lista de adyacencia de arcos* $A(i)$ de un nodo i como el conjunto de arcos que emanan de ese nodo, esto es, el conjunto de arcos $(i, j) \in A$. Similarmente, definimos la *lista de adyacencia de nodos* del nodo i como el conjunto de nodos j para los cuales $(i, j) \in A$. La *representación de la lista de adyacencia* almacena la lista de adyacencia de nodos de cada nodo como una lista ligada. Una lista ligada es una colección de celdas cada una de las cuales puede contener uno o más campos. La lista de adyacencia de nodos para el nodo i sera una lista ligada que tenga $|A(i)|$ celdas y cada celda corresponderá a un arco $(i, j) \in A$. Las celdas correspondientes a un arco (i, j) podrían tener tantos campos como información deseamos almacenar. Un campo puede almacenar al nodo j. Podríamos usar otro campo adicional para almacenar la capacidad del arco, u_{ij}. Además, cada celda contiene un campo adicional, llamado *liga*, el cual almacena un puntero a la siguiente celda en la lista de adyacencia. Si una celda es la última en la lista de adyacencia, por convención hacemos el valor de su liga igual a NULL.

Ya que necesitamos ser capaces de almacenar y accesar n listas ligadas, una por cada nodo, también necesitamos un arreglo de punteros que apunten a la primer celda de cada lista ligada. Para esto, definimos un arreglo $n-$dimensional, *primero* cuyo ele-

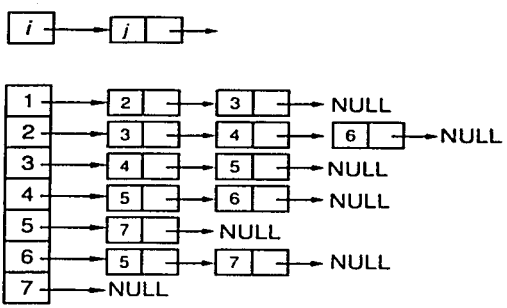

Figura 14: Representación de lista de adyacencia de una red.

mento *primero*(i) almacena el puntero a la primer celda de la lista de adyacencia del nodo i. Si la lista de adyacencia del nodo i está vacía, hacemos *primero*(i) =NULL. La Figura 14 muestra la representación en forma de lista de adyacencia de la red de la Figura 9.

2.3. Problema de Flujo Máximo

Sea $G(N, A, u, s, t)$ y definamos $U = \max_{(i,j)\in A}\{u_{ij}\}$. Para definir el problema de *flujo máximo* distinguiremos los dos nodos especiales en la red, el nodo origen s y el nodo destino t. Además tenemos que $|N| = n$ y $|A| = m$.

Un *flujo* en una red G es una función x que a cada arco (i, j) le asigna un número real positivo x_{ij}, llamado flujo en el arco (i, j) tal que el flujo en cada arco es menor o igual a su capacidad. De manera más formal :

$$x : A \longrightarrow I\!R^+ \quad \text{tal que} \quad 0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A, \qquad x \in I\!R^m.$$

El flujo que llega a un nodo i se define como:

$$\sum_{(j,i)\in A} x_{ji}$$

así mismo el flujo que sale de un nodo i se define como:

$$\sum_{(i,j)\in A} x_{ij}.$$

El valor v de un flujo x en una red G, es igual al flujo que llega al nodo destino t.

El problema de Flujo Máximo en redes consiste en encontrar el flujo máximo que puede ser enviado del nodo s al nodo t que satisfaga las capacidades de los arcos y además garantice que todo el flujo que llega a un nodo sale del mismo, con excepción de los nodos distinguidos s y t.

Formalmente, el problema de flujo máximo se define como sigue:

$$\text{Maximizar } v \text{ (flujo)} \tag{1}$$

sujeto a:

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v \text{ para } i = s \\ 0 \text{ para toda } i \in N \\ -v \text{ para } i = t \end{cases} \tag{2}$$

$$0 \leq x_{ij} \leq u_{ij} \text{ para cada } (i,j) \in A. \tag{3}$$

Nos referiremos al vector $x = \{x_{ij}\}$ que satisface las Condiciones (2) y (3) como un *flujo factible* y a su correspondiente valor v como el *valor* del flujo. A la Condición (2) se le conoce como Ley de Conservación de flujo que intuitivamente indica que todo lo que llega a un nodo es igual a lo que sale de él, excepto para los nodos s y t.

2.4. Supuestos

A continuación definimos y describimos los supuestos que utilizaremos para el problema de *flujo máximo*.

Supuesto 2.1 *Todas las capacidades son enteros no negativos.*

Esta restricción para las capacidades de los arcos de ser enteros no es necesaria para algunos algoritmos, sin embargo para otros sí, los algoritmos cuya cota de complejidad involucra U asumen que las capacidades son enteras.

En realidad, el supuesto de que los datos sean enteros no es restrictivo ya que las computadoras almacenan datos como números racionales y siempre podemos transformar un número racional a entero multiplicándolo por un entero adecuado.

Supuesto 2.2 *La red no contiene rutas dirigidas del nodo s al nodo t compuestas solamente de arcos con capacidad infinita.*

Si cada arco de una ruta dirigida P, que va de s a t, tiene capacidad infinita, entonces podemos enviar una cantidad infinita de flujo a través de esta ruta, y por lo tanto el flujo máximo es infinito.

Supuesto 2.3 *La red no contiene arcos paralelos.*

Si permitiéramos arcos paralelos tendríamos dificultades en cuanto a la notación, ya que (i, j) no podría identificar a un arco de manera única. Por lo que este supuesto es esencialmente por notación.

3. Teorema de Flujo Máximo - Corte Mínimo

En esta sección vamos a ver el teorema que es el fundamento para la justificación de uno de los algoritmos clásicos que resuelve el problema de flujo máximo. Antes de entrar de lleno al teorema, definamos algunos conceptos y mostremos algunos resultados que nos ayudarán más adelante.

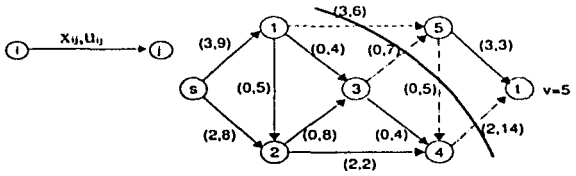

Figura 15: Ejemplo de una (s,t)-cortadura.

***(s,t)*-cortadura.** Una *(s,t)*-cortadura es una cortadura que está definida con respecto a dos nodos, s y t; y nos referiremos a ella si $s \in S$ y $t \in \bar{S}$. Además, a un arco (i,j) con $i \in S$ y $j \in \bar{S}$ le llamaremos un *arco hacia adelante* de la cortadura $[S,\bar{S}]$ y un arco (i,j) con $i \in \bar{S}$ y $j \in S$ será un *arco hacia atrás* de la cortadura $[S,\bar{S}]$.

Denotaremos $(S,\bar{S})$ al conjunto de *arcos hacia adelante* y $(\bar{S},S)$ al conjunto de *arcos hacia atrás* de la cortadura $[S,\bar{S}]$. La Figura 15 muestra un ejemplo de una (s,t)-cortadura, en la cual $S = \{s,1,2,3,4\}$ y $\bar{S} = \{5,t\}$. Además tenemos que $(S,\bar{S}) = \{(1,5),(3,5),(4,t)\}$ y $(\bar{S},S) = \{(5,4)\}$.

Capacidad de una *(s,t)*-cortadura. Definimos la capacidad $u[S,\bar{S}]$ de una cortadura $[S,\bar{S}]$ como la suma de las capacidades de los *arcos hacia adelante* en la cortadura, es decir:

$$u[S,\bar{S}] = \sum_{(i,j)\in(S,\bar{S})} u_{ij}$$

Por ejemplo, la capacidad de la cortadura de la Figura 15 es $u[S,\bar{S}] = 6+7+14 = 27$. Claramente, la capacidad de una cortadura es una cota superior para el *flujo máximo* que puede ser enviado de un nodo en S a un nodo en $\bar{S}$.

Cortadura mínima. Es una cortadura que tiene la mínima capacidad de entre todas las (s,t)-cortaduras:

$$u[S^*,\bar{S}^*] \leq u[S,\bar{S}] \text{ para todas las } (s,t)\text{-cortaduras en } G.$$

Propiedad 3.1 (Flujo a través de una (s,t)-cortadura) *Sea x un flujo factible de valor v en la red y $[S, \bar{S}]$ cualquier (s,t)-cortadura, entonces tenemos que:*

$$v = \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(i,j)\in(\bar{S},S)} x_{ij}$$

$$= \textit{(el flujo de } S \textit{ a } \bar{S}\textit{)} - \textit{(el flujo de } \bar{S} \textit{ a } S\textit{)}$$

Demostración. Aplicando la ley de conservación de flujo para los nodos en S tenemos que:

$$v = \sum_{i\in S} \left[\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} \right]$$

Podemos simplificar esta expresión si notamos que:

- siempre que dos nodos p y q pertenezcan a S y $(p,q) \in A$, la variable x_{pq} en el primer término, para $i = p$, cancela a la variable $-x_{pq}$ del segundo término, para $i = q$.
- siempre que dos nodos p y q pertenezcan ambos a $\bar{S}$ entonces x_{pq} no aparece en la expresión.

Por lo tanto, lo anterior se simplifica a:

$$v = \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(i,j)\in(\bar{S},S)} x_{ij} \qquad (4)$$

La primer suma de la expresión representa la cantidad de flujo que va de los nodos en S hacia los nodos en $\bar{S}$, y la segunda la cantidad de flujo que regresa de los nodos en $\bar{S}$ a los nodos en S. Entonces, la parte derecha de la ecuación representa el flujo total (neto) a través de la cortadura, lo que implica que el flujo a través de cualquier (s,t)-cortadura es igual a v. ◇

Propiedad 3.2 *El valor de cualquier flujo factible es menor o igual a la capacidad de cualquier cortadura en la red, esto es:*

$$v \leq \min\{u[S, \bar{S}]\} \quad \textit{para cualquier } (s,t)\textit{-cortadura}$$

Demostración. Sustituyendo $x_{ij} \leq u_{ij}$ en la expresión (4) y tomando en cuenta que $x_{ij} \geq 0$ se tiene que:

$$v \leq \sum_{(i,j)\in(S,\bar{S})} u_{ij} = u[S,\bar{S}] \tag{5}$$

Lo anterior indica que el valor de *cualquier* flujo es menor o igual a la capacidad de *cualquier* (s,t)-cortadura en la red.

Cualquier flujo del nodo s al nodo t debe pasar a través de cada (s,t)-cortadura en la red, porque cualquier cortadura divide la red en dos conjuntos ajenos, y así que el valor del flujo no puede exceder la capacidad de la cortadura. ◇

Esta propiedad implica que si descubrimos un flujo x cuyo valor es igual a la capacidad de alguna cortadura $[S,\bar{S}]$ entonces x es un *flujo máximo*. Supongamos que $[S,\bar{S}] = [\{s\}, N - \{s\}]$ es una (s,t)-cortadura. Entonces, existe una cortadura mínima. Además, si definimos $x = \{x_{ij}\} = 0$, flujo de valor cero, entonces x satisface las ecuaciones (2) y (3), por lo tanto es un flujo bien definido. Con lo que concluimos que el flujo máximo existe.
Ejemplo. Para la red de la Figura 15, podemos verificar que:

- El valor v de un flujo factible es igual al flujo a través de una cortadura, de la Ecuación 4, tenemos:

$$v = \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(i,j)\in(\bar{S},S)} x_{ij} = 3 + 0 + 2 - 0 = 5$$

- El valor v de cualquier flujo factible es menor a la capacidad de cualquier (s,t)-cortadura en la red, en este caso, para la Propiedad 3.2, tenemos:

$$v = 5 \leq 27 = u[S,\bar{S}]$$

En la Figura 16 tenemos la solución del ejemplo que estamos trabajando. En ella podemos ver que el valor del flujo máximo es $v = 14 \leq 27$ unidades, este valor se refiere a la capacidad de la cortadura que se muestra en la Figura 15.
Si observamos, la capacidad de la cortadura generada por $S = \{s,1,2,3,5\}$ y $\bar{S} = \{4,t\}$ es igual al valor del flujo v, esto es, $u[S,\bar{S}] = 2 + 4 + 5 + 3 = 14 = v$.
Con lo anterior y por la Propiedad 3.2 verificamos que el flujo es máximo y la cortadura es mínima.

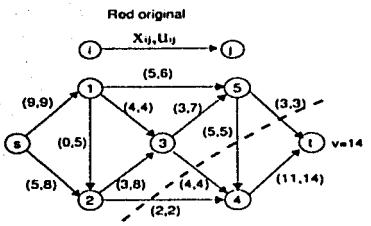

Figura 16: Solución del ejemplo trabajado

Como mencionamos anteriormente, un flujo factible x^* de valor v^* en una red G, es máximo si $v^* \geq v$ para todo flujo factible x de valor v en G; una cortadura $[S^*, \bar{S}^*]$ es mínima en G si $u[S^*, \bar{S}^*] \leq u[S, \bar{S}]$ para cualquier cortadura $u[S, \bar{S}]$.

En los siguientes resultados utilizaremos la notación $x(i,j)$ para referirnos a x_{ij} que es el flujo que va del nodo i al nodo j y $u(i,j)$ para u_{ij} que es la capacidad del arco (i,j).

Recordemos que una ruta P del nodo i_1 al nodo i_n, $P(i_1, i_n)$, en G es una secuencia finita de nodos, que empieza con el nodo i_1 y termina con el nodo i_n tal que ningún nodo en P se repite y de tal forma que (i_k, i_{k+1}) es un arco de G con $i \leq k \leq n$.
Lo podemos expresar como:

$$P : i_1, i_2, i_3 \ldots, i_{n-1}, i_n$$

Sea x un flujo en una red, una ruta $i_1, i_2, \ldots, i_n$ se dice que es *no saturada* si para cada k, $1 \leq k \leq n-1$ ocurre alguno de los dos casos:

(a) para un arco de la forma (i_k, i_{k+1}) tenemos que $x(i_k, i_{k+1}) < u(i_k, i_{k+1})$

(b) para un arco de la forma (i_{k+1}, i_k) tenemos que $x(i_{k+1}, i_k) > 0$

Intuitivamente, si tenemos la condición (a), entonces podemos incrementar el flujo de i_k a i_{k+1}, mientras que si ocurre la condición (b), entonces podemos *regresar* flujo de i_{k+1} a i_k. Si P es una ruta no saturada que va del nodo origen s al nodo destino t, se dice que P es una ***ruta aumentante***.

En la demostración del Teorema 3.1 mostraremos que una ruta aumentante puede ser utilizada para incrementar el valor v de un flujo x. Antes de presentar y demostrar el teorema ilustremos con un ejemplo las ideas que usaremos para su demostración.

Consideremos la red de la Figura 17 como ejemplo. Veamos que: $P : s, 1, 5, 4, t$ es

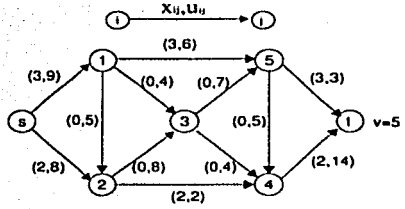

Figura 17: Ejemplo de red.

una ruta aumentante.

Sea $i_1 = s$, $i_2 = 1$, $i_3 = 5$, $i_4 = 4$, $i_5 = t$, si (i_k, i_{k+1}) satisface la condición (a), entonces $\Delta_k = u(i_k, i_{k+1}) - x(i_k, i_{k+1})$. En otro caso, (i_{k+1}, i_k) satisface la condición (b) y hacemos $\Delta_k = x(i_{k+1}, i_k)$.

Definamos $\Delta = \min\{\Delta_k \mid 1 \leq k \leq 4\}$. Notemos que, en este ejemplo, $\Delta_1 = 6$, $\Delta_2 = 3$, $\Delta_3 = 5$, $\Delta_4 = 12$, por lo cual $\Delta = 3$.
Ahora, definamos un nuevo flujo x^* como sigue:

1. si (i, j) es un arco que no pertenece a P, entonces $x^*(i, j) = x(i, j)$;
2. si (i, j) es un arco que si pertenece a P y cumple con la condición (a), entonces $x^*(i, j) = x(i, j) + \Delta$;
3. si (i, j) es un arco que si pertenece a P y cumple con la condición (b), entonces $x^*(i, j) = x(i, j) - \Delta$.

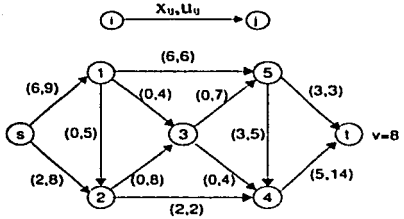

Figura 18: Ejemplo de un flujo aumentado.

Aplicando estas reglas obtenemos un nuevo flujo de valor $v^* = v + \Delta = 5 + 3 = 8$, como se muestra en la Figura 18.

El siguiente teorema muestra una relación entre las rutas aumentantes y el *flujo máximo.*

Teorema 3.1 (Rutas aumentantes) *Sea G una red, un flujo factible x en G es máximo si y sólo si no existen rutas aumentantes en G*

Demostración. Sean s y t el nodo origen y el nodo destino en G. Primero supongamos que G contiene una ruta aumentante

$$P : s = i_0, i_1, i_2, \ldots, i_{n-1}, i_n = t$$

Entonces, para cada k, $1 \leq k \leq n$

(a) si el arco es de la forma (i_{k-1}, i_k) tenemos que $x(i_{k-1}, i_k) < u(i_{k-1}, i_k)$ o,

(b) si el arco es de la forma (i_k, i_{k-1}) tenemos que $x(i_k, i_{k-1}) > 0$

Para cada k, el incremento Δ_k se define como:

- si es de la forma (i_{k-1}, i_k), hacemos $\Delta_k = u(i_{k-1}, i_k) - x(i_{k-1}, i_k)$
- si es de la forma (i_k, i_{k-1}), hacemos $\Delta_k = x(i_k, i_{k-1})$

Declaramos $\Delta = \min\{\Delta_k \mid 1 \leq k \leq n\}$, observemos que $\Delta \geq 1$.

Definamos un nuevo flujo x^* de la siguiente forma:

$$x^*(i,j) = \begin{cases} x(i,j) + \Delta & \text{si} \quad i = i_{k-1} \text{ y } j = i_k \text{ para algún } 1 \leq k \leq n; \\ x(i,j) - \Delta & \text{si} \quad i = i_k \text{ y } j = i_{k-1} \text{ para algún } 1 \leq k \leq n; \\ x(i,j) & \text{si} \quad (i,j) \notin P \end{cases}$$

Con esto, falta demostrar que x^* es un flujo bien definido y que $v^* > v$.

Por como está definido el nuevo flujo x^* tenemos que $0 \leq x^*(i,j) \leq u(i,j)$ para cada arco en N, por lo que x^* satisface la Condición (3). Ahora mostraremos que también satisface la Condición (2).

Sea $i \in N \setminus \{s,t\}$, si i no pertenece a P, entonces $x^*(i,j) = x(i,j)$ para el conjunto de nodos $A(i) = \{j \in N : (i,j) \in A\}$ y además, $x^*(j,i) = x(j,i)$ para $A'(i) = \{j \in N : (j,i) \in A\}$. Ya que

$$\sum_{j \in A(i)} x(i,j) - \sum_{j \in A'(i)} x(j,i) = 0$$

entonces en este caso, deducimos que

$$\sum_{j\in A(i)} x^*(i,j) - \sum_{j\in A'(i)} x^*(j,i) = 0$$

Con lo que concluimos que x^* satisface la Condición (2) para cualquier nodo que no pertenece a P.

Ahora bien, supongamos que i sí pertenece a P, entonces $i = i_k$ para algún k, $1 \leq k \leq n-1$. Consideremos los siguientes tres casos.

Figura 19: Primer caso.

Caso 1. Supongamos que tenemos los arcos (i_k, i_{k-1}) y (i_k, i_{k+1}), como se ilustra en la Figura 19:.
Entonces

$$\sum_{j\in A'(i)} x^*(j,i) = \sum_{j\in A'(i)} x(j,i)$$

Posteriormente,

$$\sum_{j\in A(i)} x^*(i,j) = x^*(i_k, i_{k-1}) + x^*(i_k, i_{k+1}) + \sum_{j\in T} x(i,j)$$

donde $T = A(i) \setminus \{i_{k-1}, i_{k+1}\}$.

Como

$$x^*(i_k, i_{k-1}) = x(i_k, i_{k-1}) - \Delta \quad \text{y} \quad x^*(i_k, i_{k+1}) = x(i_k, i_{k+1}) + \Delta$$

tenemos que

$$x^*(i_k, i_{k-1}) + x^*(i_k, i_{k+1}) = x(i_k, i_{k-1}) + x(i_k, i_{k+1})$$

y, de ahí

$$\sum_{j\in A(i)} x^*(i,j) = \sum_{j\in A(i)} x(i,j)$$

Además, como x es un flujo, la Condición (2) se cumple.

$$\sum_{j\in A(i)} x^*(i,j) - \sum_{j\in A'(i)} x^*(j,i) = \sum_{j\in A(i)} x(i,j) - \sum_{j\in A'(i)} x(j,i) = 0$$

De manera similar, podemos mostrar que

$$\sum_{j\in A(i)} x^*(i,j) - \sum_{j\in A'(i)} x^*(j,i) = 0$$

en cada uno de los dos casos restantes.

Caso 2. Supongamos que (i_{k-1}, i_k) y (i_{k+1}, i_k), la Figura 20 muestra este caso.

Figura 20: Segundo caso.

Caso 3. Supongamos que (i_{k-1}, i_k) y (i_k, i_{k+1}) o (i_k, i_{k-1}) y (i_{k+1}, i_k), la Figura 21 muestra este caso.

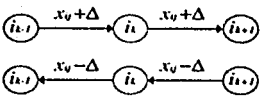

Figura 21: Tercer caso.

De esta forma si $i = i_k$ pertenece a P, después del incremento, siempre satisface la Condición (2), es decir, cumple con la Ley de Conservación de flujo.
Por lo tanto x^* es un flujo en G y

$$v^* = \sum_{j\in A(s)} x^*(s,j) - \sum_{j\in A'(s)} x^*(j,s)$$

Ahora falta mostrar que $v^* > v$.
Sea (s, i_1), entonces

$$x^*(s,i_1) = x(s,i_1)+\Delta, \quad \text{y} \quad x^*(s,j) = x(s,j) \quad \text{para toda} \quad j \in A(s)\setminus\{i_1\}.$$

También, $x^*(j,s) = x(j,s)$ para toda $j \in A'(s)$. Por lo que concluimos que, en este caso, $v^* = v + \Delta$.

Veamos el otro, sea (i_1, s), entonces

$$x^*(i_1,s) = x(i_1,s)-\Delta, \quad \text{y} \quad x^*(j,s) = x(j,s) \quad \text{para toda} \quad j \in A'(s)\setminus\{i_1\}.$$

También, $x^*(s,j) = x(s,j)$ para toda $j \in A(s)$. Una vez más, tenemos que $v^* = v + \Delta$. Así que, $v^* = v + \Delta$ en ambos casos, lo cual implica que $v^* > v$ y por lo tanto x no es un flujo máximo.

Inversamente, supongamos que en la red no hay rutas aumentantes. Comencemos mostrando que en este caso, existe una cortadura $[S, \bar{S}]$ tal que

$$x(i,j) = u(i,j) \quad \text{para cada} \quad (i,j) \in (S, \bar{S})$$

y

$$x(i,j) = 0 \quad \text{para cada} \quad (i,j) \in (\bar{S}, S)$$

Sea S el conjunto de nodos i para los cuales existe una ruta no saturada $s - i$. Entonces $s \in S$ y, por hipótesis, $t \notin S$. Con esto tenemos que $[S, \bar{S}]$ es una cortadura.

Supongamos que $(j,w) \in (S, \bar{S})$, ya que $j \in S$, existe una ruta no saturada P de $s - j$. Por lo que $x(j,w) = u(j,w)$; de otra forma, la ruta $P^* : P, w$ es una ruta no saturada $s - w$, lo cual contradice el hecho de que $w \notin S$. Similarmente si $(j,w) \in (\bar{S}, S)$, entonces $x(j,w) = 0$.

Con lo anterior y la Propiedad 3.1 concluimos que,

$$v = \sum_{(i,j)\in(S,\bar{S})} x(i,j) - \sum_{(i,j)\in(\bar{S},S)} x(i,j) = \sum_{(i,j)\in(S,\bar{S})} x(i,j) - 0 = u[S,\bar{S}]$$

Si x^* es un flujo máximo y $(T, \bar{T})$ es una cortadura mínima, entonces por la Propiedad 3.2 tenemos que, $v^* \leq u[T, \bar{T}]$. Entonces,

$$v \leq v^* \leq u[T,\bar{T}] \leq u[S,\bar{S}] = v$$

esto quiere decir que $v = v^*$, es decir, x es un flujo máximo. $\diamond$

Con todo lo anterior, estamos listos para enunciar el teorema de Flujo Máximo-Cortadura Mínima de Ford y Fulkerson.

Teorema 3.2 (Teorema de Flujo Máximo - Cortadura Mínima)
En toda red, el valor del flujo máximo es igual a la capacidad de una cortadura mínima.

Demostración. Sea x un flujo máximo, por el Teorema 3.1 no existen rutas aumentantes. Sin embargo, la demostración del Teorema 3.1 implica la existencia de una cortadura mínima $[S, \bar{S}]$ tal que el flujo de cada arco

en $(S, \bar{S})$ es igual a su capacidad, y el flujo en cada arco de $(\bar{S}, S)$ es 0. Esto es,

$$\sum_{(i,j)\in(S,\bar{S})} x(i,j) = u[S,\bar{S}] \quad y \quad \sum_{(i,j)\in(\bar{S},S)} x(i,j) = 0.$$

Por lo que,

$$v = \sum_{(i,j)\in(S,\bar{S})} x(i,j) - \sum_{(i,j)\in(\bar{S},S)} x(i,j) = u[S,\bar{S}].$$

Como en la demostración del Teorema 3.1, si $(T, \bar{T})$ es una cortadura mínima, entonces

$$u[T,\bar{T}] \leq u[S,\bar{S}] = v \leq u[T,\bar{T}]$$

Con lo que, $v = u[T,\bar{T}]$. $\diamond$

3.1. Algoritmo de Flujo Máximo - Cortadura Mínima

La demostración del Teorema 3.1 sugiere un algoritmo que resuelve el problema de flujo máximo. En este trabajo presentaremos un algoritmo cuya idea básica de es dar un método sistemático para encontrar rutas aumentantes en una red partiendo de un flujo factible dado.

El algoritmo busca las rutas aumentantes más cortas, es decir, las que tienen el menor número de arcos. La validez del algoritmo se basa tanto en la demostración del Teorema 3.1 como en la del siguiente teorema:

Teorema 3.3 *Sea G una red con nodo origen s, nodo destino t y un flujo factible x. Sea G' una red con el mismo conjunto de nodos y el conjunto de arcos definido como sigue.*

$$A' = \{(i,j) \mid (i,j) \in A, \quad u(i,j) > x(i,j) \quad o \quad (j,i) \in A, \quad x(i,j) > 0\}.$$

Entonces, G' contiene una ruta dirigida $s-t$ si y sólo si G contiene una ruta aumentante. Además, una ruta más corta $s-t$ en G' tiene el mismo tamaño que una ruta aumentante más corta en G.

Demostración. Supongamos que G' contiene una ruta de $s-t$

$$P' : s = i_0, i_1, \ldots i_n = t$$

para cada k, $1 \leq k \leq n-1$, ya sea que tengamos arcos de la forma (i_{k-1}, i_k) o (i_k, i_{k-1}), estos arcos también pertenecen a A. Con lo que podemos decir

que $P : s = i_0, i_1, \ldots i_n = t$ es una ruta en G.

Nos falta demostrar que P es una ruta no saturada. Sea (i_{k-1}, i_k) un arco en P, entonces también es un arco en G' y además $u(i_{k-1}, i_k) > x(i_{k-1}, i_k)$. Por otro lado, sea (i_k, i_{k-1}), entonces $x(i_k, i_{k-1}) > 0$. De aquí que, P es una ruta no saturada de $s - t$ y por consecuencia una ruta aumentante. Además las rutas P y P' tienen el mismo tamaño.

Inversamente, supongamos que G contiene una ruta aumentante

$$P : s = i_0, i_1, \ldots, i_{n-1} i_n = t$$

Entonces, para cada $1 \leq k \leq n-1$:

- si un arco en P es de la forma (i_{k-1}, i_k) y $u(i_{k-1}, i_k) > x(i_{k-1}, i_k)$, entonces $(i_{k-1}, i_k) \in A'$
- si un arco en P es de la forma (i_k, i_{k-1}) y $x(i_k, i_{k-1}) > 0$, entonces $(i_{k-1}, i_k) \in A'$

entonces,

$$P' : s = i_0, i_1, \ldots, i_{n-1} i_n = t$$

es una ruta $s - t$ en G'. Además, P y P' tienen el mismo tamaño.

Con lo anterior podemos concluir que una ruta más corta $s - t$ en G' tiene el mismo tamaño que un ruta aumentante más corta en G. $\diamond$

Sea x un flujo en G, si x no es máximo, entonces G contiene una ruta aumentante, digamos

$$P : s = i_0, i_1, \ldots, i_{n-1} i_n = t$$

Definimos

$$\Delta(a_k) = \begin{cases} u(i_{k-1}, i_k) - x(i_{k-1}, i_k) & \text{si} \quad a_k = (i_{k-1}, i_k) \\ x(i_k, i_{k-1}) & \text{si} \quad a_k = (i_k, i_{k-1}) \end{cases}$$

y hacemos $\Delta = \min\{\Delta(a_k) \mid 1 \leq k \leq n\}$. Si a_j es un arco de P tal que $\Delta(a_j) = \Delta$, entonces a_j es llamado un **arco saturado** de G con respecto al flujo x y a la ruta aumentante P.

Sea x^* un flujo obtenido a partir de x como describimos en el Teorema 3.1:

$$x^*(i,j) = \begin{cases} x(i,j) + \Delta & \text{si} \quad i = i_{k-1} \text{ y } j = i_k \quad \text{para algún} \quad k,\ 1 \leq k \leq n; \\ x(i,j) - \Delta & \text{si} \quad i = i_k \text{ y } j = i_{k-1} \quad \text{para algún} \quad k,\ 1 \leq k \leq n; \\ x(i,j) & \text{si} \quad (i,j) \notin P. \end{cases}$$

Algoritmo 3.1 Algoritmo de Flujo Máximo - Cortadura Mínima.

1. Construir la gráfica G' con los nodos $N' = N$ y un conjunto de arcos definido como sigue:
   $A' = \{(i,j) \mid (i,j) \in A \text{ y } u(i,j) > x(i,j) \text{ o } (j,i) \in A \text{ y } x(j,i) > 0\}$

2. Aplicar el Algoritmo de Búsqueda a G' para determinar si existe una ruta (más corta) de s a t. Si no existe, entonces ir a paso 5; en otro caso, sea

   $$P' : s = i_0, i_1, \ldots, i_r = t$$

   P es una ruta más corta $s - t$ en G', continuar

3. Ejecutar un aumento. Sea $P : s = i_0, i_1, \ldots, i_r = t$, donde cada arco de la cadena es de la forma

   $$\begin{array}{lll} (i_{k-1}, i_k) & \text{y} & u(i_{k-1}, i_k) > x(i_{k-1}, i_k) \quad \text{o,} \\ (i_k, i_{k-1}) & \text{y} & x(i_k, i_{k-1}) > 0 \end{array}$$

   Para $k = 1, 2, \ldots, r$ sea

   $$\Delta_k \leftarrow \begin{cases} u(i_{k-1}, i_k) - x(i_{k-1}, i_k) & \text{si es de la forma} \quad (i_{k-1}, i_k) \\ x(i_k, i_{k-1}) & \text{si es de la forma} \quad (i_k, i_{k-1}) \end{cases}$$

   Hacer $\Delta \leftarrow \min\{\Delta_k \mid 1 \leq k \leq r\}$. Para $k = 1, 2, \ldots, r$, si un arco es de la forma (i_{k-1}, i_k) hacer:

   $$x(i_{k-1}, i_k) \leftarrow x(i_{k-1}, i_k) + \Delta$$

   para los demás arcos, (i_k, i_{k-1}), hacer:

   $$x(i_k, i_{k-1}) \leftarrow x(i_k, i_{k-1}) - \Delta$$

4. Regresar al Paso 1

5. Obtener el flujo máximo y determinar la cortadura mínima.
   Mostrar $x(i,j)$ para todo $(i,j) \in A$. Sea S el conjunto de nodos i de G' que recibieron etiquetas finitas en el Paso 2 después de aplicar el Algoritmo de Búsqueda a G'. Entonces $[S, \bar{S}]$ es una cortadura mínima.

Entonces decimos que x^* fue obtenido aumentando x a través de la ruta aumentante P. El método descrito queda especificado en el Algoritmo de Flujo Máximo - Cortadura mínima, Algoritmo 3.1. Este algoritmo parte de un flujo inicial, que puede ser el flujo cero.

El Algoritmo de Búsqueda utilizado en el paso 2 se describe en el Apéndice B.

3.2. Complejidad del algoritmo

Analicemos ahora la complejidad de este algoritmo. Sea G una gráfica con n nodos y m arcos. Podemos observar que el Paso 1 del Algoritmo 3.1 requiere revisar m arcos. En consecuencia, la complejidad del Paso 1 es de $O(m)$.

En el Apéndice B, mostramos que la complejidad del Paso 2 es de $O(m)$.

Para el Paso 3, primero calculamos Δ_k, para $1 \leq k \leq r$. Esto requiere a lo más $n-1$ cálculos ya que $r \leq n-1$. Por lo que hacemos $n-1$ comparaciones para encontrar Δ. Además, en este paso, actualizamos el flujo en cada uno de los arcos que pertenecen a P para producir un nuevo flujo. Ya que esto también requiere r operaciones, donde $r \leq n-1 \leq m$, la complejidad de todo el Paso 3 es $O(m)$.

Con lo anterior, podemos concluir que la complejidad para los Pasos 1-3 es de $O(m)$. Se puede demostrar que estos pasos son ejecutados a lo más $\frac{1}{2} \cdot n \cdot m$ veces, lo que significa que si cada aumento en el algoritmo de flujo máximo - cortadura mínima es hecho a través de rutas más cortas $s-t$, entonces obtenemos el flujo máximo a lo más después de $\frac{1}{2} \cdot n \cdot m$ aumentos. Con lo que se concluye que la complejidad de todo el algoritmo es de $O(n \cdot m^2)$.

La demostración en el número de aumentos se basa en el hecho de que cada que un mismo arco interviene en una ruta aumentante como arco saturación, el tamaño de la ruta más corta de $s-t$ aumenta en al menos 2 unidades. Como el tamaño de la ruta aumentante más corta tiene tamaño de al menos 1 y a lo más $n-1$, se tiene que

$$1 + 2(w-1) \leq d_w(w,t) \leq n-1$$

donde, w es el número de la última ruta aumentante en la cual aparece el arco (i,j) o el arco (j,i), $d_w(s,t)$ es la distancia de s a t después del w-*ésimo* aumento.

Con lo anterior tenemos que $w \leq n/2$, por lo que un mismo arco (ya sea (i,j) o (j,i)) puede aparecer a lo más en $n/2$ rutas aumentantes más cortas como un arco saturación. Cada ruta aumentante más corta tiene al menos un arco saturación. Entonces se necesitan a lo más $n \cdot m/2$ rutas aumentantes más cortas para encontrar el flujo máximo.

Este argumento muestra que para este Algoritmo de Flujo Máximo - Cortadura Mínima su complejidad es del $O(n \cdot m^2)$.

4. Algoritmo genérico para rutas aumentantes

Empezamos con uno de los algoritmos más simples e intuitivos para resolver el problema de flujo máximo. Este algoritmo es conocido como *algoritmo de rutas aumentantes*. Pero antes definamos algunos conceptos y retomemos algunas propiedades del capítulo anterior.

Red residual. Dado un flujo x, la capacidad residual r_{ij} de cualquier arco $(i,j) \in A$ es el máximo flujo adicional que puede ser enviado de i a j usando los arcos (i,j) y (j,i). La capacidad residual r_{ij} tiene dos componentes:

1. $u_{ij} - x_{ij}$, que es la capacidad no utilizada en el arco (i,j) y
2. x_{ji}, que se refiere al flujo actual de (j,i) el cual puede ser cancelado para incrementar el flujo de i a j.

Por consecuencia, $r_{ij} = u_{ij} - x_{ij} + x_{ji}$.

Nos referiremos a la red $G(x)$ constituida por arcos con capacidad residual positiva como la *red residual* respecto al flujo x.

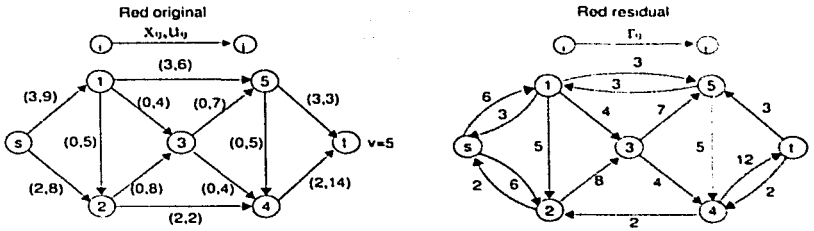

Figura 22: Ejemplo de una red residual a partir de un flujo x.

En la Figura 22 podemos ver un ejemplo de una red con un flujo de valor $v = 5$ y su respectiva red residual. Por ejemplo, la capacidad residual del par de nodos $s, 1$, es decir r_{s1}, la calculamos así: $r_{s1} = u_{s1} - x_{s1} + x_{1s} = 9 - 3 + 0 = 6$. De igual forma, para calcular r_{1s}, tenemos $r_{1s} = u_{1s} - x_{1s} + x_{s1} = 0 - 0 + 3 = 3$. Y aplicando la misma fórmula podemos calcular las capacidades residuales para cada par de nodos.

Capacidad residual de una *(s,t)*-cortadura. Definamos la capacidad residual $r[S, \bar{S}]$ de una (s,t)-cortadura como la suma de las capacidades residuales de los *arcos hacia adelante* de la cortadura. Esto es:

$$r[S, \bar{S}] = \sum_{(i,j)\in(S,\bar{S})} r_{ij}$$

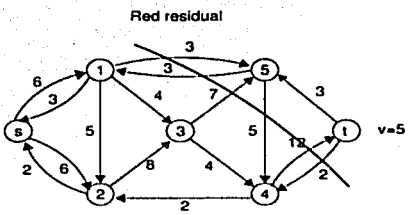

Figura 23: (s, t)-cortadura en una red residual.

Propiedad 4.1 *Para cualquier flujo x de valor v en una red, el flujo adicional que puede ser enviado del nodo origen al nodo destino es menor o igual a la capacidad residual de cualquier (s, t)-cortadura.*

Demostración. Supongamos que x es un flujo de valor v. Además suponga-mos que x' es un flujo de valor $v + \Delta v$ para algún $\Delta v \geq 0$. La Desigualdad (5) del capítulo anterior implica que:

$$v + \Delta v \leq \sum_{(i,j)\in(S,\bar{S})} u_{ij} \tag{6}$$

Restando la Ecuación (4) de la Ecuación (6) tenemos que:

$$\Delta v \leq \sum_{(i,j)\in(S,\bar{S})} (u_{ij} - x_{ij}) + \sum_{(i,j)\in(\bar{S},S)} x_{ij} \tag{7}$$

Usando el hecho de que

$$\sum_{(i,j)\in(\bar{S},S)} x_{ij} = \sum_{(i,j)\in(S,\bar{S})} x_{ji}$$

la desigualdad anterior nos queda:

$$\Delta v \leq \sum_{(i,j)\in(S,\bar{S})} (u_{ij} - x_{ij} + x_{ji}) = \sum_{(S,\bar{S})} r_{ij} \qquad \diamond$$

Utilizando el ejemplo de la Figura 23, en la cual tenemos una (s,t)-cortadura con $r[S,\bar{S}] = 3+7+12 = 22$, podemos observar que dado un flujo x de valor v, el flujo adicional que puede ser enviado del nodo origen al nodo destino es menor o igual a la capacidad de cualquier (s,t)-cortadura. En este caso para la Propiedad 4.1.

$$\Delta v \leq 22 = r[S,\bar{S}]$$

Lo que implica que 22 es una cota para el flujo adicional que puede ser enviado a partir del flujo actual y entonces el flujo máximo para nuestro ejemplo no puede ser mayor a 5+22=27 unidades.

Nos referiremos a una ruta dirigida que va del nodo origen al nodo destino en la red residual como una ***ruta aumentante***. Definimos la capacidad residual δ de una ruta aumentante como la mínima capacidad residual de entre los arcos que pertenecen a dicha ruta.

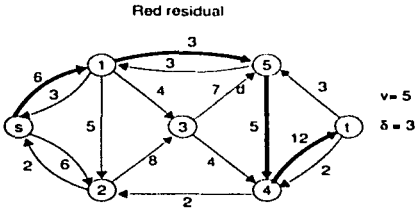

Figura 24: Ruta aumentante con $\delta = 3$

Por definición, la capacidad δ siempre es positiva. Consecuentemente, si la red contiene rutas aumentantes podemos enviar un flujo adicional del nodo origen al nodo destino. El algoritmo genérico de rutas aumentantes está basado esencialmente en esta observación, identifica rutas aumentantes y aumenta el flujo a través de éstas hasta que la red ya no contenga rutas aumentantes. El algoritmo está descrito en el Algoritmo 4.1.

Relación entre la red original y la red residual

Una ruta aumentante en la red original G es una ruta P, no necesariamente dirigida, de s a t tal que $x_{ij} < u_{ij}$ en cada arco hacia adelante (i,j) y $x_{ij} > 0$ en cada arco hacia atrás (i,j).

Supongamos que actualizamos las capacidades residuales en algún punto del algo-

Algoritmo 4.1 Algoritmo de rutas aumentantes

```
begin
  x ← 0;
  while G(x) contenga rutas dirigidas del nodo s al nodo t do
    Identificar una ruta aumentante P del nodo s al nodo t;
     δ ← mín{r_ij : (i,j) ∈ P};
    aumentar δ unidades de flujo a lo largo de P;
    actualizar G(x);
   end while;
end;
```

ritmo. ¿Cuál sería el efecto en x_{ij}? Recordemos que $r_{ij} = u_{ij} - x_{ij} + x_{ji}$, esto implica que un flujo adicional de δ unidades en el arco (i, j) en la red residual corresponde a:

1) un incremento en x_{ij} por δ unidades en la red original; o
2) un decremento en x_{ji} por δ unidades en la red original; o
3) una combinación de las anteriores.

Ahora bien, una vez dados los nuevos valores r_{ij} de la red residual, ¿ cómo podemos determinar el valor del flujo x_{ij}? Ya que para $r_{ij} = u_{ij} - x_{ij} + x_{ji}$ muchas combinaciones de x_{ij} y x_{ji} corresponden a un mismo valor de r_{ij} debemos tener una regla para escoger una de estas combinaciones.
Veamos que podemos reescribir

$$r_{ij} = u_{ij} - x_{ij} + x_{ji}$$

como

$$x_{ij} - x_{ji} = u_{ij} - r_{ij}.$$

Ahora, si $u_{ij} \geq r_{ij}$ entonces hacemos

$$x_{ij} = u_{ij} - r_{ij} \; ; \quad x_{ji} = 0$$

si $u_{ij} < r_{ij}$ hacemos

$$x_{ij} = 0 \quad \text{y} \quad x_{ji} = r_{ij} - u_{ij}.$$

En la Figura 25(a) observamos la red de la Figura 24 después de aplicar el aumento $\delta = 3$ a través de la ruta señalada. Así mismo en la Figura 25(b) observamos el resultado en la red original.

Existen diferentes formas de identificar una ruta aumentante en una red residual. La especificación de dicho proceso nos proporcionará una versión o especialización

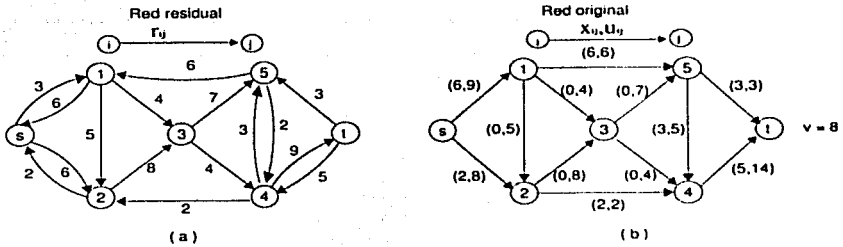

Figura 25: Red residual y red original actualizadas.

del algoritmo genérico.

En las siguientes secciones vamos a ver especializaciones de este algoritmo. Empezaremos presentando el algoritmo de etiquetamiento y realizaremos un análisis detallado del mismo.

4.1. Algoritmo de etiquetamiento

El *algoritmo de etiquetamiento* es una implementación del algoritmo genérico de rutas aumentantes. Este algoritmo usa una técnica de búsqueda para identificar en $G(x)$ una ruta dirigida de s a t, empieza desde el nodo origen y encuentra todos los nodos que son alcanzables desde el nodo s a lo largo de rutas dirigidas en la red residual. En cada paso del algoritmo se tiene una partición en la red: *los nodos etiquetados y los no etiquetados.*

Los nodos etiquetados son aquellos que el algoritmo ha alcanzado en el proceso de búsqueda y para los cuales se ha determinado una ruta dirigida del nodo origen a dichos nodos. Los no etiquetados son aquellos que el algoritmo no ha alcanzado aún.

El algoritmo selecciona iterativamente un nodo etiquetado y revisa sus arcos adyacentes, en la red residual, para alcanzar y etiquetar nuevos nodos. Eventualmente, el nodo destino llega a ser etiquetado y el algoritmo envía el máximo flujo posible a través de la ruta encontrada de s a t. Se borran las etiquetas y se repite este proceso. Este algoritmo termina cuando se han revisado todos los nodos etiquetados y el nodo destino permanece sin etiqueta, lo que implica que el nodo origen no está conectado al nodo destino en la red residual.

El código de una versión del Algoritmo de Etiquetamiento se muestra en el Algoritmo 4.3 mientras que en el Algoritmo 4.2 tenemos el procedimiento auxiliar que utiliza.

Algoritmo 4.2 Procedimiento aumenta

```
begin
  con las etiquetas de pred(j) obtén una ruta aumentante P de s a t;
  δ ← mín{r_ij : (i,j) ∈ P};
  aumenta δ unidades de flujo a lo largo de P;
  actualiza las capacidades residuales;
end;
```

Justificación del algoritmo de etiquetamiento y resultados

Para realizar una justificación al algoritmo de etiquetamiento, notemos que en cada iteración el algoritmo encuentra una ruta aumentante o termina al no poder etiquetar el nodo destino t.

Algoritmo 4.3 Algoritmo de etiquetamiento

```
begin
 etiqueta nodo t;
 while t esta etiquetado do
   desetiqueta todos los nodos;
   for each nodo j ∈ N do pred(j) ← 0;
   etiqueta nodo s;
   agrega s a LISTA;
   while LISTA ≠ ∅ o t no esta etiquetado do
    remueve un nodo i de LISTA;
    for each arco (i,j) en G(x) que emana del nodo i do
     if (r_ij > 0) y (nodo j sin etiqueta) then
      pred(j) ← i;
      etiqueta nodo j;
      agrega j a LISTA;
     end if;
    end for;
   end while;
   if t está etiquetado then aumenta;
 end while;
end;
```

En el último caso deberíamos mostrar que el flujo actual x es el flujo máximo. Supongamos en esta etapa que S es el conjunto de nodos etiquetados y $\bar{S} = N \setminus S$ es el conjunto de nodos no etiquetados. Claramente, $s \in S$ y $t \in \bar{S}$. Ya que el algoritmo no puede etiquetar ningún nodo en $\bar{S}$ desde cualquier nodo en S, $r_{ij} = 0$ para cada $(i,j) \in (S, \bar{S})$. Además, ya que $r_{ij} = (u_{ij} - x_{ij}) + x_{ji}$, $x_{ij} \leq u_{ij}$ y $x_{ji} \geq 0$ la condición $r_{ij} = 0$ implica que $x_{ij} = u_{ij}$ para cada arco $(i,j) \in (S, \bar{S})$ y $x_{ij} = 0$ para cada arco $(i,j) \in (\bar{S}, S)$.

Sustituyendo el valor de este flujo en la Ecuación (4), encontramos que:

$$v = \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(i,j)\in(\bar{S},S)} x_{ij} = \sum_{(i,j)\in(S,\bar{S})} u_{ij} = u[S,\bar{S}]$$

Esto muestra que el valor del flujo actual x es igual a la capacidad de la cortadura $[S, \bar{S}]$. Por la Propiedad 3.2 tenemos que x es un flujo máximo y $[S, \bar{S}]$ es una cortadura mínima. Esta conclusión establece que el algoritmo de etiquetamiento es correcto y deja como resultado el siguiente teorema de flujo máximo cortadura-mínima.

Teorema 4.1 (Flujo Máximo - Cortadura Mínima) *El valor máximo de un flujo desde un nodo origen s a un nodo destino t en una red con capacidades es igual a la capacidad mínima de entre todas las (s,t)-cortaduras.* ◇

Este teorema indica que cuando el algoritmo de etiquetamiento termina, también encuentra una cortadura mínima. Del algoritmo también se obtiene el siguiente resultado.

Teorema 4.2 (Rutas Aumentantes) *Un flujo x^* es un flujo máximo sí y sólo si la red residual $G(x^*)$ no contiene rutas aumentantes.*

Demostración.
⇒) Si la red residual $G(x^*)$ contiene una ruta aumentante, claramente x^* no es un flujo máximo.
⇐) Ahora bien, si la red residual $G(x^*)$ no contiene rutas aumentantes el conjunto de nodos S etiquetados por el algoritmo define una (s,t)-cortadura $[S,\bar{S}]$ cuya capacidad es igual al flujo, lo que implica que el flujo es máximo. ◇

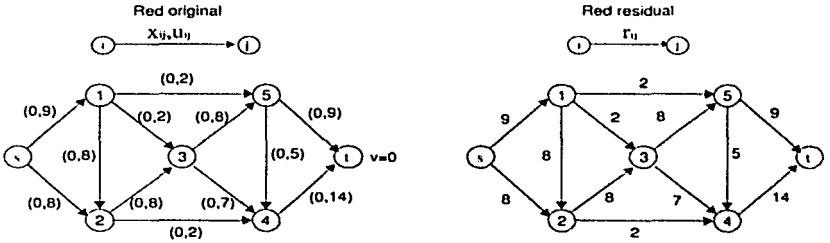

Figura 26: Ejemplo del algoritmo de etiquetamiento.

Teorema 4.3 *Si todas las capacidades de los arcos son enteros, el problema de flujo máximo tiene una solución entera.*

Demostración. Este resultado se sigue por un argumento de inducción aplicado al número de aumentos. Ya que el algoritmo de etiquetamiento inicia con un flujo cero y todas las capacidades de los arcos son enteras, las capacidades residuales iniciales también son todas enteras. El flujo aumentado en cualquier iteración es igual a la mínima capacidad residual de alguna ruta, la cual por hipótesis de inducción es entera. Consecuentemente, la

capacidad residual en la siguiente iteración será de nuevo entera. Como las capacidades residuales r_{ij} y las capacidades de los arcos u_{ij} son todas enteras, el flujo de los arcos x_{ij} serán valores enteros también. Además, si las capacidades son enteras en cada aumento se agrega al menos una unidad al valor del flujo; ya que el flujo máximo no puede exceder la capacidad de cualquier cortadura, el algoritmo termina en un número finito de iteraciones. ◇

Veamos un ejemplo aplicando el Algoritmo de Etiquetamiento. Tomemos la red de la Figura 26.

Ejecutando los pasos del algoritmo de etiquetamiento en la red residual, primero etiquetamos el nodo t, como t está etiquetado quitamos las etiquetas a todos los nodos, y asignamos 0 a todas las etiquetas de predecesores de cada nodo. Etiquetamos el nodo s y lo agregamos a $LISTA$, $LISTA = \{s\}$. Como $LISTA \neq \emptyset$, sacamos de $LISTA$ el nodo s y revisamos los nodos que salen de s, el nodo 1 emana de s y no tiene etiqueta además $r_{s1} > 0$, entonces etiquetamos 1, hacemos $pred(1) = s$, y agregamos 1 a $LISTA$, $LISTA = \{1\}$. Posteriormente se etiquetan los nodos como se muestra en la Tabla 27.

Nodo en revisión	Nodo etiquetado	predecesor	$LISTA$
s	1	s	1
s	2	s	1,2
1	3	1	2,3
1	5	1	2,3,5
2	4	2	3,5,4
5	t	5	4,t

Figura 27: Primera fase del algoritmo de etiquetamiento

El último etiquetamiento de la Tabla 27 significa que hemos encontrado una ruta aumentante P que va desde s a t, entonces ejecutamos el procedimiento *aumenta*. Con las etiquetas de predecesores recuperamos la ruta de $s - t$, en este caso, la ruta que encontramos es $P : s, 1, 5, t$. Comparamos las capacidades de los arcos relacionados con P y obtenemos el mínimo, $\delta = \min\{9, 2, 9\} = 2$. El siguiente paso sería aumentar δ unidades de flujo a través de P y acualizar las capacidades residuales. El resultado de estos pasos lo podemos ver en la Figura 28. Hasta este paso el valor del flujo es $v = 2$.

Siguiendo los pasos del algoritmo de etiquetamiento. El siguiente paso es desetiquetar

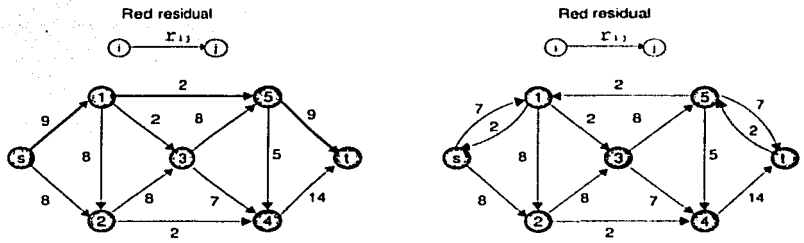

Figura 28: Ejemplo del algoritmo de etiquetamiento.

los nodos y volver a iniciar la búsqueda de otra ruta aumentante. En la Tabla de la Figura 30 mostramos las rutas aumentantes subsecuentes, así como el valor del flujo en cada momento, la Figura 29 y la Figura 31 ilustran la red resultante después de cada uno de estos aumentos.

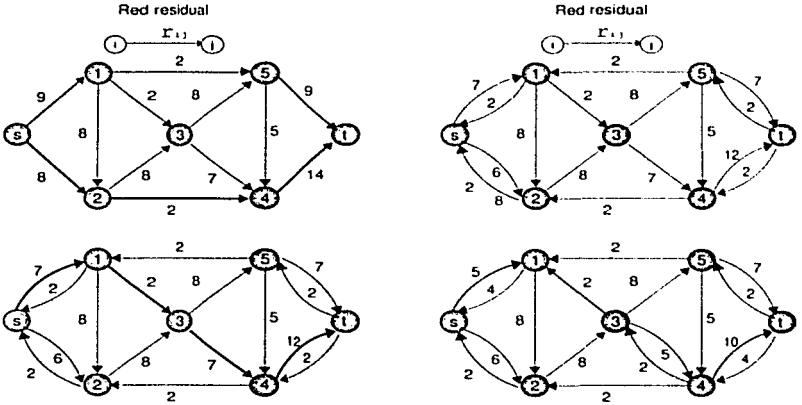

Figura 29: Aumentos del algoritmo de etiquetamiento.

Para este ejemplo, observamos que se requirieron seis aumentos o rutas aumentantes para alcanzar el flujo máximo $v = 14$. Además podemos verificar el Teorema de Flujo Máximo - Cortadura Mínima. En la Figura 32 se muestra la solución del

Número de ruta	Ruta aumentante encontrada	Valor de aumento (δ)	Valor del Flujo
2	$s, 2, 4, t$	2	4
3	$s, 1, 3, 4, t$	2	6
4	$s, 2, 3, 4, t$	5	11
5	$s, 2, 3, 5, t$	1	12
6	$s, 2, 3, 5, t$	2	14

Figura 30: Rutas que encuentra el algoritmo de etiquetamiento

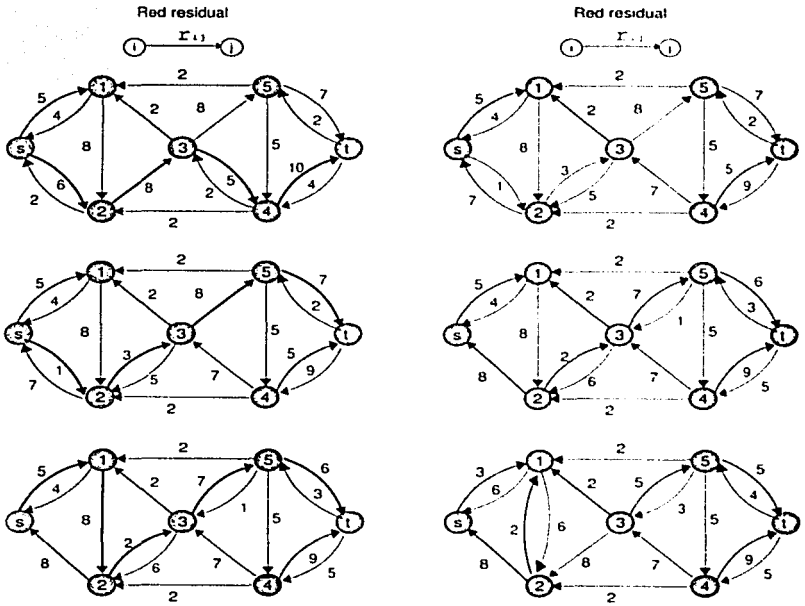

Figura 31: Aumentos del algoritmo de etiquetamiento. Continuación.

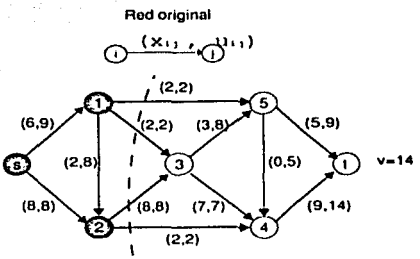

Figura 32: Solución al ejemplo.

ejemplo, donde el flujo máximo es $v = 14$ y la cortadura mínima es $S = [s, 1, 2]$ con $u[S, \bar{S}] = 2 + 2 + 8 + 2 = 14$.

Complejidad del Algoritmo de Etiquetamiento

Para estudiar el peor de los casos en el algoritmo de etiquetamiento recordemos que en cada iteración, excepto en la última, el algoritmo ejecuta un aumento. Es fácil observar que cada aumento requiere tiempo de $O(m)$ porque el método busca la cantidad δ de una cadena aumentante, examinando cualquier arco de cualquier nodo a lo más una vez. Por lo tanto, la complejidad del algoritmo es $O(m)$ veces el número de aumentos. Ahora bien, ¿cuántos aumentos puede ejecutar el algoritmo? Si todas las capacidades de los arcos son enteras y acotadas por un número finito U, la capacidad de la cortadura $(s, N \setminus \{s\})$ es a lo más nU. Por lo que el valor del flujo máximo está acotado por nU. El algoritmo de etiquetamiento incrementa el valor del flujo por al menos una unidad en cualquier aumento. Consecuentemente, $O(nmU)$ es una cota para el tiempo de ejecución de este algoritmo. Con lo anterior, tenemos el siguiente resultado.

Teorema 4.4 *El algoritmo de etiquetamiento resuelve el problema de flujo máximo en un tiempo de $O(nmU)$, en el peor de los casos.*

Desventajas del Algoritmo de Etiquetamiento

El algoritmo de etiquetamiento es posiblemente el algoritmo más simple para resolver el problema de flujo máximo, sin embargo en el peor de los casos el número de iteraciones no es muy satisfactorio para valores de U muy grandes. Por ejemplo, si $U = 2^n$, la cota es exponencial para el número de nodos. Además, el algoritmo podría ejecutar innecesariamente muchas iteraciones como nos sugiere el ejemplo de

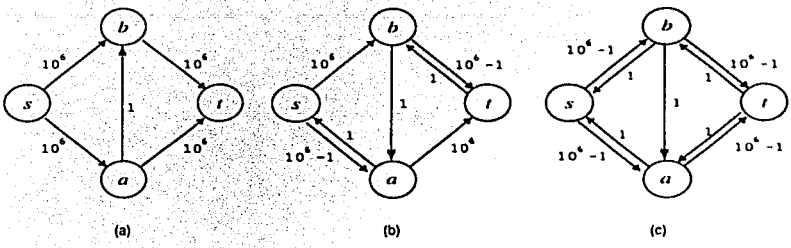

Figura 33: Caso patológico del algoritmo de etiquetamiento

la Figura 33. La Figura 33(a) muestra una red con flujo cero; en la Figura 33(b) vemos la red residual después de un aumento de 1 a lo largo de la ruta $s - a - b - t$ y la Figura 33(c) muestra la red residual después de un aumento de 1 a lo largo de $s - b - a - t$. Con lo que podemos ver que el algoritmo, como elige cualquier nodo para etiquetar, podría elegir la peor opción y así ejecutar aumentos innecesarios.

Otra desventaja del algoritmo es que si las capacidades de los arcos son irracionales, el algoritmo podría no terminar. Aunado a esto, el algoritmo no tiene memoria, en cada iteración genera etiquetas de nodos que contienen información sobre las posibles rutas aumentantes desde el nodo origen a otros nodos. En esta implementación que hemos descrito se borran las etiquetas en cada iteración, perdiendo así información que podría ser útil para las siguientes iteraciones.

En resumen, el algoritmo genérico de rutas aumentantes tiene dos limitaciones computacionales muy importantes:

1. en el peor de los casos la complejidad de $O(nmU)$ es poco práctica para ejemplares con capacidades muy grandes, y
2. para ejemplares con capacidades *irracionales* el algoritmo podría converger a una solución no óptima.

Por tal motivo se han desarrollado métodos con algunas adecuaciones para el mejor desempeño del algoritmo para el peor caso. En las siguientes secciones trataremos algunas mejoras o refinamientos del algoritmo genérico de rutas aumentantes.

4.2. Algoritmo de capacidades escalables

Veamos ahora el *algoritmo de rutas aumentantes de máxima capacidad*. Este algoritmo siempre aumenta flujo a través de una ruta con máxima capacidad residual.

Sea x cualquier flujo y v el valor del flujo. Si v^* es el valor del flujo máximo, por la propiedad de descomposición de flujo aplicado a la red residual $G(x)$ implica que podemos encontrar m o menos rutas dirigidas desde el nodo origen al nodo destino cuyas capacidades residuales sumen $(v^* - v)$. Entonces la ruta aumentante de máxima capacidad tiene capacidad residual al menos de $(v^* - v)/m$ unidades.

Ahora consideremos una secuencia consecutiva de $2m$ aumentos de máxima capacidad empezando del flujo x. Si cada uno de esos aumentos son al menos de $(v^* - v)/2m$ unidades de flujo, entonces dentro de $2m$ o menos iteraciones tendríamos un flujo máximo. Sin embargo, si alguno de esos $2m$ aumentos consecutivos carga menos de $(v^* - v)/2m$ unidades de flujo, entonces partiendo del vector de flujo inicial x, hemos reducido la capacidad de la ruta aumentante de máxima capacidad por un factor de al menos 2. Este argumento muestra que dentro de $2m$ iteraciones consecutivas, el algoritmo establece un flujo máximo o reduce la capacidad residual de la ruta aumentante de máxima capacidad por un factor de al menos 2. Ya que la capacidad residual de cualquier ruta aumentante es a lo más $2U$ y al menos 1, después de $O(m \log U)$ iteraciones, el flujo debería ser máximo.

Como hemos visto, el algoritmo de rutas aumentantes de máxima capacidad reduce el número de aumentos con respecto al algoritmo de etiquetamiento de $O(nU)$ a $O(m \log U)$. Sin embargo, el algoritmo ejecuta más cálculos por iteración ya que necesita identificar una ruta aumentante con la máxima capacidad residual, no cualquier ruta aumentante. En su lugar veremos una variación del algoritmo de rutas aumentantes de máxima capacidad que no ejecuta más cómputos por iteración y aún así establece un flujo máximo en tiempo $O(m \log U)$. Este algoritmo es conocido como: *algoritmo de capacidades escalables*. El cual es ilustrado en el Algoritmo 4.4.

La idea esencial en el algoritmo de capacidades escalables es conceptualmente muy simple: aumentamos flujo a través de una ruta con capacidad residual suficientemente grande, en lugar de una con máxima capacidad aumentante, esto es porque podemos obtener una ruta con capacidad residual suficientemente grande de una manera sencilla, en un tiempo de $O(m)$. Para definir el algoritmo de capacidades escalables introduzcamos un parámetro Δ y, con respecto a un flujo x, definamos la *Δ-red residual* como la red que contiene arcos cuya capacidad residual es al menos Δ. Sea $G(x, \Delta)$ la Δ-red residual, observemos que $G(x, 1) = G(x)$ y $G(x, \Delta)$ es una subgráfica de $G(x)$. En la Figura 34 ilustramos esta definición. Para la red residual de la Figura 34(a), la Δ-red residual con $\Delta = 8$ se muestra en la Figura 34(b).

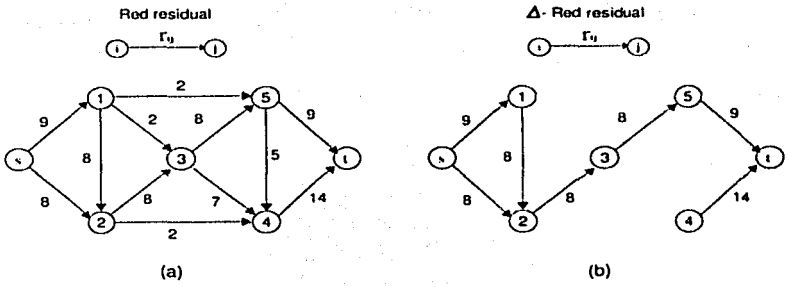

Figura 34: Ejemplo de una Δ-red Residual.

A la fase del algoritmo durante la cual Δ se mantiene constante le llamaremos *fase de escalamiento* y a una fase de escalamiento con un valor específico Δ como *Δ-fase de escalamiento*. Observemos que en una Δ-fase de escalamiento cada aumento es de al menos Δ unidades de flujo. El algoritmo empieza con $\Delta = 2^{\lfloor \log U \rfloor}$ y divide su valor en cada fase de escalamiento hasta que $\Delta = 1$. En consecuencia el algoritmo ejecuta $1 + \lfloor \log U \rfloor = O(\log U)$ fases de escalamiento. En la última fase de escalamiento, $\Delta = 1$, entonces $G(x, \Delta) = G(x)$, lo que muestra que el algoritmo termina con un flujo máximo.

Algoritmo 4.4 Algoritmo de capacidades escalables

```
begin
  x ← 0;
  Δ ← 2^⌊logU⌋;
  while Δ ≥ 1 do
     while G(x, Δ) contiene rutas de s a t do
        identifica una ruta P en G(x, Δ);
        δ ← min{r_ij : (i, j) ∈ P};
        aumenta δ unidades de flujo por P;
        actualiza G(x, Δ);
     end while;
     Δ ← Δ/2;
  end while;
end;
```

La eficiencia del algoritmo depende del hecho de que ejecuta a lo más $2m$ aumentos por fase de escalamiento. Consideremos el flujo al final de la Δ-fase de escalamiento. Sea x' este flujo y v' su valor, sea S el conjunto de nodos alcanzables del nodo s en $G(x', \Delta)$. Ya que $G(x', \Delta)$ no contiene rutas aumentantes del nodo origen al nodo destino, $t \notin S$, entonces $[S, \bar{S}]$ forma una (s,t)-cortadura . La definición de S implica que la capacidad residual de cada arco en $[S, \bar{S}]$ es estrictamente menor que Δ, por lo que la capacidad de la cortadura $[S, \bar{S}]$ es a lo más $m\Delta$. Por consecuencia, $v^* - v' \leq m\Delta$. En la siguiente fase de escalamiento cada aumento lleva al menos $\Delta/2$ unidades de flujo, entonces esta fase de escalamietno puede ejecutar a lo más $2m$ aumentos.

El algoritmo de etiquetamiento requiere un tiempo de $O(m)$ para identificar una ruta aumentante, y la actualización de la red residual también requiere un tiempo de $O(m)$. Estos argumentos nos lleva a establecer el siguiente resultado.

Teorema 4.5 *El algoritmo de capacidades escalables resuelve el problema de flujo máximo en $O(m \log U)$ aumentos y se ejecuta en un tiempo de $O(m^2 \log U)$, en el peor de los casos* $\diamond$

Ahora bien, a manera de comparación utilizaremos el mismo problema del ejemplo del algoritmo de etiquetamiento para ejemplificar y comparar el algoritmo de capacidades escalables.

Iniciamos el algoritmo partiendo de un flujo $x = 0$, y asignamos el valor $\Delta = 2^{\lfloor log U \rfloor}$. Para la red de la 34(a) $\Delta = 2^{\lfloor log 14 \rfloor} = 8$, entonces generamos la Δ-red residual $G(x, 8)$ resultando la subgráfica de la Figura 34(b).

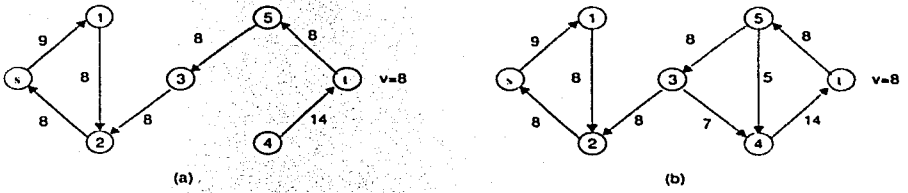

Figura 35: Actualización de $G(x, \Delta)$ para $\Delta = 8$

El siguiente paso es encontrar una ruta de s a t en $G(x, 8)$. Utilizando el algoritmo de etiquetamiento de la sección anterior encontramos la primer ruta P de s a t,

$P : s - 2 - 3 - 5 - t$ y hacemos $\delta = \min\{8, 8, 8, 9\} = 8$. Con lo que el aumento que podemos efectuar es de 8 unidades. Actualizamos $G(x, 8)$, quedando como en la Figura 35(a). Como puede observarse, al actualizar $G(x, \Delta)$ después del aumento en este caso, no existen rutas dirigidas de s a t, lo que significa que hemos terminado una Δ-fase de escalamiento. Hacemos $\Delta = \Delta/2 = 4$, y buscamos una nueva ruta de s a t en $G(x, 4)$, Figura 35(b). Nuevamente, podemos ver que en $G(x, 4)$ no hay rutas de s a t por lo que deberemos actualizar de nuevo el valor de Δ y por lo tanto cambiar de fase.

Ahora, hacemos $\Delta = \Delta/2 = 2$ y actualizamos $G(x, \Delta)$, Figura 36(a). Otra vez, utilizando el algoritmo de etiquetamiento tratamos de encontrar una ruta de s a t. La segunda ruta encontrada es $P : s - 1 - 2 - 4 - t$, para la cual $\delta = \min\{9, 8, 2, 14\} = 2$, aumentamos dos unidades de flujo por P y actualizamos $G(x, \Delta)$, en la Figura 36(b) tenemos $G(x, \Delta)$ para $\Delta = 2$, después de este aumento de flujo de dos unidades.

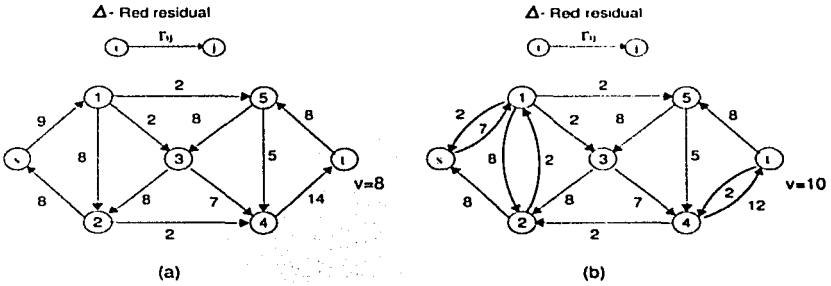

Figura 36: Actualización de $G(x, \Delta)$ para $\Delta = 2$.

Aplicamos de nuevo el algoritmo para encontrar otra ruta aumentante. La siguiente ruta es $P : s - 1 - 3 - 4 - t$ con $\delta = \min\{7, 2, 7, 12\} = 2$ y aumentamos el flujo a través de P, quedando la red como en la Figura 37(a). La última ruta aumentante en este ejemplo es $P : s - 1 - 5 - 4 - t$ con la cual obtenemos un aumento $\delta = \min\{5, 2, 5, 10\} = 2$. Al actualizar $G(x, 2)$ después de aumentar este flujo y aplicar el algoritmo de etiquetamiento verificamos que ya no existen rutas de s a t, por lo que hemos terminado otra fase, la red de la Figura 37(b) nos muestra $G(x, \Delta)$ para $\Delta = 2$ en este paso. Hacemos $\Delta = \Delta/2 = 1$. En este punto $G(x, 1) = G(x)$ y en el ejemplo, en particular, ya no existen rutas de s a t en la red residual, lo que indica que el flujo es máximo.

Como pudimos observar, este algoritmo encuentra rutas aumentantes con capacidad

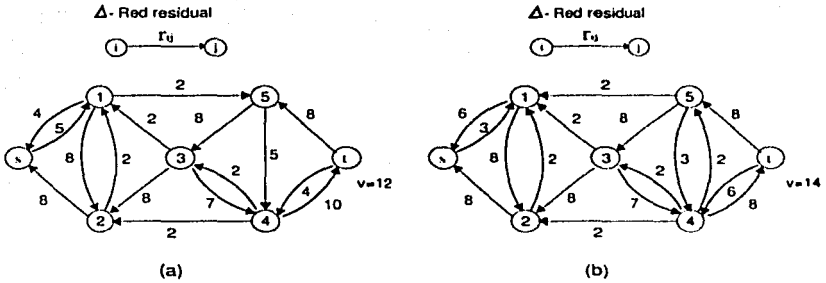

Figura 37: Otra actualización de $G(x, \Delta)$ para $\Delta = 2$.

residual suficientemente grandes, intentando enviar grandes cantidades de flujo y tratar de llegar al flujo máximo en un tiempo más corto. Comparando con el algoritmo de etiquetamiento, en este algoritmo se requirieron solamente cuatro aumentos para llegar al flujo máximo, en cambio en el algoritmo de etiquetamiento se requirieron seis aumentos. Por lo que podemos darnos cuenta de que el algoritmo de capacidades escalables es mejor que el de etiquetamiento para este caso; y en general así es.

4.3. Algoritmo de rutas aumentantes más cortas

El *algoritmo de rutas aumentantes más cortas* siempre aumenta flujo a través de rutas más cortas del nodo origen al nodo destino en la red residual. Antes de ver una descripción del algoritmo, definamos algunos conceptos que utilizaremos más adelante.

Una *función de distancia* $d : N \rightarrow Z^+ \cup \{0\}$ con respecto a las capacidades residuales r_{ij} es una función que va del conjunto de nodos a los enteros no negativos. Diremos que una función de distancia es válida con respecto al flujo x si satisface las siguientes condiciones:

$$d(t) = 0; \tag{8}$$

$$d(i) \leq d(j) + 1 \qquad \text{para todo } (i,j) \text{ en la red residual } G(x) \tag{9}$$

Nos referiremos a $d(i)$ como la etiqueta de distancia del nodo i y a las condiciones (8) y (9) como *condiciones de validación*.

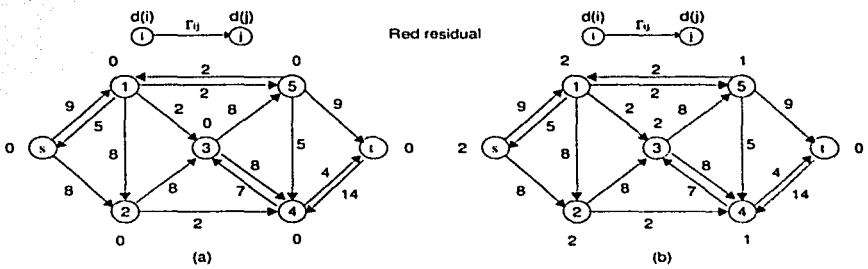

Figura 38: Red residual con etiquetas de distancia.

En la red residual de la Figura 38 tenemos que $d = (0,0,0,0,0,0,0)$ es un arreglo de etiquetas de distancia válidas ya que cumple con las condiciones 8 y 9. También $d = (2,2,2,2,1,1,0)$ es válido, en cambio $d = (2,2,2,2,2,1,0)$ no representa etiquetas de distancia válidas ya que $2 = d(4) > d(t) + 1 = 0 + 1$ para el arco $(4,t)$.

Propiedad 4.2 *Si las etiquetas de distancia son válidas, la etiqueta $d(i)$ es una cota inferior en el tamaño de la ruta dirigida más corta del nodo origen al nodo destino en la red residual.*

Demostración. Sea $i = i_1 - i_2 - i_3 - \ldots - i_k - i_{k+1} = t$ cualquier ruta de tamaño k desde el nodo i al nodo t en la red residual, por las condiciones de validación tenemos que:

$$\begin{gathered} d(i_k) \leq d(i_{k+1}) + 1 = d(t) + 1 = 1 \\ d(i_{k-1}) \leq d(i_k) + 1 \leq 2 \\ d(i_{k-2}) \leq d(i_{k-1}) + 1 \leq 3 \\ \vdots \\ d(i) = d(i_1) \leq d(i_2) + 1 \leq k \end{gathered}$$ ◇

Como k, que es el tamaño de la ruta, cumple que $d(i) \leq k$ y además la ruta que utilizamos en la demostración es cualquiera, se aplica a la ruta más corta, lo que significa que el tamaño de una ruta más corta que va de un nodo i a t, es mayor o igual al valor de la etiqueta $d(i)$, siempre que las etiquetas sean válidas.

En la Figura 39 tenemos una red residual en donde al lado de cada nodo se encuentra su etiqueta de distancia, la cual representa el tamaño de la ruta más corta desde ese nodo al nodo t, $d(3, 2, 2, 2, 1, 1, 0)$. Como podemos ver las etiquetas de distancia válidas de las dos redes de la Figura 38 son menores o iguales una a una con las etiquetas de la Figura 39, la red es la misma en ambas figuras.

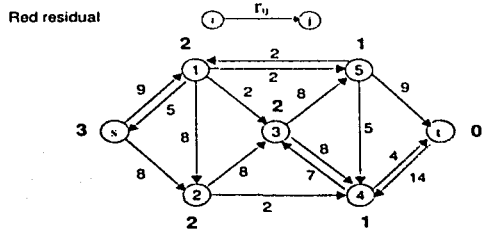

Figura 39: Red residual con etiquetas de distancia exactas.

Propiedad 4.3 *Si $d(s) \geq n$, entonces la red residual no contiene rutas dirigidas de s a t.*

Justificación. Lo anterior se sigue del hecho de que $d(s)$ es una cota inferior para el tamaño de la ruta más corta desde s a t en la red residual, y ya que ninguna ruta dirigida puede contener más de $(n-1)$ arcos, entonces si $d(s) \geq n$, la red residual no contiene rutas dirigidas desde el nodo s al nodo t.

Notación adicional. Decimos que las etiquetas de distancia son *exactas* si para cada nodo i, $d(i)$ es igual al tamaño de la ruta más corta que va desde el nodo i hasta el nodo t en la red residual. Podemos determinar etiquetas de distancia exactas para todos los nodos en un tiempo de $O(m)$ empezando desde el nodo t.
De aquí en adelante al hablar de etiquetas nos referiremos a etiquetas de distancia exactas.

Arcos y rutas admisibles

Diremos que un arco (i, j) en la red residual es *admisible* si satisface la condición: $d(i) = d(j) + 1$; nos referiremos a todos los demás arcos como *no admisibles*. Una ruta de s a t que consiste solamente de arcos admisibles es una *ruta admisible*.

Propiedad 4.4 *Una ruta admisible es una ruta aumentante más corta del nodo origen al nodo destino en la red residual.*

> Justificación. Ya que cada arco (i, j) en una ruta admisible P es admisible, la capacidad residual de estos arcos y las etiquetas de distancia de sus nodos satisfacen que:
>
> 1. $r_{ij} > 0$, esto implica que P es una ruta aumentante
> 2. $d(i) = d(j) + 1$, implica que si P contiene k arcos, entonces $d(s) = k$
>
> Ya que $d(s)$ es una cota inferior sobre el tamaño de cualquier ruta desde el nodo origen al nodo destino en la red residual, la ruta P debe ser una ruta aumentante más corta.

Descripción del algoritmo

Las rutas a través de las cuales el algoritmo aumenta flujo son rutas admisibles, éstas se construyen incrementalmente agregando un arco a la vez. El algoritmo conserva una ruta admisible parcial, es decir, una ruta de s a algún nodo i que consiste solamente de arcos admisibles e iterativamente ejecuta una operación de avance o retroceso desde el último nodo de la ruta admisible parcial, nos referiremos a dicho nodo como *nodo actual*.

Si el nodo actual i tiene un arco admisible (i, j), ejecutamos una operación de avance y agregamos el arco (i, j) a la ruta admisible parcial; si no contiene un arco incidente que sea admisible entonces ejecutamos una operación de retroceso y regresamos un arco. Repetimos esa operación hasta que la ruta admisible parcial alcanza el nodo t, en ese momento aumentamos el flujo. Lo anterior se repite hasta que el flujo es máximo. En el Algoritmo 4.5 mostramos el pseudocódigo de este algoritmo y en el Algoritmo 4.6 los procedimientos que utiliza.

Algoritmo 4.5 Algoritmo de rutas aumentantes más cortas

```
begin
  x ← 0;
  obtenga las etiquetas de distancia exactas d(i);
  i ← s;
  while d(s) < n do
     if i tiene un arco admisible then
        avanza(i);
        if i = t then
          aumenta;
          i ← s;
        end if;
     else
        retrocede(i);
     end if;
  end while;
end;
```

Algoritmo 4.6 Procedimientos para el algoritmo de rutas aumentantes más cortas

```
Procedimiento avanza(i);
begin
  sea (i, j) un arco admisible en A(i);
  pred(j) ← i;
  i ← j;
end;

Procedimiento retrocede(i);
begin
  d(i) ← mín{d(i) + 1 : (i, j) ∈ A(i) y r_ij > 0};
  if i ≠ s then i ← pred(i);
end;

Procedimiento aumenta;
begin
  usa las etiquetas pred(j) para obtener una ruta aumentante P desde s a t;
  δ ← mín{r_ij : (i, j) ∈ P};
  aumtenta δ unidades de flujo a lo largo de P;
end;
```

Justificación al algoritmo

Veamos que el algoritmo verdaderamente resuelve el problema de flujo máximo.

Lema 4.1 *El algoritmo de rutas aumentantes más cortas conserva etiquetas de distancia válidas en cada paso. Mejor aún, cada operación de re-etiquetamiento (o retroceso) incrementa la etiqueta de distancia de un nodo.*

Demostración. Mostraremos que el algoritmo conserva etiquetas de distancia válidas en cada paso, ya sea en una operación de aumento o re-etiquetamiento. En la operación de avance no se afecta lo admisible de ningún arco porque éste no cambia ninguna capacidad residual o etiqueta de distancia. Inicialmente, el algoritmo construye etiquetas de distancia válidas. Se asume, inductivamente, que las etiquetas de distancia son válidas antes de una operación, es decir, que satisfacen las condiciones de validez. Necesitamos verificar si esas condiciones se mantienen válidas en los siguientes casos:

1. Después de una operación de aumento; y
2. Después de una operación de re-etiquetamiento (o retroceso).

1. Un aumento de flujo en el arco (i, j) podría eliminarlo de la red residual, esta modificación no afecta la validez de las etiquetas de distancia para este arco. Sin embargo, un aumento en el arco (i, j) también podría crear un arco adicional (j, i) con $r_{ji} > 0$ y entonces crear una desigualdad adicional $d(j) \leq d(i) + 1$ que las etiquetas de distancia deberían satisfacer. Las etiquetas de distancia satisfacen esta condición, ya que $d(i) = d(j) + 1$ por que la ruta aumentante es admisible.

2. La operación de re-etiquetamiento (o retroceso) modifica $d(i)$; por lo tanto debemos mostrar que cada arco que entra y sale del nodo i satisface las condiciones de validación con respecto a las nuevas etiquetas de distancia, $d'(i)$. El algoritmo ejecuta una operación de re-etiquetamiento al nodo i cuando ya no tiene arcos admisibles; esto es ningún arco $(i, j) \in A$ satisface las condiciones: $d(i) = d(j) + 1$ y $r_{ij} > 0$. Esta observación, en términos de condiciones de validación $d(i) \leq d(j) + 1$, implica que $d(i) < d(j) + 1$ para todos los arcos $(i, j) \in A$ con capacidad residual positiva. Por lo que $d(i) < \min\{d(j) + 1 : (i, j) \in A, r_{ij} > 0\} = d'(i)$, la cual es la nueva etiqueta de distancia después de la operación de re-etiquetamiento. Hemos mostrado entonces que el re-etiquetamiento preserva las condiciones de validación para todos los arcos que emanan del nodo i, y que cada operación de re-etiquetamiento incrementa el valor de $d(i)$. Finalmente, cada

nodo entrante (k, i) satisface $d(k) \leq d(i) + 1$, por hipótesis de inducción. Ya que $d(i) < d'(i)$, el re-etiquetamiento una vez más preserva las condiciones de validación para el arco (k, i). $\diamond$

El algoritmo de rutas aumentantes más cortas termina cuando $d(s) \geq n$ indicando que la red no contiene rutas aumentantes del nodo origen al nodo destino. Por consecuencia, el flujo obtenido al final del algoritmo es un flujo máximo. Obtenemos, entonces, el siguiente resultado:

Teorema 4.6 *El algoritmo de rutas aumentantes más cortas calcula correctamente el flujo máximo.* $\diamond$

Complejidad del algoritmo

El algoritmo de rutas aumentantes más cortas se ejecuta en un tiempo de $O(n^2m)$. Para mostrar lo anterior vamos a describir una estructura de datos usada para seleccionar un arco admisible proveniente de un nodo dado. Llamaremos a esta estructura, *estructura de datos del arco actual*. Tenemos la lista de arcos $A(i)$ la cual contiene los arcos que salen de i, podemos ordenar arbitrariamente los arcos en esa lista, una vez establecido el orden se mantiene sin cambios durante el algoritmo. Cada nodo i tiene un *arco actual*, el cual también pertenece a $A(i)$ y es el próximo candidato a la prueba de admisibilidad. Inicialmente el arco actual del nodo i es el primer arco en $A(i)$. Siempre que el algoritmo intente encontrar un arco admisible que emana del nodo i, prueba si el arco actual del nodo es admisible. Si no, designa el siguiente arco dentro $A(i)$ como el arco actual. El algoritmo repite este proceso hasta que encuentra un arco admisible o alcanza el final de la lista.
Consideremos el caso en el que el algoritmo alcanza el final de la lista de arcos sin encontrar un arco admisible. ¿Podemos decir que $A(i)$ no tiene arcos admisibles? Sí, porque es posible mostrar que si un arco (i, j) no es admisible en iteraciones anteriores, sigue siendo no admisible hasta que $d(i)$ se incremente. Entonces, si alcanzamos el final de la lista de arcos, ejecutamos el re-etiquetamiento (o retroceso) y de nuevo ponemos como arco actual del nodo i el primer arco de la lista $A(i)$. La operación de re-etiquetamiento también examina una vez cada arco en $A(i)$ para calcular las nuevas etiquetas de distancia, lo cual gasta el mismo tiempo que en identificar arcos admisibles en el nodo i en una revisión de la lista de arcos. De lo antrior se obtiene la siguiente propiedad.

Propiedad 4.5 *Si el algoritmo de rutas aumentantes más cortas etiqueta cualquier nodo a lo más k veces, el total de tiempo utilizado en encontrar arcos admisibles y re-etiquetar nodos es*

$$O\left(k \cdot \sum_{i \in N} |A(i)|\right) = O(k \cdot m)$$

Lema 4.2 *Si el algoritmo re-etiqueta cualquier nodo a lo más k veces, el algoritmo satura arcos, es decir, reduce su capacidad residual a cero, a lo más $\lfloor k \cdot m/2 \rfloor$ veces.*

Demostración. Mostraremos que entre dos saturaciones consecutivas de un arco (i, j), tanto $d(i)$ y $d(j)$ deben incrementarse al menos dos unidades cada una. Ya que por hipótesis el algoritmo incrementa cada etiqueta de distancia a lo más k veces, este resultado implica que el algoritmo podría saturar cualquier arco a lo más $k/2$ veces. Por esto el número de saturaciones de arcos podría ser $\lfloor k \cdot m/2 \rfloor$ como dice el lema.

Supongamos que un aumento satura un arco (i, j). Como el arco (i, j) es admisible, tenemos que,

$$d(i) = d(j) + 1. \tag{10}$$

Antes de que el algoritmo sature este arco de nuevo, debería enviar flujo de regreso del nodo j al nodo i. En este momento, las etiquetas de distancia $d'(i)$ y $d'(j)$ satisfacen la ecuación

$$d'(j) = d'(i) + 1. \tag{11}$$

En la siguiente saturación del arco $(i.j)$, deberíamos tener

$$d''(i) = d''(j) + 1. \tag{12}$$

Usando las Ecuaciones (10) y (11) en (12) observamos que:

$$d''(i) = d''(j) + 1 \geq d'(j) + 1 = d'(i) + 2 \geq d(i) + 2.$$

Similarmente es posible mostrar que $d''(j) \geq d(j) + 2$.

Como resultado, entre dos saturaciones consecutivas del arco (i, j), tanto $d(i)$ y $d(j)$ incrementan al menos dos unidades. Con lo que se concluye la demostración del lema. ◇

Lema 4.3

1. *En el algoritmo de rutas aumentantes más cortas cada etiqueta de distancia incrementa a lo más n veces. Por consecuencia, el total de re-etiquetamientos (o retrocesos) es a lo más n^2.*

2. *El número de operaciones de aumento es a lo más $n \cdot m/2$.*

   Demostración. Cada operación de re-etiquetamiento al nodo i incrementa el valor de $d(i)$ al menos en 1 unidad. Después de que el algoritmo ha re-etiquetado el nodo i a lo más n veces, $d(i) \geq n$. En este punto el algoritmo no vuelve a seleccionar el nodo i durante una operación de avance ya que para cada nodo k en la ruta parcial admisible, $d(k) < d(s) < n$. Entonces el algoritmo re-etiqueta un nodo a lo más n veces y el total de operaciones de re-etiquetamientos está acotado por n^2. Utilizando el Lema 4.2 y éste, tenemos que el algoritmo satura a lo más $n \cdot m/2$ arcos. Ya que cada aumento satura al menos un arco inmediatamente se tiene que el número de aumentos está acotado por $n \cdot m/2$. ◇

Teorema 4.7 *El algoritmo de rutas aumentantes más cortas se ejecuta en un tiempo de $O(n^2m)$, en el peor de los casos.*

Demostración. Utilizando el Lema 4.3 y la Propiedad 4.5 encontramos que el esfuerzo requerido en encontrar arcos admisibles y en re-etiquetar los nodos es de $O(nm)$. El Lema 4.3 implica que el número total de aumentos es de $O(nm)$. Ya que cada aumento requiere tiempo $O(n)$, el tiempo total para una operación de aumento es de $O(n^2m)$. Cada operación de retroceso re-etiqueta un nodo, por lo que el número total de operaciones de retroceso es de $O(n^2)$. Por otra parte, cada operación de avance agrega un arco a la ruta admisible parcial y cada operación de retroceso borra un arco de ésta. Como cada ruta admisible parcial tiene tamaño n a lo más, el algoritmo requiere a lo más $O(n^2 + n^2m)$ operaciones de avance. El primer término proviene del número de operaciones de retroceso (re-etiquetamiento), el segundo término se refiere al número de operaciones de aumento. La combinación de esas cotas establece el teorema. ◇

Veamos ahora un ejemplo utilizando este algoritmo de rutas aumentantes más cortas. Utilicemos la red de la Figura 40.

Iniciamos el algoritmo partiendo de un flujo $x = 0$ y obtenemos las etiquetas de distancia exactas, $d(i)'s$, Figura 40, derecha. Hacemos $i = s$, como la etiqueta $d(s) = 3$ es menor que n, buscamos si s tiene un arco admisible. El arco $(s, 1)$ es admisible,

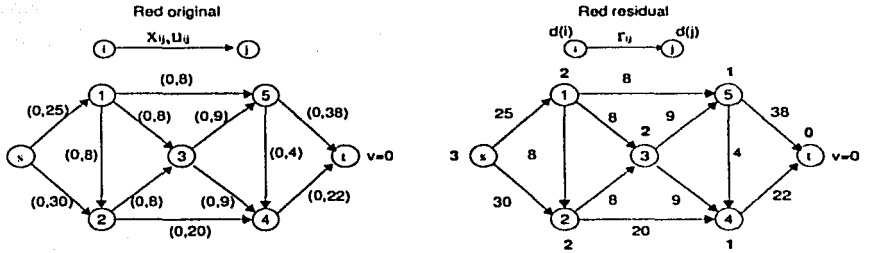

Figura 40: Ejemplo para el algoritmo de rutas aumentantes más cortas.

ya que $r_{s1} > 0$ y $d(s) = d(1) + 1$, entonces lo agregamos a la ruta parcial admisible mediante el procedimiento $avanza(i)$ haciendo $pred(1) = s$ y tomando al nodo 1 como el nodo i. Como $1 \neq t$ y $d(s) < n = 7$, revisamos si el nodo 1 tiene arcos admisibles, el arco $(1, 5)$ es admisible entonces ejecutamos el procedimiento $avanza(i)$, con esto $pred(5) = 1$ y $i = 5$.

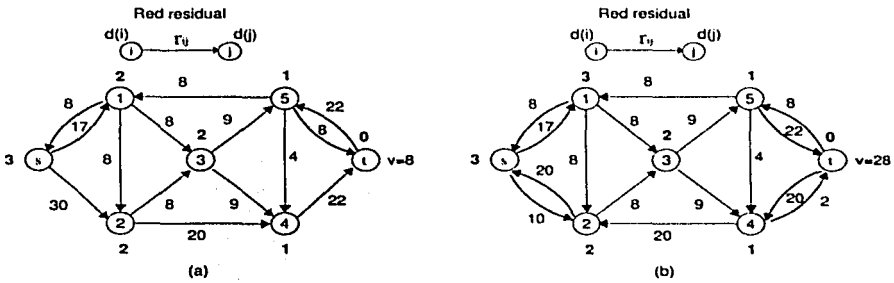

Figura 41: Actualización de Redes residuales.

Nuevamente, como $5 \neq t$ y $d(s) < n$, buscamos si es que el nodo 5 tiene arcos admisibles. Sí, el arco $(5, t)$ es admisible, el cual también será agregado a la ruta parcial mediante el procedimeinto $aumenta(i)$. Entonces $pred(t) = 5$ y $i = t$. Observemos que hemos alcanzado el nodo t por lo que debemos ejecutar el proceso $aumenta$, primero, con las etiquetas de predecesores reconstruimos la ruta de s a t que encontramos, $P : s, 1, 5, t$ y calculamos $\delta = \min\{25, 8, 38\} = 8$. Aumentamos 8 unidades a través de P, actualizamos la red residual. La Figura 41(a) muestra la red residual después

de este primer aumento de 8 unidades. Hacemos $i = s$ y empezamos de nuevo.

i	Arco admisible	Proceso	Operaciones	$d(s)$	Valor del flujo
s	$(s,2)$	$avanza(i)$	$pred(2) = s$, $i = 2$	3	8
2	$(2,4)$	$avanza(i)$	$pred(4) = 2$, $i = 4$	3	8
4	$(4,t)$	$avanza(i)$	$pred(t) = 4$, $i = t$	3	8
t	———	$aumenta$	$i = s$	3	28
s	$(s,2)$	$avanza(i)$	$pred(2) = s$, $i = 2$	3	28
2	———	$retrocede(i)$	$d(2) = 3$, $i = pred(2) = s$	3	28
s	———	$retrocede(i)$	$d(s) = 4$	4	28
s	$(s,1)$	$avanza(i)$	$pred(1) = s$, $i = 1$	4	28
1	$(1,3)$	$avanza(i)$	$pred(3) = 1$, $i = 3$	4	28
3	$(3,4)$	$avanza(i)$	$pred(4) = 3$, $i = 4$	4	28
4	$(4,t)$	$avanza(i)$	$pred(t) = 4$, $i = t$	4	28
t	———	$aumenta$	$i = s$, Figura 43(a)	4	30
s	$(s,1)$	$avanza(i)$	$pred(1) = s$, $i = 1$	4	30
1	$(1,3)$	$avanza(i)$	$pred(3) = 1$, $i = 3$	4	30
3	$(3,4)$	$avanza(i)$	$pred(4) = 3$, $i = 4$	4	30
4	———	$retrocede(i)$	$d(4) = 3$, $i = pred(4) = 3$	4	30
3	$(3,5)$	$avanza(i)$	$pred(5) = 3$, $i = 5$	4	30
5	$(5,t)$	$avanza(i)$	$pred(t) = 5$, $i = t$	4	30
t	———	$aumenta$	$i = s$, Figura 43(b)	4	36
s	$(s,1)$	$avanza(i)$	$pred(1) = s$, $i = 1$	4	36
1	———	$retrocede(i)$	$d(1) = 4$, $i = pred(1) = s$	4	36
s	$(s,2)$	$avanza(i)$	$pred(2) = s$, $i = 2$	4	36
2	$(2,3)$	$avanza(i)$	$pred(3) = 2$, $i = 3$	4	36
3	$(3,5)$	$avanza(i)$	$pred(5) = 3$, $i = 5$	4	36
5	$(5,t)$	$avanza(i)$	$pred(t) = 5$, $i = t$	4	36
t	———	$aumenta$	$i = s$, Figura 44(a)	4	39

Figura 42: Iteraciones para el ejemplo.

Como $d(s) < n$, veamos si s tiene un arco admisible. El arco $(s,1)$ es admisible, por lo que ejecutamos una operación de $avanza(i)$, con lo que hemos agregado este arco a la ruta parcial y además $pred(1) = s$, $i = 1$. Ya que $i \neq t$ entonces no hay ruta de s a t, $d(s)$ sigue siendo menor que n. Ahora toca el turno del nodo 1 para verificar la existencia de arcos admisibles. Encontramos que el nodo 1 no tiene arcos admisibles y entoces ejecutamos un proceso $retrocede(i)$. Esto nos lleva a modificar la etiqueta de distancia del nodo $i = 1$, hacemos $d(1) = \min\{3+1, 2+1, 2+1\} = 3$ que corresponden a las etiquetas de distancia de los nodos para los cuales $r_{ij} > 0$, en este caso, son los

nodos s, 2 y 3 respectivamente. Como $1 \neq s$, entonces $i = pred(1) = s$ con lo que nos regresamos un nodo.

En la Tabla 42 mostramos las siguientes ejecuciones del algoritmo de rutas aumentantes más cortas. La red residual después del segundo aumento de flujo esta ilustrado en la Figura 41(b). El aumento al final de la Tabla 42 es el último que ejecuta el algoritmo.

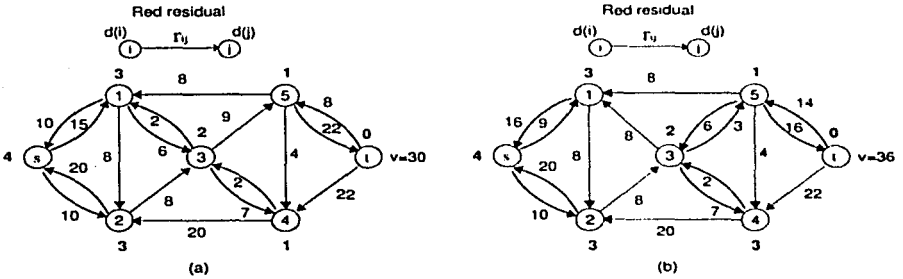

Figura 43: Red residual, 3a y 4a iteración.

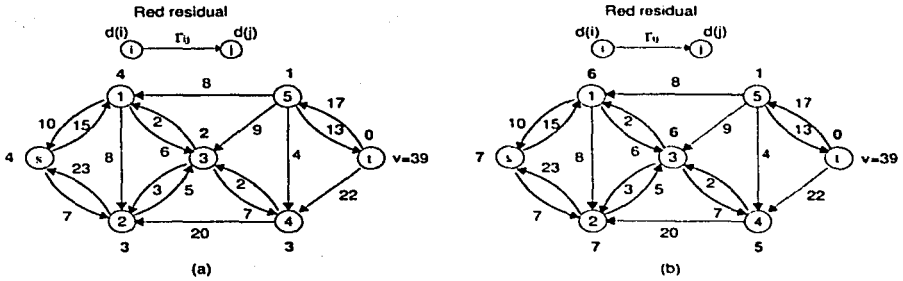

Figura 44: Últimas redes residuales

La Figura 43 muestra las redes residuales después de (a) la tercera iteración y (b) la cuarta iteración. En la Figura 44 tenemos las últimas redes residuales generadas por el algoritmo. En la Figura 44(b) podemos observar que $d(s) = n$ por lo que el

algoritmo termina.

i	Arco admisible	Proceso	Operaciones	$d(s)$	Valor del flujo
s	$(s,2)$	$avanza(i)$	$pred(2) = s$, $i = 2$	4	39
2	$(2,3)$	$avanza(i)$	$pred(3) = 2$, $i = 3$	4	39
3	———	$retorcede(i)$	$d(3) = 4$, $i = pred(3) = 2$	4	39
2	———	$retorcede(i)$	$d(2) = 5$, $i = pred(2) = s$	4	39
s	———	$retorcede(i)$	$d(s) = 5$	5	39
s	$(s,1)$	$avanza(i)$	$pred(1) = s$, $i = 1$	5	39
1	———	$retorcede(i)$	$d(1) = 6$, $i = pred(1) = s$	5	39
s	———	$retorcede(i)$	$d(s) = 6$	6	39
s	$(s,2)$	$avanza(i)$	$pred(2) = s$, $i = 2$	6	39
2	$(2,3)$	$avanza(i)$	$pred(3) = 2$, $i = 3$	6	39
3	$(3,4)$	$avanza(i)$	$pred(4) = 3$, $i = 4$	6	39
4	———	$retorcede(i)$	$d(4) = 5$, $i = pred(4) = 3$	6	39
3	———	$retorcede(i)$	$d(3) = 6$, $i = pred(3) = 2$	6	39
2	———	$retorcede(i)$	$d(2) = 7$, $i = pred(2) = s$	6	39
s	———	$retorcede(i)$	$d(s) = 7$	7	39

Figura 45: Continuación

En la Tabla 45 se muestran los siguientes pasos, en los cuales ya no se genera ningún aumento en el flujo, es decir, que ya no existen rutas aumentantes en la red residual. En la última fila de esta tabla tenemos que la etiqueta $d(s) = n = 7$; el cual es el criterio que usa el algoritmo para decidir que ha terminado.

Con este ejemplo podemos observar que aunque el algoritmo ya ha encontrado el flujo máximo, éste continúa su ejecución hasta que la etiqueta $d(s) \geq n$.

Una mejora práctica

El algoritmo de rutas aumentantes más cortas termina cuando $d(s) \geq n$. Este criterio es satisfactorio para el peor de los casos pero podría no ser eficiente en la práctica. El algoritmo gasta mucho tiempo en re-etiquetar nodos y una gran parte de este esfuerzo es realizado después de que el algoritmo ha establecido un flujo máximo. Esto sucede porque el algoritmo no sabe que ha encontrado un flujo máximo. Una técnica propuesta es capaz de detectar la presencia de una cortadura mínima y por lo tanto la existencia de un flujo máximo mucho antes de que la etiqueta del nodo s satisfaga la condición $d(s) \geq n$. Incorporando esta técnica al algoritmo, éste mejora substancialmente en la práctica.

Para implementar esta técnica debemos mantener un arreglo $n-dimensional$, $numb$, cuyos índices varían de 0 a $(n-1)$. El valor $numb(k)$ es el número de nodos cuyas etiquetas de distancia son iguales a k. El algoritmo da valores iniciales a este arreglo mientras calcula las etiquetas de distancia inicial usando una búsqueda. En este punto, las entradas positivas del arreglo son consecutivas, es decir, las entradas $numb(0)$, $numb(1)$, ..., $numb(l)$ serán positivas para algún índice l y el resto de las entradas serán cero. Subsecuentemente, siempre que el algoritmo incrementa la etiqueta de distancia de un nodo de k_1 a k_2, éste sustrae 1 de $numb(k_1)$, agrega 1 a $numb(k_2)$ y revisa si $numb(k_1) = 0$. Si $numb(k_1)$ es cero, el algoritmo termina. Para ver porque este criterio funciona, sea $S = \{i \in N : d(i) > k_1\}$ y $\bar{S} = \{i \in N : d(i) < k_1\}$. Es fácil ver que $s \in S$ y $t \in \bar{S}$. Ahora consideremos la (s,t)-cortadura, $[S, \bar{S}]$. La definición de los conjuntos de S y $\bar{S}$ implican que $d(i) > d(j) + 1$ para todos los arcos $(i,j) \in [S, \bar{S}]$. La Condición (9) de validación implica que $r_{ij} = 0$ para cada arco $(i,j) \in [S, \bar{S}]$. Por lo tanto $[S, \bar{S}]$ es una cortadura mínima y el flujo actual es el máximo.

5. Algoritmo genérico de preflujo

En este capítulo hablaremos de una clase de algoritmos para resolver el problema de flujo máximo conocidos como: algoritmos de empuje - preflujo. Este tipo de algoritmos son más generales, más poderosos y más flexibles que los algoritmos de rutas aumentantes. Actualmente el mejor algoritmo de empuje - preflujo supera al mejor de los algoritmos de rutas aumentantes tanto en la teoría como en la práctica.

La desventaja de los algoritmos de rutas aumentantes es el tiempo que se requiere cuando se envía flujo a lo largo de una ruta, que en el peor de los casos requiere un tiempo de $O(n)$. Los algoritmos de empuje - preflujo no tienen esta desventaja. Para entender esto mejor, veamos el ejemplo (extremo) de la Figura 46. Si aplicamos a este ejemplo cualquier algoritmo de rutas aumentantes, estos encontrarán siete rutas aumentantes cada una de tamaño siete que a su vez aumentarían una unidad de flujo cada una de ellas. Notemos cada una de esas rutas tienen en común los primeros cinco arcos y cada una de ellas recorre cada uno de estos arcos. Si pudieramos llevar siete unidades del nodo 1 al nodo 6, y luego enviar una unidad a través de siete rutas distintas de tamaño dos, podríamos evitarnos operaciones repetitivas al recorrer un conjunto de arcos en común. Esta es la idea esencial de los algoritmos de preflujo.

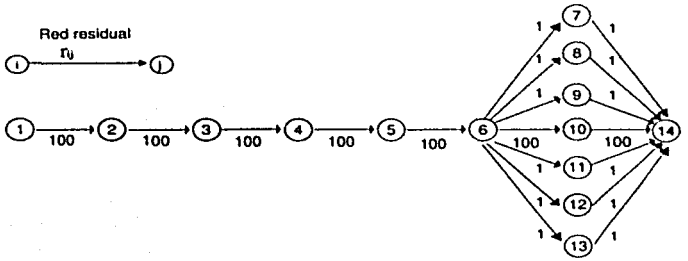

Figura 46: Desventaja del algoritmo de rutas aumentantes.

La operación básica para los algoritmos de rutas aumentantes es aumentar el flujo a lo largo de rutas que van de s a t, esta operación se puede descomponer en operaciones aún más elementales como enviar flujo a través de arcos individuales. Por ejemplo, enviar flujo de δ unidades a lo largo de una ruta de k arcos se puede descomponer en k operaciones básicas enviando un flujo de δ unidades a lo largo de cada arco de la ruta. Nos referiremos a cada una de esas operaciones como un *empuje*.

Los algoritmos de empuje - preflujo envían flujo por arcos individuales en lugar de hacerlo por rutas aumentantes, lo que provoca que, en etapas intermedias del algoritmo, no se satisfaga la ley conservación de flujo. De hecho, se permite que el flujo que entra a un nodo exceda al flujo que sale del mismo nodo.
Un *preflujo* es una función $x : A \rightarrow R$ que satisface:

$$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} \geq 0 \text{ para } i \in N - \{s,t\}$$

$$0 \leq x_{ij} \leq u_{ij} \text{ para cada } (i,j) \in A$$

El algoritmo de empuje - preflujo mantiene un preflujo en cada etapa intermedia. Para un preflujo dado definimos el *exceso* de cada nodo $i \in N$ como:

$$e(i) = \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij}$$

En un preflujo, $e(i) \geq 0$ para cada $i \in N - \{s,t\}$. Incluso, como no hay arcos que emanen del nodo t en el algoritmo de preflujo, también $e(t) \geq 0$; por lo que el nodo s es el único nodo con exceso negativo.

Nos referiremos a un nodo con exceso estrictamente positivo como un nodo *activo* y por convención tanto el nodo origen s como el nodo destino t nunca son activos. El algoritmo de rutas aumentantes siempre mantiene la factibilidad de la solución y busca la optimalidad. En contraste, el algoritmo de preflujo busca lograr la factibilidad. En el algoritmo de preflujo, la presencia de nodos activos indica que la solución no es factible. Consecuentemente, la operación básica del algoritmo es seleccionar un nodo activo y tratar de eliminar su exceso enviando flujo a los nodos vecinos que estén más cerca del nodo destino ya que lo que deseamos es enviar flujo de s a t. Al igual que en el algoritmo de rutas aumentantes, podemos medir lo cerca que está un nodo con las etiquetas de distancia, por lo que enviar flujo a través de nodos cercanos al nodo destino es equivalente a enviar flujo a través de arcos admisibles. Por lo anterior, enviamos flujo solamente a través de arcos admisibles. Si el nodo actual que estamos considerando no tiene arcos admisibles, incrementamos su etiqueta de distancia de tal forma que se genere al menos un arco admisible. El algoritmo termina cuando la red no contiene nodos activos.
El algoritmo de empuje - preflujo usa las rutinas de preproceso y empuje/re-etiqueta las cuales se muestran en el Algoritmo 5.1.

Un envío de δ unidades del nodo i al nodo j provoca un decremento en $e(i)$ y en r_{ij} de δ unidades y al mismo tiempo, un incremento en $e(j)$ y r_{ji} de δ unidades.

Algoritmo 5.1 Rutinas utilizadas en el algoritmo genérico de preflujo.

Procedimiento *preproceso*;
begin
  $x \leftarrow 0$;
  calcula las etiquetas de distancia exactas $d(i)$;
  $x_{sj} \leftarrow u_{sj}$ **para cada arco** $(s,j) \in A(s)$;
  $d(s) \leftarrow n$;
end;

Procedimiento *empuje/re − etiqueta*(i);
begin
  if la red contiene un arco admisible (i,j) **then**
    envía $\delta \leftarrow \min\{e(i), r_{ij}\}$ **unidades de flujo del nodo** i **al nodo** j
  else
    $d(i) \leftarrow \min\{d(j) + 1 : (i,j) \in A(i)$ **y** $r_{ij} > 0\}$;
  end if;
end;

Decimos que un envío de δ unidades de flujo en un arco (i,j) es un envío que *satura* si $\delta = r_{ij}$ y que *no satura* en otro caso. Un envío que no satura hecho desde del nodo i reduce $e(i)$ a cero. Nos referiremos a la operación de incrementar una etiqueta de distancia como una operación de *re-etiquetamiento*. El propósito de la operación de re-etiquetamiento es crear al menos un arco admisible en el cual el algoritmo pueda realizar futuros envíos. La versión genérica del algoritmo de empuje de preflujo se muestra en el Algoritmo 5.2.

Algoritmo 5.2 Algoritmo genérico de preflujo

begin;
  preproceso;
  while la red contiene un nodo activo do
    selecciona un nodo activo i;
    empuje/re-etiqueta(i);
  end while;
end;

La operación de preproceso realiza varias tareas importantes. Primero, le da un exceso positivo a cada nodo adyacente al nodo s, por lo que el algoritmo bien puede empezar seleccionando alguno de estos nodos con exceso positivo. Segundo, ya que el preproceso satura todos los arcos que inciden en s, ninguno de ellos es admisible, y además asigna $d(s) = n$, lo que satisface la Condición (9) de distancia válidas.

Tercero, como $d(s) = n$ la Propiedad 4.3 implica que la red residual no contiene rutas dirigidas del nodo s al nodo t. Ya que las etiquetas de distancia son no decrecientes, podemos garantizar que en iteraciones subsecuentes la red residual no tendrá rutas dirigidas de s a t, por lo que no se enviará flujo de s a t de nuevo.

A manera de ilustrar el algoritmo, veamos el ejemplo de la Figura 47(a). En la Figura 47(b) se muestra el preflujo generado por la operación de *preproceso*, todos los arcos que salen del nodo origen han sido saturados y para ellos se ha generado un exceso, además $d(s) = 4 = n$.

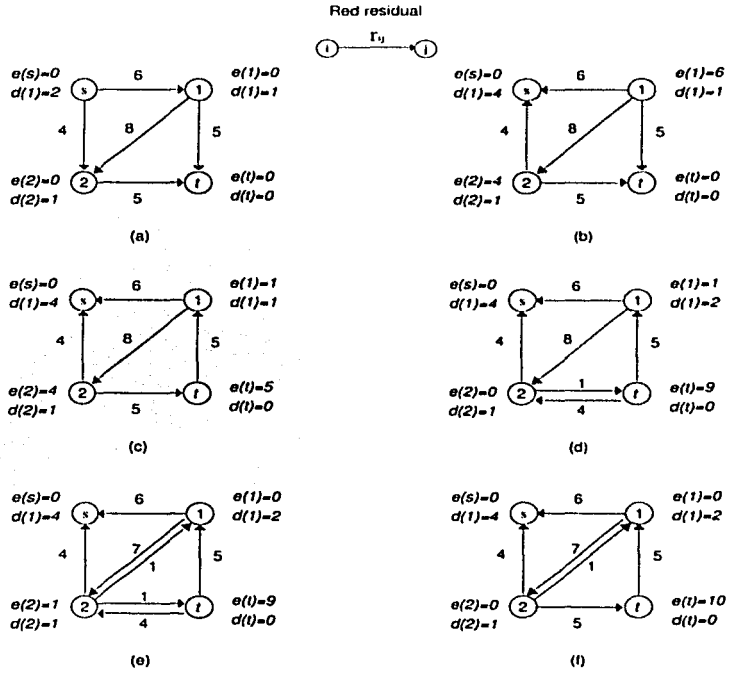

Figura 47: Ejemplo del algoritmo genérico de empuje - preflujo.

En la siguiente iteración supongamos que el algoritmo selecciona al nodo 1 de entre los nodos activos y ejecuta una operación de *empuje / re-etiqueta* sobre este nodo. El arco $(1,t)$ es el único arco admisible, por lo que enviamos $\delta = \text{mín}\{e(1), r_{1t}\} = \text{mín}\{6,5\} = 5$. Este es un envío que satura. Figura 47(c).

Observemos que el nodo 1 aún continua con exceso, supongamos que el algoritmo selecciona de nuevo el nodo 1 y ejecuta la operación de *empuje / re-etiqueta*. Como el nodo 1 no tiene arcos admisibles, se actualiza su etiqueta de distancia (operación de re-etiquetamiento), $d(1) = \text{mín}\{d(s)+1, d(2)+1\} = \text{mín}\{5,2\} = 2$. El resultado de esta operación sólo cambia la etiqueta del nodo 1, lo demás se conserva igual.

Ahora supongamos que el algoritmo elige al nodo 2, que es activo. El único arco admisible del nodo 2 es $(2,t)$, entonces realizamos un envío de $\delta = \text{mín}\{e(2), r_{2t}\} = \text{mín}\{4,5\} = 4$. Este es un envío que no satura, y el nodo 2 deja de ser activo. El resultado de este envío lo vemos en la Figura 47(d).

En el siguiente paso, el algoritmo toma el nodo 1 que se quedó aún como nodo activo. Veamos que podemos enviar un flujo a través del arco (1,2) de $\delta = \text{mín}\{e(1), r_{12}\} = \text{mín}\{1,8\} = 1$, el cual es un envío que no satura y el nodo 1 deja de ser activo. Quedando la red como en la Figura 47(e).

El algoritmo toma entonces el nodo 2 que se volvió activo y ejecuta un envío saturante sobre el arco admisible $(2,t)$ de valor $\delta = \text{mín}\{e(2), r_{2t}\} = \text{mín}\{1,1\} = 1$. Como se muestra en la Figura 47(f). En este momento ya no existen nodos activos en la red, por lo que el algoritmo termina su ejecución.

Suponiendo que el algoritmo genérico de empuje de preflujo termina, podemos mostrar que se ha encontrado el flujo máximo. El algoritmo termina cuando los excesos se ubican en el nodo origen o en el nodo destino, lo que implica que el preflujo actual es un flujo. Como $d(s) = n$, la red residual no contiene rutas dirigidas de s a t. Esta condición es el criterio para finalizar el algoritmo de rutas aumentantes, y entonces el exceso en el nodo destino es el valor del flujo máximo.

Complejidad del algoritmo

Para analizar la complejidad del algoritmo, empezaremos por establecer un resultado importante: las etiquetas de distancia se mantienen válidas y no se incrementan "muchas" veces. La primera de estas conclusiones se sigue del Lema 4.1, porque, al igual que en el algoritmo de rutas aumentantes más cortas, el algoritmo de preflujo solamente envía flujo a través de arcos admisibles y solamente re-etiqueta un nodo cuando no hay arcos admisibles que emanen de él. La segunda conclusión se sigue del

siguiente lema.

Lema 5.1 *En cualquier etapa del algoritmo de empuje - preflujo, cada nodo i con exceso positivo es conectado al nodo s por una ruta dirigida de i a s en la red residual.*

> *Demostración*. Veamos que para un preflujo x, $e(s) \leq 0$ y $e(i) \geq 0$ para toda $i \in N - \{s\}$. Por el teorema de descomposición de flujo, podemos descomponer cualquier preflujo x en la red original G en flujos no negativos a través de: (1) rutas del nodo s a t; (2) rutas del nodo s a nodos activos; y (3) flujo alrededor de ciclos dirigidos. Sea i un nodo activo relativo al preflujo x en G, la descomposición de flujo de x debe contener una ruta P de s a i, ya que las rutas de s a t y los flujos alrededor de ciclos no contribuyen al exceso del nodo i. La red residual contiene la inversa de P, es decir, P con la orientación de cada arco invertida, por lo tanto una ruta dirigida de i a s. ◇

Este lema implica que durante una operación de re-etiquetamiento, el algoritmo no trabaja sobre un conjunto vacío.

Lema 5.2 *Para cada nodo $i \in N$, $d(i) < 2n$.*

> *Demostración*. La última vez que el algoritmo re-etiqueta un nodo i, el nodo tenía un exceso positivo, por lo que la red residual contenía una ruta P a lo más de tamaño de $n - 2$ del nodo i a s. El hecho de que $d(s) = n$ y que $d(k) \leq d(l) + 1$ para todo arco (k, l) en P implica que $d(i) \leq d(s) + |P| < 2n$. ◇

Ya que cada vez que el algoritmo re-etiqueta un nodo i, $d(i)$ incrementa al menos una unidad, tenemos el siguiente resultado.

Lema 5.3 *Cada etiqueta de distancia incrementa a lo más $2n$ veces. Consecuentemente el número total de operaciones de re-etiquetamiento es a lo más $2n^2$.* ◇

Lema 5.4 *El algoritmo ejecuta a lo más $(n \cdot m)$ saturaciones.*

> *Demostración*. Este resultado se sigue directamente de los Lemas 4.2 y el 5.2. Por el Lema 4.2 si el algoritmo re-etiqueta un nodo a lo más k veces, entonces se saturan arcos a lo más $k \cdot m/2$ veces. Además el Lema 5.2 menciona que la etiqueta de distancia se incrementa a lo más a $2n$. Combinando estos dos Lemas concluimos que el algoritmo ejecuta a lo más $(2n \cdot m/2) = n \cdot m$ saturaciones. ◇

Tomando la Propiedad 4.5, el Lema 5.3 implica que el total de tiempo que se requiere para identificar arcos admisibles y ejecutar operaciones de re-etiquetamiento es $O(nm)$. Ahora veremos el número de envíos que no saturan que ejecuta el algoritmo.

Lema 5.5 *El algoritmo de empuje de preflujo ejecuta del $O(n^2m)$ envíos que no saturan, en el peor de los casos.*

Demostración. Para demostrarlo utilizaremos un argumento basado en potenciales. Sea I el conjunto de nodos activos. Consideremos la función potencia $\Phi = \sum_{i \in I} d(i)$. Como $|I| < n$, y $d(i) < 2n$ para toda $i \in I$, el valor inicial de Φ, después de la operación de preproceso, es a lo más $2n^2$. Cuando el algoritmo termina Φ es cero. Durante el algoritmo puede suceder uno de los siguientes casos:

Caso 1. El algoritmo no es capaz de encontrar un arco admisible por el cual pueda enviar flujo. En este caso la etiqueta de distancia del nodo i se incrementa por $\epsilon \geq 1$ unidades. Esta operación incrementa Φ por a lo más ϵ unidades. Ya que el incremento total en $d(i)$ para cada nodo i durante la ejecución del algoritmo está acotado por $2n$, el total de incremento en Φ que se debe al incremento en las etiquetas de distancia está acotado por $2n^2$.

Caso 2. El algoritmo es capaz de identificar un arco por el cual puede enviar flujo, y este envío puede ser que sature o que no sature. Un envío que satura sobre un arco (i, j) podría crear un nuevo exceso en el nodo j, con lo que se incrementaría el número de nodos activos en 1 y también aumentaría Φ en $d(j)$, donde $d(j)$ es a lo más $2n$ por saturación, dando un incremento total de $2n^2m$ unidades por todos los envíos que saturan. Ahora veamos, un envío que no satura sobre un arco (i, j) no incrementa $|I|$. Uno de estos envíos disminuye el valor de Φ en $d(i)$ ya que i deja de ser activo, pero al mismo tiempo crece el valor de Φ en $d(j) = d(i) - 1$; si el envío causa que el nodo j se convierta en activo el decremento total en Φ se vuelve 1. Si el nodo j ya era activo antes de el envío, Φ disminuye en $d(i)$. En conclusión, la disminución total en Φ es al menos 1 por cada envío que no satura. Sumando los resultados anteriores, el valor inicial de Φ es a lo más $2n^2$ y puede incrementar como máximo $2n^2 + 2n^2 \cdot m$. Cada envío que no satura disminuye a Φ en al menos una unidad. Por lo anterior, el algoritmo puede ejecutar a lo más la siguiente cantidad de envíos que no saturan $2n^2 + 2n^2 + 2n^2 \cdot m = O(n^2 \cdot m)$. $\diamond$

Finalmente, veamos como es que el algoritmo mantiene pista de los nodos activos para la operación de re-etiquetamento. El algoritmo puede mantener un conjunto LIST de nodos activos. Agrega a LIST los nodos que se convierten en nodos activos después de un envío de flujo y que no estaban en LIST, y borra de LIST los nodos que dejan de ser activos debido a un envío que no satura. Varias estructuras de datos (por ejemplo, listas doblemente ligadas) pueden ser utilizadas para LIST y así que el algoritmo pueda agregar, eliminar, o selecccionar elementos de la lista en un tiempo $O(1)$. Con lo que podemos decir que el algoritmo de empuje de preflujo corre en un tiempo de $O(n^2 \cdot m)$. Estableciendo el siguiente teorema.

Teorema 5.1 *El algoritmo genérico de empuje de preflujo se ejecuta en un tiempo de $O(n^2 \cdot m)$, en el peor de los casos.* $\diamond$

Varias modificaciones al algoritmo genérico de empuje de preflujo podrían mejorar su desempeño computacional.

Implementaciones específicas del algoritmo genérico de empuje de preflujo

El tiempo de ejecución del algoritmo genérico de preflujo es comparable con el del algoritmo de rutas aumentantes más cortas. Sin embargo, el algoritmo de empuje de preflujo es más flexible. Especificando diferentes reglas para seleccionar nodos activos para la operación de empuje/re-etiquetamiento, podemos generar varios algoritmos diferentes con una complejidad diferente al del algoritmo genérico. El cuello de botella en el algoritmo genérico es el número de envíos que no saturan y usando algunas reglas específicas para examinar nodos activos podríamos reducir el número de envíos que no saturan. Consideremos las siguientes tres implementaciones:

1. *Algoritmo de empuje de preflujo PEPS.* Este algoritmo examina los nodos activos en orden PEPS, (primero en entrar, primero en salir). Este algoritmo se ejecuta en un tiempo de $O(n^3)$.

2. *Algoritmo de empuje de preflujo de etiquetas máximas.* Este algoritmo siempre envía flujo de un nodo activo a un nodo que cuenta con la máxima etiqueta de distancia. Este algoritmo se ejecuta en un tiempo de $O(n^2 \cdot m^{1/2})$.

3. *Algoritmo de excesos escalables.* Este algoritmo envía flujo de un nodo con exceso suficientemente grande a un nodo con exceso suficientemente pequeño. Este algoritmo se ejecuta en un tiempo de $O(n \cdot m + n^2 \cdot \log U)$.

Estas implementaciones son descritas y justificadas en las siguientes secciones de este capítulo.

5.1. Algoritmo de preflujo PEPS

Antes de describir la implementación PEPS definamos el concepto de *revisión de nodo*. En una iteración, el algoritmo genérico de preflujo selecciona un nodo, digamos i, y ejecuta un envío de flujo que puede ser que sature o que no sature, o re-etiqueta el nodo. Si el algoritmo ejecuta un envío que satura, el nodo i podría mantenerse activo, pero esto no obliga al algoritmo a seleccionar este nodo de nuevo para la siguiente iteración. El algoritmo podría seleccionar otro nodo para la siguiente operación de envío/re-etiquetamiento. Sin embargo, es fácil incorporar la regla de que cuando el algoritmo seleccione un nodo activo lo mantenga para enviar flujo hasta que el exceso del nodo sea cero o el algoritmo re-etiquete el nodo. En consecuencia el algoritmo podría ejecutar varios envíos que saturan seguidos por un envío que no sature o una operación de re-etiquetamiento. Nos referiremos a esta secuencia de operaciones como una examinación de nodo. A partir de este momento asumiremos que cada algoritmo de preflujo adopta esta regla para seleccionar nodos en la operación de envío/re-etiquetamiento.

El algoritmo de preflujo PEPS examina los nodos activos en orden PEPS (primero en entrar, primero en salir). Mantiene un conjunto LISTA como una cola, selecciona un nodo i del inicio de LISTA, ejecuta envíos desde este nodo y agrega nodos activos al final de LISTA. Examina un nodo i hasta que deja de ser activo o es re-etiquetado, en el último caso, se agrega este nodo al final de LISTA. El algoritmo termina cuando la lista de nodos activos LISTA está vacía.

A manera de ilustrar esta especialización del algoritmo genérico de empuje - preflujo, veamos el ejemplo de la Figura 48(a). La operación de preproceso genera un exceso en los arcos adyacentes al nodo origen (nodo 1).

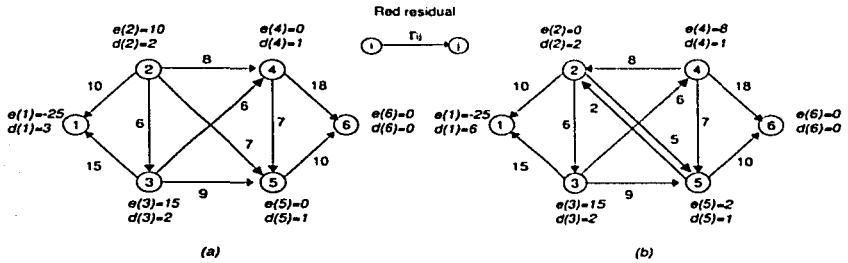

Figura 48: Ejemplo del algoritmo PEPS de empuje - preflujo.

Supongamos que la lista que contiene los nodos activo es $LISTA = \{2, 3\}$ en este momento. El algoritmo saca el nodo 2 de la lista y lo examina. Supongamos que el algoritmo hace un envío que satura de 8 unidades en el arco $(2, 4)$ y uno que no satura en el arco $(2, 5)$ de 2 unidades. En la Figura 48(b) podemos ver la red después de estas operaciones. Como resultado de estos envíos, los nodos 4 y 5 se convierten en activos y los agregamos a la lista en ese orden, resultando $LISTA = \{3, 4, 5\}$. Podemos ver en la tabla de la Figura 49 los siguientes pasos del algoritmo.

i	Arco admisible	Proceso empuje/re-etiqueta	$LISTA$
3	$(3,4)$	envía $\delta = 6$ unidades al nodo 4, $e(3) = 9$, $e(4) = 6$	4,5
3	$(3,5)$	envía $\delta = 9$ unidades al nodo 5, $e(3) = 0$, $e(5) = 11$	4,5
4	$(4,6)$	envía $\delta = 14$ unidades al nodo 6, $e(4) = 0$, $e(6) = 14$	5
5	———	d(5)=3	5
5	$(5,6)$	envía $\delta = 10$ unidades al nodo 6, $e(5) = 1$, $e(6) = 24$	5
5	$(5,2)$	envía $\delta = 1$ unidades al nodo 2, $e(5) = 0$, $e(2) = 1$	2
2	———	d(2)=3	2
2	$(2,3)$	envía $\delta = 1$ unidades al nodo 3, $e(2) = 0$, $e(3) = 1$	3
3	———	d(3)=4	3
3	$(3,2)$	envía $\delta = 1$ unidades al nodo 2, $e(3) = 0$, $e(2) = 1$	2
2	———	d(2)=4	2
2	$(2,5)$	envía $\delta = 1$ unidades al nodo 5, $e(2) = 0$, $e(5) = 1$	5
5	———	d(5)=5	5
5	$(5,2)$	envía $\delta = 1$ unidades al nodo 2, $e(5) = 0$, $e(2) = 1$	2
2	———	d(2)=5	2
2	$(2,3)$	envía $\delta = 1$ unidades al nodo 3, $e(2) = 0$, $e(3) = 1$	3
3	———	d(3)=6	3
3	$(3,2)$	envía $\delta = 1$ unidades al nodo 2, $e(3) = 0$, $e(2) = 1$	2
2	———	d(2)=6	2
2	$(2,5)$	envía $\delta = 1$ unidades al nodo 5, $e(2) = 0$, $e(5) = 1$	5
5	———	d(5)=7	5
5	$(5,2)$	envía $\delta = 1$ unidades al nodo 2, $e(5) = 0$, $e(2) = 1$	2
2	———	d(2)=7	2
2	$(2,1)$	envía $\delta = 1$ unidades al nodo 1, $e(2) = 0$, $e(1) = -24$	

Figura 49: Pasos del algoritmo PEPS de empuje - preflujo

Por lo que el valor de flujo máximo del ejemplo es $v = 24$ y la red resultante la podemos ver en la Figura 50.

Para analizar la complejidad del peor de los casos del algoritmo de preflujo PEPS, particionaremos el número de examinación de nodos en diferentes fases. La primera

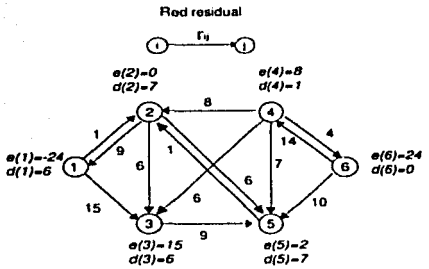

Figura 50: Algoritmo PEPS de empuje - preflujo, resultado.

fase consiste en la examinación de los nodos que se convierten en acitvos durante la operación de preproceso. La segunda fase consiste en la examinación de los nodos que están en la cola después de que el algoritmo ha examinado todos los nodos de las primer fase. Similarmente, la tercera fase consiste en la examinación de todos los nodos que están en la lista después de que el algoritmo ha examinado los nodos de la segunda fase, y así sucesivamente. En el ejemplo anterior, la primera fase consistió en la examinación del conjunto de nodos {2,3}, y la segunda fase en la examinación del conjunto {4,5}. Observemos que cualquier nodo es examinado por el algoritmo a lo más una vez en cada fase.

Mostraremos que el algoritmo ejecuta a lo más $(2n^2 + n)$ fases. Cada fase examina cualquier nodo a lo más una vez y cada examinación de nodo ejecuta a lo más un envío que no satura. Por lo que, la cota de $(2n^2+n)$ en el número total de fases implica una cota de $O(n^3)$ de envíos que no saturan. Este resultado también podría implicar que el algoritmo de preflujo PEPS se ejecuta en un tiempo de $O(n^3)$ porque el cuello de botella de las operaciones en el algoritmo genérico de preflujo es precisamente el número de envíos que no saturan.

Para acotar el número de fases en el algoritmo, consideraremos el cambio total en la función potencia $\Phi = \max\{d(i) : i \text{ es activo}\}$ en una fase completa. Por "el cambio total" nos referimos a la diferencia entre el valor inicial y el valor final de la función potencia durante una fase. Consideremos los siguientes dos casos:

1. El algoritmo ejecuta al menos una operación de re-etiquetamiento durante una fase. Entonces Φ podría incrementar tanto como el incremento máximo de cualquier etiqueta de distancia y entonces por el Lema 5.3 el incremento total de Φ es a lo más $2n^2$.

2. El algoritmo no ejecuta ninguna operación de re-etiquetamiento durante una fase. En este caso el exceso en cada nodo activo al inicio de la fase se mueve hacia los nodos con menor etiqueta de distancia, por lo que Φ decrece al menos una unidad.

Combinando los casos 1 y 2, encontramos que el número total de fases es a lo más $(2n^2 + n)$; el segundo término corresponde al valor inicial de Φ. Con lo que tenemos el siguiente teorema.

Teorema 5.2 *El algoritmo de preflujo PEPS se ejecuta en un tiempo de* $O(n^3)$*, en el peor de los casos.*

o

5.2. Algoritmo de preflujo de etiquetas más altas

El algoritmo de preflujo de etiquetas más altas siempre envía flujo desde el nodo activo con la etiqueta de distancia más alta. Es fácil ver que una cota para los envíos que no saturan es $O(n^3)$. Sea $h^* = \max\{d(i) : i \text{ es activo }\}$. El algoritmo primero examina los nodos con etiqueta de distancia igual a h^* y envía flujo a nodos con etiqueta de distancia igual a $(h^* - 1)$, y estos nodos, cuando les llegue su turno, enviarán flujo a nodos con etiquetas de distancia igual a $(h^* - 2)$, y así sucesivamente hasta que el algoritmo re-etiqueta un nodo o se agotan todos los nodos activos. Cuando se ha re-etiquetado un nodo, el algoritmo repite el mismo proceso. Notemos que si el algoritmo no re-etiqueta ningún nodo en n examinaciones consecutivas, todos los excesos llegan al nodo destino (o al nodo origen) y el algoritmo termina. Como el algoritmo ejecuta a lo más $2n^2$ operaciones de re-etiquetamiento, por el Lema 5.3, inmediatamente obtenemos una cota de $O(n^3)$ para el número de examinaciones. Ya que cada examinación de nodos supone a lo más un envío que no satura, el algoritmo de preflujo de etiquetas más altas ejecuta $O(n^3)$ envíos que no saturan.
En la discusión anterior no hemos mencionado un detalle importante: ¿Cómo seleccionamos un nodo con la etiqueta de distancia más alta sin gastar mucho tiempo? Utilicemos la siguiente estructura de datos. Para cada $k = 1, 2, \ldots, 2n - 1$ mantendremos una lista ligada.

$$LISTA[k] = \{i : i \text{ es activo y } d(i) = k\}.$$

Definimos una variable *nivel* que es una cota superior del más alto de los valores de k para los cuales $LISTA[k]$ es no vacía. Para determinar el nodo con la etiqueta de distancia más alta, examinamos las listas $LISTA[nivel]$, $LISTA[nivel - 1]$, ..., hasta que encontramos una lista no vacía, digamos $LISTA[p]$. Hacemos nivel igual p y seleccionamos cualquier nodo de $LISTA[p]$. Además, si la etiqueta de distancia de un nodo se incrementa mientras el algoritmo lo está examinando, hacemos nivel igual a la nueva etiqueta de distancia del nodo. Obsérvese que el total de incremento en nivel es a lo más $2n^2$, por el Lema 5.3, por lo que el decremento total es a lo más de $2n^2 + n$. En consecuencia, revisar las listas $LISTA[nivel]$, $LISTA[nivel - 1]$, ..., para encontrar la primer lista no vacía no es una operación pesada.

El algoritmo de preflujo de etiquetas más altas es muy eficiente en la práctica ya que ejecuta el menor número de envíos que no saturan. Para ilustrar intuitivamente porque el algoritmo es eficiente en la práctica, consideremos el problema de flujo máximo que se muestra en la Figura 51(a). La operación de preproceso genera un exceso de 1 unidad en cada uno de los nodos 2, 3, ..., $n - 1$, en la Figura 51(b) se muestra este hecho. El algoritmo de etiquetas más altas examina el nodo 2, 3, ..., $n - 1$, en este orden y envía el todo el exceso al nodo destino. En cambio, el algoritmo de preflujo PEPS podría ejecutar muchos más envíos. Supongamos que al final de la operación de preproceso la lista de nodos activos es $LISTA = \{n - 1, n - 2, \ldots, 3, 2\}$. Entonces

el algoritmo podría examinar cada uno de esos nodos en la primera fase obteniendo la solución de la Figura 51(c). En este punto, $LISTA = \{n-1, n-2, \ldots, 4, 3\}$. Con esto podemos decir que todo el algoritmo podría ejecutar $(n-2)$ fases y usar $(n-2)+(n-3)+\cdots+1$ envíos que no saturan.

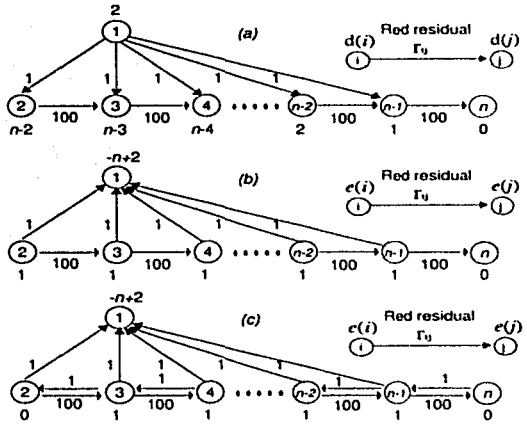

Figura 51: Mal ejemplo para aplicarle el algoritmo de preflujo PEPS.

Aunque el ejemplo anterior es un caso extremo, ilustra la ventaja de enviar flujo desde nodos activos con etiquetas de distancia más grandes. En este ejemplo el algoritmo PEPS selecciona un nodo con exceso y envía el exceso por todo el camino rumbo al nodo destino. Selecciona otro exceso y lo envía al nodo destino, y repite este proceso hasta que ningún nodo contenga exceso. Por el otro lado, el algoritmo de etiquetas más altas empieza en el nivel más alto y envía todos los excesos de este nivel al siguiente nivel inferior y repite este proceso. Como el algoritmo examina nodos con etiquetas de distancia cada vez menor, acumula los excesos y envía esta acumulación de flujos hacia el nodo destino. Por consecuencia, el algoritmo de preflujo de etiquetas de distancias más altas evita envíos repetitivos en arcos cargando una pequeña cantidad de flujo.

Esta característica del algoritmo de preflujo de etiquetas más altas también se traduce en una cota más pequeña en el número de envíos que no saturan. Mostraremos que la cota de $O(n^3)$ puede ser mejorada con un poco de análisis y que en realidad el

algoritmo ejecuta $O(n^2 \cdot m^{1/2})$ envíos que no saturan.

En cada etapa del algoritmo de preflujo, cada nodo distinto del nodo destino tiene a lo más un arco actual el cual, por definición, debe ser admisible. Denotamos esta colección de arcos actuales como el conjunto F. Este conjunto tiene a lo más $n-1$ arcos, tiene a lo más un arco saliente por nodo y no contiene ningún ciclo. Estas características implican que F define un *bosque*, al cual nos referiremos, de ahora en adelante después de cada actualización, como el *bosque actual*. La Figura 52 muestra un ejemplo de un *bosque actual*. Notemos que cada árbol en el bosque es un árbol con raíz, la raíz es el nodo del cual no salen arcos.

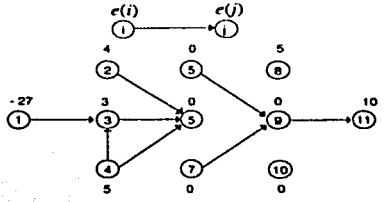

Figura 52: Ejemplo de un bosque actual.

Antes de continuar, introduzcamos notación adicional. Para cualquier nodo $i \in N$, definimos a $D(i)$ como el conjunto de descendientes de ese nodo en F. Notemos que las etiquetas de distancia de los descendientes de cualquier nodo i son mayores a $d(i)$. Nos referiremos a un nodo activo sin descendientes activos, diferentes a él mismo, como un *nodo activo maximal*. Definamos H como el conjunto de nodos activos maximales. Veamos que dos nodos activos maximales tienen distintos descendientes. También notemos que el algoritmo de preflujo de etiquetas más altas siempre envía flujo desde nodos activos maximales.

Verifiquemos la cota de $O(n^2 \cdot m^{1/2})$ para el algoritmo de preflujo de etiquetas más altas utilizando un argumento de funciones potencia. El argumento se basa en un parámetro K, cuyo valor óptimo veremos después. Nuestra función potencia es $\Phi = \sum_{i \in H} \Phi(i)$, donde $\Phi(i) = max\{0, K + 1 - |D(i)|\}$. Observemos que para cualquier nodo i, $\Phi(i)$ es a lo más K (porque $|D(i)| \geq 1$). También veamos que Φ cambia siempre que cambia el conjunto de nodos activos maximales H o cuando $|D(i)|$ cambia para un nodo activo maximal i.

Estudiemos el efecto en la función potencia Φ ocasionado por varias operaciones del

algoritmo de preflujo. El algoritmo cambia el conjunto de arcos actuales, ejecutando envíos que saturan y que no saturan, y re-etiquetando nodos. Todas esas operaciones tienen efecto en el valor de Φ. Mediante la observación de cada uno de estos efectos sobre Φ, encontraremos una cota para el número de envíos que no saturan.

Primero consideremos un envío que no satura sobre el arco (i, j) que emana de un nodo activo maximal i. Notemos que un envío que no satura ocurre en un arco actual y no cambia el bosque actual; simplemente mueve el exceso del nodo i al nodo j. La Figura 53(a) ilustra un envío que no satura en el arco (3,4). Como resultado del envío el nodo i deja de ser activo y j podría convertirse en un nuevo nodo activo maximal. Ya que $|D(j)| > |D(i)|$, este envío disminuye $\Phi(i) + \Phi(j)$ por al menos 1 unidad si $|D(i)| \leq K$ si no $\Phi(i) + \Phi(j)$ no cambia.

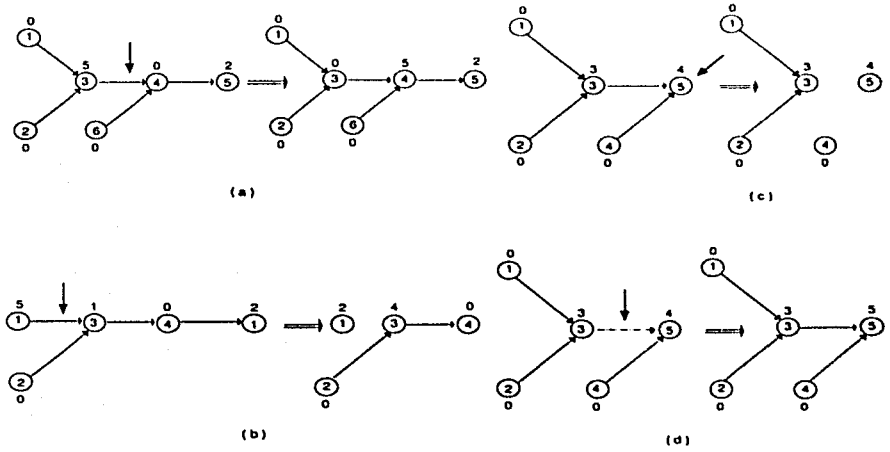

Figura 53: Operaciones a un bosque actual.

Ahora consideremos un envío que satura en un arco (i, j) que emana de un nodo activo maximal i. Como resultado del envío, el arco (i, j) deja de ser admisible y queda fuera del bosque actual. La Figura 53(b) muestra un envío que satura en el arco (1,3). El nodo i se mantiene como activo maximal y el nodo j podría convertirse en un nodo activo maximal. En consecuencia, esta operación podría incrementar el valor de Φ hasta por K unidades.

Veamos el re-etiquetamiento de un nodo activo maximal i. Re-etiquetamos un nodo cuando éste no tiene arcos admisibles; por lo que no hay arcos actuales que emanen de él. Como consecuencia, el nodo i debe ser una raíz en el bosque actual. Más aún, ya que el nodo i es un nodo activo maximal, ninguno de sus propios descendientes puede ser activo. Después de que el algoritmo ha re-etiquetado el nodo i, todos los arcos que entran al nodo i se dejan de ser admisibles; por ello todos los arcos actuales que entran al nodo i dejarán de pertenecer al bosque actual. La Figura 53(c) lo ilustra. Claramente, este cambio no crea ningún nodo activo maximal, sin embargo el número de descendientes del nodo i disminuye hasta uno. Con lo que $\Phi(i)$ puede incrementar a lo más K.

Finalmente, consideremos la incorporación de nuevos arcos actuales en el bosque actual la cual no crea ningún nuevo nodo activo maximal. Esto podría, de hecho, eliminar algunos nodos activos maximales e incrementar el número de descendientes de algunos nodos. La Figura 53(d) ilustra este caso. En ambos casos la función Φ no se incrementa. Para resumir lo anterior tenemos el siguiente resultado:

Propiedad 5.1

(a) *Un envío que no satura, desde un nodo activo maximal i, no incrementa Φ. Si $|D(i)| \leq K$ entonces Φ disminuye al menos una unidad.*

(b) *Un envío que satura, desde un nodo activo maximal i, puede incrementar Φ a lo más K unidades.*

(c) *El re-etiquetamiento de un nodo activo maximal i puede incrementar Φ a lo más K unidades.*

(d) *Agregar arcos actuales no incrementa Φ.*

Para analizar el peor de los casos, definimos el concepto de *fase* como una secuencia de envíos entre 2 operaciones consecutivas de re-etiquetamiento. El Lema 5.3 implica que el algoritmo contiene $O(n^2)$ fases. Nos referiremos a una fase *ligera* si ejecuta a lo más $2n/K$ envíos que no saturan y *pesada* en otro caso.

Claramente, el número de envíos que no saturan en fases ligeras es a lo más $O(n^2 \cdot 2n/K) = O(n^3/K)$. Para la cota en el caso de envíos pesados usaremos un argumento basado en funciones potencia Φ.

Por definición, una fase pesada ejecuta al menos $2n/K$ envíos que no saturan. Ya que la red puede contener a lo más n/K nodos con K descendientes o más, al menos

n/K envíos que no saturan deben ser de nodos con menos de K descendientes. El algoritmo de preflujo de etiquetas máximas siempre ejecuta la operación de envía/re-etiqueta sobre un nodo activo maximal, en consecuencia, la Propiedad 5.1(a) lo que implica que cada uno de esos envíos que no saturan producen un decremento en Φ de al menos 1. Entonces la Propiedad 5.1(b) y (c) implican que el incremento total en Φ debido a envíos que saturan y re-etiquetaminetos es a lo más $O(nmK)$. Por ello, el algoritmo puede ejecutar $O(nmK)$ envíos que no saturan en una fase pesada.

Para resumir, notemos que las fases ligeras ejecutan $O(n^3/K)$ envíos que no saturan y las fases pesadas ejecutan $O(nmK)$ envíos que no saturan. Obtenemos el valor óptimo de K igualando los términos: $n^3/K = nmK$ o $K = n/m^{1/2}$. Para este valor de K, el número de envíos que no saturan es $O(n^2m^{1/2})$. Con lo que podemos establecer el siguiente resultado:

Teorema 5.3 *El algoritmo de preflujo de etiquetas más altas realiza del $O(n^2m^{1/2})$ envíos que no saturan y se ejecuta en el mismo tiempo.* ∘

5.3. Algoritmo de excesos escalables

El algoritmo genérico de preflujo permite flujo que en cada paso intermedio se viole la ley de conservación de flujo. Mediante envíos desde nodos activos, el algoritmo intenta satisfacer dicha ley. La función $e_{max} = \max\{e(i) : i \text{ es un nodo activo}\}$ nos da una medida de la no factibilidad de un preflujo. Durante una ejecución del algoritmo genérico, podríamos no ver ningún patrón en particular en los valores de e_{max} excepto que su valor decrece eventualemente hasta llegar a ser 0. El *algoritmo de excesos escalables* desarrolla una técnica para que sistemáticamente el valor de e_{max} disminuya hasta ser 0.

En el algoritmo genérico de preflujo los envíos que no saturan que llevan un flujo pequeño son los que forman el cuello de botella del algoritmo en la teoría. El algoritmo de exceso escalable asegura que, en cada envío que no satura, se lleva una cantidad de flujo "sufucientemente grande" y entonces el número de envíos que no saturan sería "sufucientemente pequeño."

Sea el *exceso dominante* Δ una cota superior de e_{max}. Nos referiremos a un nodo i con *exceso grande* si cumple que $e(i) \geq \Delta/2 \geq e_{max}/2$ y como un nodo con *poco exceso* en otro caso. El algoritmo de excesos escalables siempre envía flujo desde nodos con un exceso grande, lo que asegura que en envíos que no saturan se lleve gran cantidad de flujo cerca del nodo destino.

Además, el algoritmo no permite que el máximo de los excesos crezca más alla de Δ. Esta estrategia algorítmica es útil por la siguiente razón; supongamos que varios nodos envían flujo a un solo nodo j, creando un gran exceso. Es probable que el nodo j no pueda envíar el flujo acumulado cerca del nodo destino, y entonces el algoritmo necesitaría incrementar su etiqueta de distancia y regresaría mucho de su exceso a los nodos de los cuales vino. Así que enviar demasiado flujo a cualquier nodo podría ser un esfuerzo desperdiciado.

Las dos condiciones que hemos discutido -que cada envío que no satura debe llevar por lo menos $\Delta/2$ unidades de flujo y que el exceso no debe ser mayor a Δ- implican que necesitamos elegir con cuidado los nodos para la operación de empuje/re-etiquetamiento. La siguiente regla de elección asegura que se logran estos objetivos.

Regla de selección de nodo. *De entre todos los nodos con exceso grande, selecciona un nodo con la etiqueta de distancia más pequeña, rompiendo los empates arbitrariamente.*

El algoritmo de exceso escalable usa la misma operación de envia/re-etiqueta(i) que usa el algoritmo genérico, pero con una pequeña diferencia. En lugar de enviar

$\delta = \min\{e(i), r_{ij}\}$ unidades de flujo, este envía $\delta = \min\{e(i), r_{ij}, \Delta - e(j)\}$ unidades. Este cambio asegura que el algoritmo no permite que el exceso sea mayor a Δ.

El algoritmo ejecuta fases de escalamiento disminuyendo el valor del exceso dominante Δ fase a fase. Nos referiremos a una fase específica de escalamiento con un valor particular Δ como una Δ-*fase de escalamiento*. Inicialmente, $\Delta = 2^{\lfloor \log U \rfloor}$. Como el logaritmo es base 2, $U \leq \Delta \leq 2U$. Durante una Δ-*fase de escalamiento*, $\Delta/2 < e_{max} \leq \Delta$; el valor de e_{max} podría incrementarse o disminuir durante la fase. Cuando $e_{max} \leq \Delta/2$, se inicia una nueva fase. Después de que el algoritmo ha ejecutado $\lceil \log U \rceil + 1$ fases de escalamiento, e_{max} disminuye a 0, obteniendo el flujo máximo. El Algoritmo 5.3 muestra el pseudocódigo del algoritmo de preflujo de excesos escalables.

Algoritmo 5.3 Algoritmo de Excesos Escalables

```
begin
  preproceso;
  Δ ← 2^⌊log U⌋;
  while Δ ≥ 1 do
    while la red contiene un nodo i con exceso grande do
      de entre todos los nodos con exceso grande, selecciona un nodo
      con la etiqueta de distancia más pequeña;
      ejecuta envia/re-etiqueta(i) mientras se asegure que el exceso
      del nodo no supera a Δ;
    end while;
    Δ ← Δ/2;
  end while;
end;
```

Lema 5.6 *El algoritmo satisface las siguientes condiciones:*

(a) *Cada envío que no satura lleva al menos $\Delta/2$ unidades de flujo.*

(b) *Ningún exceso es mayor que Δ*

Demostración. Consideremos un envío que no satura en el arco (i, j). Como (i, j) es admisible $d(j) < d(i)$. Además, i es un nodo con la menor de las etiquetas de distancia de entre todos los nodos con exceso grande por lo que $e(i) \geq \Delta/2$ y $e(j) < \Delta/2$. Ya que este envío no satura, las unidades de flujo a enviar son $\min\{e(i), \Delta - e(j)\} \geq \Delta/2$, dando como resultado la primera parte del lema. Esta operación de envío solamente incrementa el exceso del nodo j. El nuevo exceso de j es

$$e(j) + \min\{e(i), \Delta - e(j)\} \leq e(j) + \{\Delta - e(j)\} \leq \Delta$$

Por lo que los excesos de todos los nodos permanecen menores o iguales a Δ. $\diamond$

Lema 5.7 *El algoritmo de excesos escalables ejecuta $O(n^2)$ envíos que no saturan por fase de escalamiento y $O(n^2 \log U)$ envíos en total.*

Demostración. Consideremos la función potencia $\Phi = \sum_{i \in N} e(i) \cdot d(i)/\Delta$. Utilizando esta función, estableceremos la primera afirmación del lema. Como el algoritmo ejecuta $O(\log U)$ fases de escalamiento, la segunda afirmación es consecuencia de la primera.
El valor inicial de Φ al inicio del la $\Delta - fase$ de escalamiento está acotado por $2n^2$ porque $e(i)$ está acotado por Δ y $d(i)$ está acotado por $2n$. Durante una operación de envía/re-etiqueta(i) uno de los siguientes dos casos sucede:

Caso 1. El algoritmo no es capaz de encontrar un arco admisible por el cual pueda ser enviado flujo. En este caso la etiqueta de distancia del nodo i incrementa en $\epsilon \geq 1$ unidades. Esto incrementa el valor de Φ a lo más en ϵ unidades porque $e(i) \leq \Delta$. Ya que para cada i el incremento total en $d(i)$ durante la ejecución del algoritmo está acotado por $2n$, por el Lema 5.3, el incremento total en Φ debido al re-etiquetamiento de nodos esta acotado por $2n^2$ en la Δ-fase de escalamiento.

Caso 2. El algoritmo es capaz de identificar un arco por el cual puede ser enviado flujo, esto provoca un envío ya sea que satura o que no satura. En ambos casos, Φ disminuye. Un envío que no satura en un arco (i, j) envía al menos $\Delta/2$ unidades de flujo del nodo i al nodo j y ya que $d(j) = d(i) - 1$, depués de esta operación Φ disminuye al menos en $1/2$ unidad. Como el valor inicial de Φ al inicio de la Δ-fase de escalamiento es a lo más $2n^2$ y el incremento en Φ durante esta fase de escalamiento suma a lo más $2n^2$ (del Caso 1), el número de envíos que no saturan están acotados por $8n^2$. $\diamond$

Este lema implica una cota de $O(nm+n^2 \log U)$ para el algoritmo de excesos escalables ya que como hemos visto anteriormente las otras operaciones -tales como envíos que saturan, operaciones de re-etiquetamiento y encontrar arcos admisibles- requieren un tiempo de $O(nm)$.
Con esta observación podemos resumir todo lo anterior en el siguiente resultado:

Teorema 5.4 *El algoritmo de excesos escalables se ejecuta en un tiempo de $O(nm + n^2 \log U)$, en el peor de los casos.* $\diamond$

6. Simulación

En este Capítulo presentaremos la simulación de los algoritmos, descritos durante este trabajo.
Esta herramienta consiste en la representación gráfica de la ejecución de algunos de los algoritmos que mostramos a lo largo de este trabajo. Esta aplicación no intenta ser una herramienta para solucionar problemas de Flujo Máximo en redes sino más bien una ayuda para la presentación de este tema.

6.1. Descripción de la aplicación.

A continuación describimos la herramienta desarrollada en Excel para ilustrar los algoritmos.

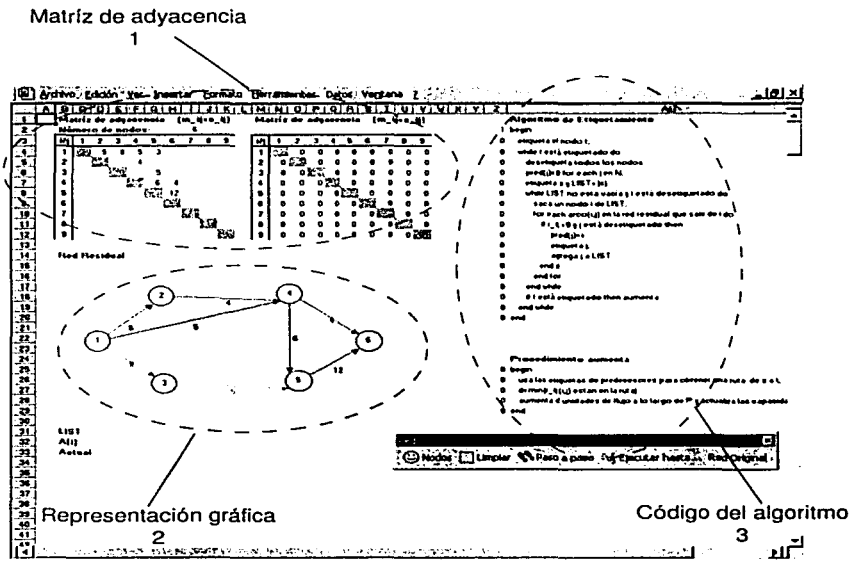

Figura 54: Pantalla de la aplicación de Excel

La pantalla de la aplicación consta de las siguientes partes. Figura 54.

Región 1 En esta región tenemos dos matrices, la de la izquierda representa la matriz de adyacencia de la gráfica en cuestión y cada entrada i, j representa la capacidad de cada arco. La matriz de la derecha también es una matriz de adyacencia, sólo que la entrada i, j de esta matriz representa el flujo que se va generando en cada arco a lo largo de la ejecución del algoritmo.

Región 2 En esta región tenemos la representación gráfica de los datos de las dos matrices de la Región 1, una a la vez. Se alternan con un botón de la barra de herramientas que veremos un poco más adelante.

Región 3 En esta región tenemos el código del algoritmo que tratamos de simular, mostramos la rutina principal y, en su caso, las subrutinas que utiliza.

Como habíamos dicho, la aplicación cuenta con una barra de herramientas. La cual se muestra en la Figura 55.

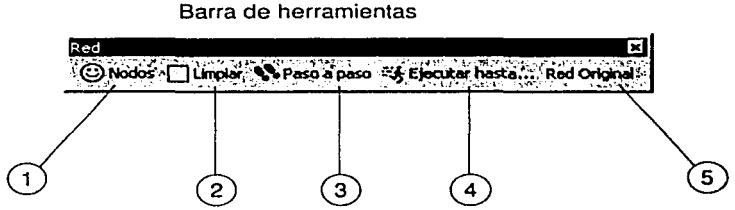

Figura 55: Barra de herramientas de la aplicación de Excel

La barra cuenta con 5 botones:

Botón 1 Con este botón generamos los nodos de la gráfica, según el número de nodos que hayamos escrito en la celda, arriba de la primera matriz.

Botón 2 Con este botón generamos los arcos de la gráfica, según los datos que hayamos escrito en la matriz de capacidades, matriz izquierda. Además de que limpia los datos para iniciar una ejecución del ritmo.

Botón 3 Con este botón podemos ejecutar paso a paso cada línea del algoritmo mostrado en la Región 3, cada click ejecuta un paso del algoritmo.

Botón 4 Con este botón podemos ejecutar el algoritmo hasta una línea determinada, simplemente nos situamos en la línea donde queremos que se detenga y damos click al botón, el código se ejecutará y se detendrá en esa línea.

Botón 5 Con este botón podemos alternar de gráfica entre la red residual y la red original. La gráfica se calcula con base a las matrices en ese momento.

Los algoritmos implementados en esta herramienta fueron:

- Algoritmo de etiquetamiento
- Algoritmo de capacidades escalables
- Algoritmo de rutas aumentantes más cortas
- Algoritmo de preflujo PEPS
- Algoritmo de preflujo de etiquetas más altas
- Algoritmo de preflujo de excesos escalables

En las siguientes secciones se ilustran pantallas de la simulación de algunos de los distintos algoritmos implementados.

6.2. Simulación para el Algoritmo de Etiquetamiento.

Veamos la simulación para el ejemplo del Algoritmo de Etiquetamiento de la Figura 26 de la página 37.

La simulación funciona de la siguiente manera, en cada entrada ij de la matriz de la izquierda de la Región 1 debemos escribir las capacidades de cada uno de los arcos (i, j) de la red. Mientras que la matriz de la derecha se encuentra en ceros ya que el flujo con el que iniciamos es el flujo $x = 0$, esta última matriz se irá actualizando automáticamente.

Una vez que tenemos las capacidades de los arcos almacenadas en la matriz, escribimos en la celda "12" el número de nodos que va a tener la red y damos un click sobre el primer botón de la barra de herramientas para generar los nodos, como se muestra en la Figura 56.

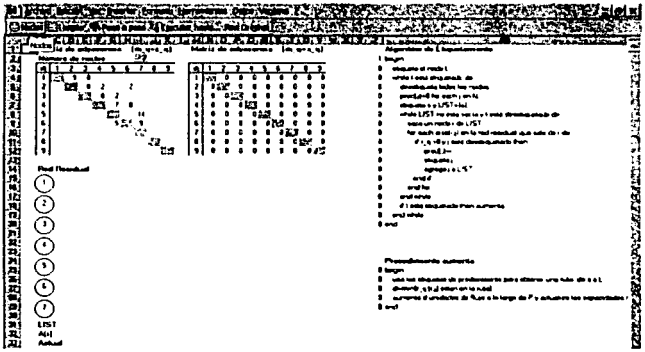

Figura 56: Simulación para el Algoritmo de Etiquetamiento

Los nodos generados los acomodamos a lo largo de la pantalla, según el diseño de la red. Para generar los arcos entre cada nodo damos click al segundo botón de la barra de herramientas. En la Figura 57 observamos la red residual con sus arcos y capacidades residuales.

Ya que tenemos construida la red residual, para empezar a ejecutar el algoritmo tenemos el tercer y cuarto botón de la barra de herramientas. Un click al tercer botón ejecuta una línea de código del algoritmo y un click al cuarto ejecuta el código hasta

una línea determinada. La Figura 57 nos muestra el resultado de la ejecución de la primera línea del código del Algoritmo de Etiquetamiento.

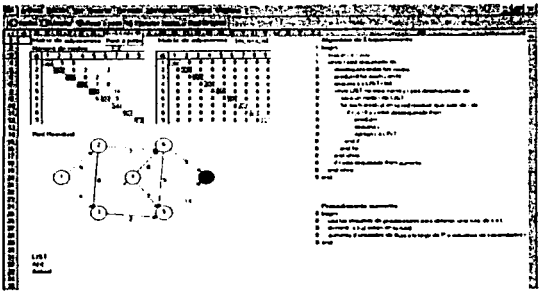

Figura 57: Primer paso del Algoritmo de Etiquetamiento

En la Figura 58 tenemos la simulación después del primer aumento que hace el algoritmo. Como podemos observar, la matriz del lado izquierdo indica los arcos a través de los cuales pasa el flujo encontrado.

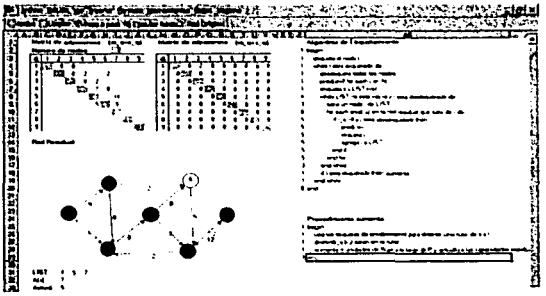

Figura 58: Primer aumento del Algoritmo de Etiquetamiento

En cualquier momento de la ejecución del algoritmo podemos cambiar de la red residual a la red original mostrando los arcos por los cuales se ha mandado flujo hasta ese momento. Esto se obtiene dando click al último botón de la barra.

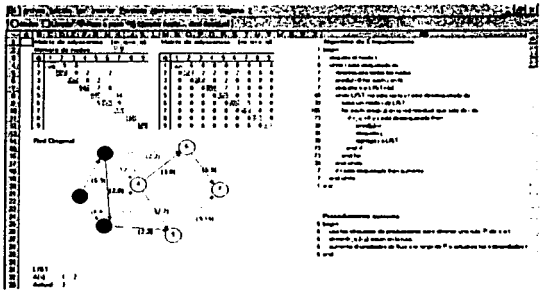

Figura 59: Resultado del Algoritmo de Etiquetamiento

Por último, en la Figura 59 podemos observar el flujo máximo encontrado para este ejemplo. En la red que observamos tenemos los arcos por los cuales pasa el flujo máximo que encontró el algoritmo. Además podemos observar que el algoritmo etiquetó algunos nodos (nodos coloreados). Estos nodos definen la partición de la cortadura mínima que el algoritmo pudo encontrar.

6.3. Simulación para el Algoritmo de Capacidades escalables.

Presentaremos la simulación para el algoritmo de Capacidades escalables tomando el ejemplo de la Figura 34 de la página 44.

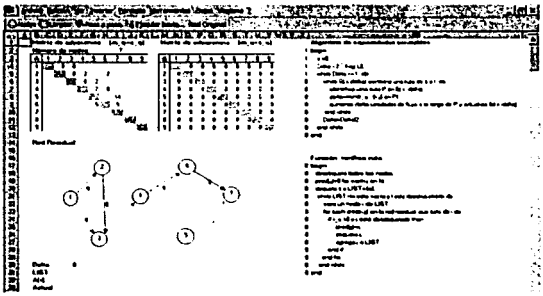

Figura 60: Simulación para el Algoritmo de Capacidades escalables

Una vez que hemos ingresado las capacidades de los arcos en la matriz de la izquierda de la Región 1 y hemos puesto el número de nodos de la red en la celda "12". Damos click al boton "Nodos", de la barra, para generar los nodos los cuales debemos acomodar a lo largo de la pantalla. Igual que en la simulación anterior, para generar los arcos damos click en el botón "Limpia" de la barra. Para ver la ejecución del algoritmo damos click al botón "Paso a paso", en la Figura 60 podemos observar la primer $\Delta-$red residual para $\Delta = 8$.

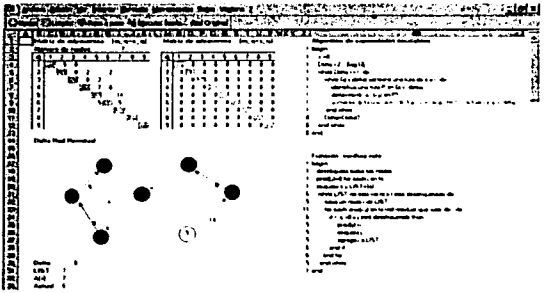

Figura 61: Aumento para el Algoritmo de Capacidades escalables

Como podemos ver en el código, este algoritmo utiliza en su ejecución una red llamada Δ-red residual y en ella busca rutas que van desde el nodo origen al nodo destino. En la Figura 61 mostramos la simulación cuando el algoritmo ha encontrado la primera ruta aumentante sobre la $\Delta-$red residual con $\Delta = 8$.

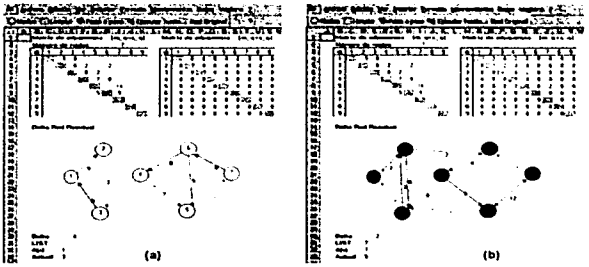

Figura 62: $\Delta-$red residual

Evidentemente, el algoritmo ya no encuentra rutas aumentantes en esta $\Delta-$red residual. Por lo que recalcula la $\Delta-$red residual con $\Delta = 4$, Figura 62(a). Pero encuentra que tampoco hay rutas aumentantes, lo que lleva al algoritmo a calcular la nueva $\Delta-$red residual con $\Delta = 2$ y encuentra la ruta aumentante $P : 1, 2, 3, 5, 7$ que se muestra en la Figura 62(b). Lo que origina un aumento en dos unidades de flujo.

Esta implementación del algoritmo utiliza la misma rutina de búsqueda de rutas que el algoritmo de etiquetamiento.

La última pantalla de la para este ejemplo se muestra en la Figura 63, a la derecha la red residual y a la izquierda la red original con el flujo máximo.

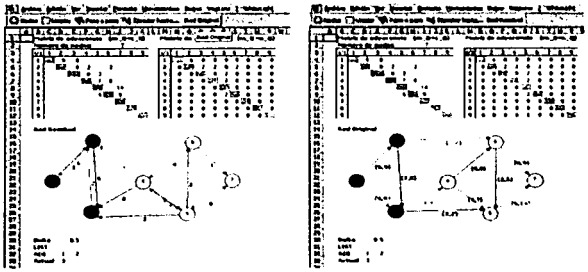

Figura 63: Red residual y red original. Resultado.

6.4. Simulación para el Algoritmo de Rutas aumentantes más cortas.

Presentaremos la simulación para el algoritmo de Rutas aumentantes más cortas tomando el ejemplo de la Figura 40 de la página 56.

Una vez que hemos generado la red con sus arcos y sus capacidades residuales, podemos ejecutar el algoritmo con la simulación. En la Figura 64 podemos ver la ejecución del algoritmo después de que ha encontrado las etiquetas de distancia exactas. En la columna "X" de la ventana podemos observar dichas etiquetas. Como podemos ver en el código, el algoritmo inicia con el nodo 1, nodo origen, y va buscando los arcos que son admisibles buscando alcanzar el nodo destino. Una vez que logra alcanzarlo

significa que ha encontrado una ruta aumentante más corta, la cual es construida con las etiquetas de predecesores que genera en cada paso. En nuestro ejemplo, la primer ruta aumentante se puede observar en la Figura 65, el aumento es de $\delta = 8$ y la red residual se encuentra encuentra actualizada.

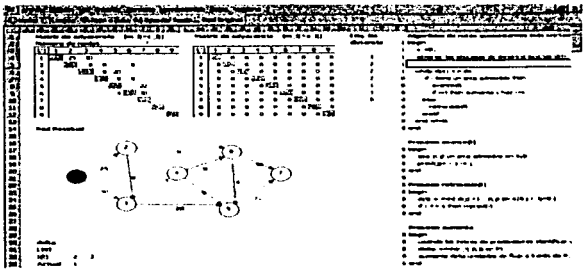

Figura 64: Simulación para Algoritmo de Rutas aumentantes más cortas

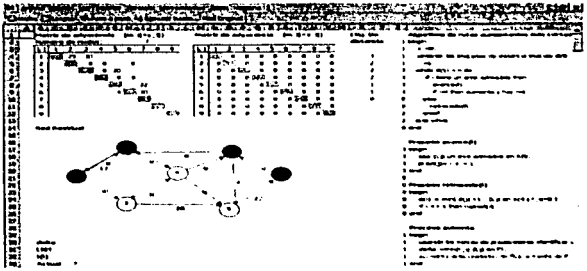

Figura 65: Primera ruta aumentante más corta

Después de que el algoritmo ha ejecutado un aumento, vuelve a iniciar con el nodo origen para emprender la búsqueda de arcos admisibles. En su camino, puede ser que en el nodo actual no haya arcos admisibles que incidan en él, en este caso el algoritmo hace dos cosas: primero modifica la etiqueta de dicho nodo, de tal forma que se genere un arco admisible para etapas posteriores y segundo regresa al último nodo que tenía un arco admisible. En la Figura 66 podemos ver como la etiqueta del nodo 2 es incrementada por el algoritmo al no encontrar arcos admisibles incidentes en él.

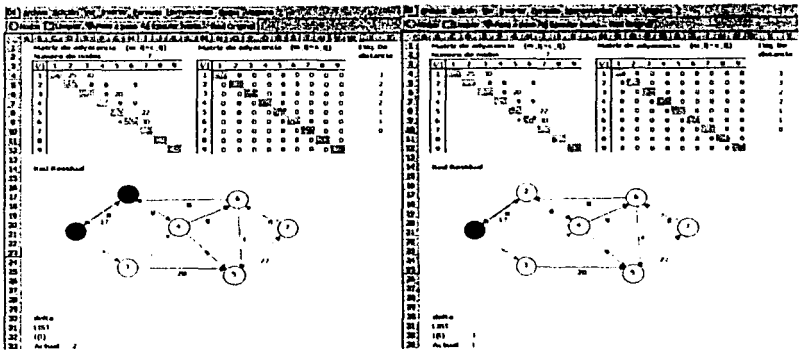

Figura 66: Re-etiquetamiento y retroceso.

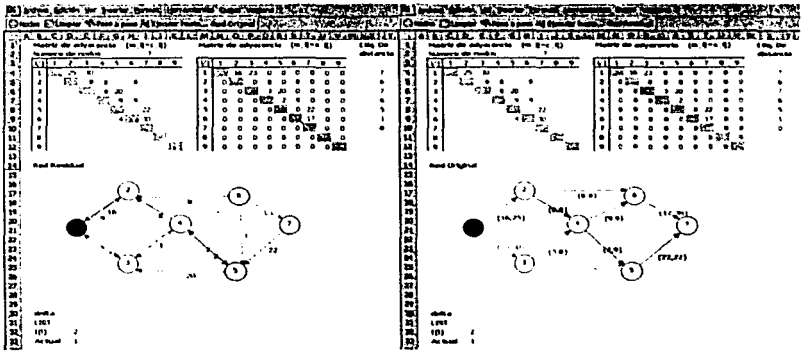

Figura 67: Resultado del algoritmo de Rutas aumentantes más cortas

El algoritmo termina cuando la etiqueta de distancia del nodo origen es mayor o igual al número de nodos de la red. En nuestro caso, son 7 nodos. En la Figura 67 tenemos la última red residual y la red original con el flujo máximo encontrado.

6.5. Simulación para el Algoritmo de preflujo PEPS.

Presentaremos la simulación para el algoritmo de Preflujo PEPS tomando el ejemplo de la Figura 48 de la página 69.

Este algoritmo, al igual que todos los de preflujo, tiene un procedimiento llamado preproceso, en el cual se calculan las etiquetas de distancia exactas, se saturan los arcos que salen del nodo origen y se re-etiqueta el nodo origen con el número de nodos. En la Figura 68 podemos ver la red residual después de este preproceso.

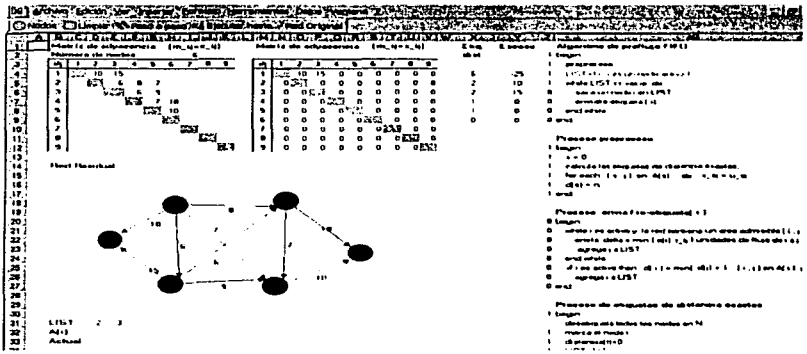

Figura 68: Simulación para el Algoritmo de preflujo PEPS

Notemos que al saturar los arcos que salen del nodo origen, generamos nodos con exceso o nodos activos, el algoritmo se ejecuta mientras haya exceso en los nodos. La existencia de nodos activos significa que el flujo actual no es factible ya que no se estaría cumpliendo con la ley de conservación de flujo.

A grandes rasgos, el algoritmo trabaja de la siguiente forma: selecciona nodos activos y distribuye el flujo enviándolo a través de arcos admisibles tratando de eliminar el exceso del nodo, en caso de no tener arcos admisibles, el nodo es re-etiquetado de tal forma que se genera al menos un arco admisible. De esta forma el flujo es distribuido ya sea hacia el nodo destino o el nodo origen. El algoritmo termina cuando ya no existen nodos activos.

La forma en que el algoritmo selecciona los nodos activos es la que le da su nombre. Cada que se genera un nuevo nodo activo, es agregado a una lista por orden de

aparición lo que genera una cola de nodos activos y se van seleccionando en orden PEPS (primero en entrar, primero en salir).

En la Figura 69 podemos ver que el nodo 2, que era activo después del preproceso y tenía un exceso de 10, ha distribuido el flujo hacia los nodos 4 y 5, por 8 y 2 unidades respectivamente. Los nodos 4 y 5 se convierten en activos y son agregados en la lista para ser seleccionados posteriormente.

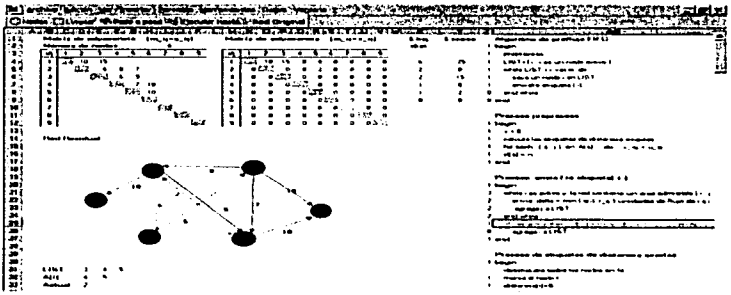

Figura 69: Algoritmo de preflujo PEPS.

La última red residual y el flujo máximo encontrado lo podemos ver en la Figura 70.

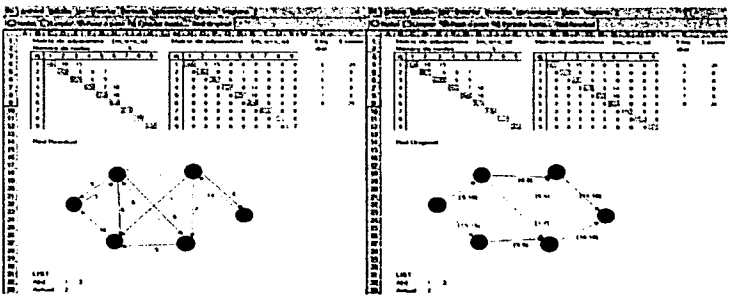

Figura 70: Resultado del Algoritmo de preflujo PEPS.

7. Conclusiones

Haciendo un recuento de los temas más importantes en este trabajo, recordemos el Capítulo 4 en el cual vimos el algoritmo genérico de *rutas aumentantes*, mencionando una de sus especializaciones más fáciles de implementar, el algoritmo de *etiquetamiento*. Este algoritmo mantiene, a lo largo de su ejecución un flujo factible, su mecánica es identificar rutas dirigidas del nodo origen al nodo destino en la red residual y aumentar el máximo flujo posible a través de ellas. Sin embargo, aún cuando su idea es muy simple, nos dimos cuenta que su tiempo de ejecución en el peor de lo casos es pseudopolinomial, ya que no se tiene un control sobre la cantidad de flujo enviado en cada ruta y podrían ejecutarse envíos de cantidades muy pequeñas, lo que lo hace poco eficiente en la práctica.

Motivados por esta desventaja en la eficiencia del algoritmo de etiquetamiento, revisamos una especialización de este algoritmo llamada algoritmo de *capacidades escalables*. La propuesta de este algoritmo es, al igual que el algoritmo de etiquetamiento, aumentar flujo a través de rutas dirigidas que van del nodo origen al nodo destino, con la diferencia de que elige las que tengan una capacidad residual suficientemente grande, lo que nos asegura que los envíos que se hacen através de estas rutas son relativamente grandes. Mostramos que su tiempo de ejecución en el peor de los casos es mejor que el del algoritmo de etiquetamiento pero aún así en la práctica este algoritmo tampoco era muy eficiente en la práctica. Lo anterior se debe principalmente a que su desempeño computacional está en función, entre otras cosas, del tamaño de las capacidades de sus arcos.

Por último, para este tipo de algoritmos, mostramos el algoritmo de *rutas aumentantes más cortas*, que es otra especialización del algoritmo de Etiquetamiento. Este algoritmo se basa, como su nombre lo indica, en localizar las rutas más cortas del nodo origen al nodo destino y por ellas aumentar el máximo flujo posible. Para lograrlo, utiliza las etiquetas de distancia para identificar las rutas má cortas, con lo que su implementación es relativamente fácil. Este algoritmo es eficiente en la práctica. Su tiempo de ejecución no está en función de las capacidades de los arcos sino simplemente en el tamaño de los datos.

En el Capítulo 5 presentamos el otro tipo de algoritmos utilizados para solucionar el problema de flujo máximo, el algoritmo *genérico de empuje - preflujo*. Este nuevo enfoque para resolver los problemas desecha la idea de aumentar flujo sólo a través de rutas que van desde el nodo origen al nodo destino, la propuesta de estos algoritmos es enviar flujo a través de arcos individuales, lo cual se puede ver como aumentar flujo a través de varias rutas a la vez. Esta técnica provoca que en etapas intermedias del

algoritmo se mantenga un pseudoflujo, ya que permite que existan nodos intermedios en donde el flujo que entra es mayor al flujo que sale de ellos, llamados nodos con exceso. Es sobre estos nodos que el algoritmo ejecuta sus operaciones. Este algoritmo genérico es flexible ya que no existe un orden específico para examinar nodos con excesos. El tiempo de ejecución en el peor de los casos es comparable al del algoritmo de Rutas aumentantes más cortas.

En este mismo capítulo, mostramos tres especializaciones de este algoritmo genérico de empuje - preflujo (o simplemente de preflujo). El algoritmo *PEPS de preflujo* y el algoritmo de *etiquetas más altas* son especializacionese en las cuales se especifica un orden por el cual se examinan los nodos con exceso. Este orden es importante ya que hace la diferencia entre los tiempos de ejecución en el peor de los casos. El algoritmo PEPS de preflujo, selecciona los nodos con exceso en orden PEPS, mientras que el de etiquetas de distancia mas altas lo hace en base a sus etiquetas. En ambos casos el tiempo de ejecución en el peor de los casos supera al del algoritmo genérico de preflujo. Ambos son muy eficientes en la práctica.

Como última especialización de este tipo de algoritmos presentamos el algoritmo de *excesos escalables*, el cual elige los nodos con gran cantidad de exceso y de entre los cuales elige los que tienen etiqueta de distancia más pequeña. Este algoritmo alcanza un buen tiempo en el peor de los casos, además es relativamente fácil de implementar.

Hicimos notar que los algoritmos de rutas aumentantes emplean mucho de su esfuerzo computacional en enviar flujo a través de rutas que van desde el nodo origen al nodo destino, ya que estos algoritmos podrían enviar flujo en repetidas ocasiones a lo largo de segmentos de ruta comúnes. Esta desventaja no se tiene con los algoritmos de empuje-preflujo. Otra diferencia entre estos dos tipos de algoritmos es que el algoritmo de rutas aumentantes busca la optimalidad, mientras que el algoritmo de empuje-preflujo busca la factibilidad.

En la tabla de la Figura 71 se muestran los tiempos de ejecución de los algoritmos presentados a lo largo de este documento.

Figura 71: Resumen de algoritmos de flujo máximo [1]

Tipo de Algoritmo	Algoritmos	Tiempo de ejecución
Rutas Aumentantes	De Etiquetamiento	$O(nmU)$
	De Capacidades escalables	$O(nm \log U)$
	De Rutas más cortas	$O(n^2m)$
Empuje - Preflujo	Genérico de preflujo	$O(n^2m)$
	De Preflujo PEPS	$O(n^3)$
	De Preflujo de etiquetas más altas	$O(n^2\sqrt{m})$
	De Excesos Escalables	$O(nm + n^2 \log U)$

Podemos concluir entonces que, los algoritmos de empuje-preflujo son más rápidos que los algoritmos de rutas aumentantes, siendo los mejores algoritmos, para rutas aumentantes: el algoritmo de rutas aumentantes más cortas y para los de empuje-preflujo: el algoritmo de etiquetas de distancia más altas.

Referencias

[1] **Ahuja, R.K., Magnanti, T.L. and Orlin, J.B.,**
Network flows: Theory, Algorithms, and Applications,
Prentice Hall, 1993.

[2] **Chartran, G. and Oellermann, R.,**
Applied and Algorithmic Graph Theory ,
Mc. Graw Hill, 1993.

[3] **Rockafellar, R. T.,**
Network flows and Monotropic Optimization,
John Wiley & Sons, 1984.

Apéndice A. Descomposición de flujo

En la formulación de problemas de flujo en redes podemos definir flujos en arcos o definir el flujo a través de rutas y ciclos. Por ejemplo en la Figura 72(a) se muestra un flujo en arcos que envía 7 unidades de flujo del nodo 1 al nodo 6. En la Figura 72(b) se muestra el mismo flujo pero ahora a través de rutas y ciclos.
En el flujo a través de rutas y ciclos enviamos 4 unidades a lo largo de la ruta 1-2-4-6, 3 unidades por la ruta 1-3-5-6, y 2 unidades por el ciclo 2-4-5-2. En este documento utilizamos la representación de flujo en arcos.

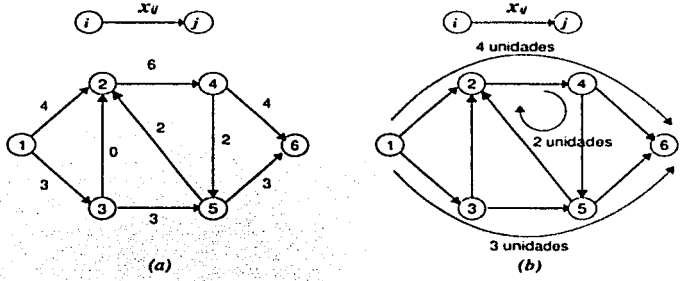

Figura 72: Red residual

Por un "flujo en un arco" nos referimos a un vector $x = \{x_{ij}\}$ que satisface las siguientes condiciones:

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = e(i) \text{ para toda } i \in N \quad (a)$$

$$0 \leq x_{ij} \leq u_{ij} \text{ para todo } (i,j) \in A. \quad (b)$$

donde $\sum_{i=0}^{n} e(i) = 0$. Nos referiremos a $e(i)$ como el *desequilibrio* del nodo, y representa el flujo que entra menos el flujo que sale del nodo i. Si el flujo que entra es mayor al flujo que sale, $e(i) > 0$ y decimos que el nodo es un nodo con *exceso*. Si el flujo que entra es menor al flujo que sale, $e(i) < 0$ y decimos que el nodo i es un nodo con *déficit*. Si el flujo que entra es igual al flujo que sale, decimos que el nodo está equilibrado.

En la representación de flujo a través de arcos, las variables de decisión son flujos

x_{ij} en el arco $(i,j) \in A$. La representación de flujo en rutas y ciclos inicia con la enumeración de todas las rutas dirigidas P entre cualquier par de nodos y todos los ciclos dirigidos W en la red. Sea $\mathcal{P}$ la colección de de todas las rutas y $\mathcal{W}$ la colección de todos los ciclos. Las variables de decisión en la representación de flujo en ruta y ciclos son $f(P)$, el flujo en P, y $f(W)$, el flujo en el ciclo W; definimos esas variables para todas las rutas dirigidas P en $\mathcal{P}$ y para cada ciclo dirigido W en $\mathcal{W}$

Notemos que el flujo en cada conjunto de rutas y de ciclos puede ser representado de manera única por flujo sobre arcos. El flujo x_{ij} en el arco (i,j) es igual a la suma de los flujos $f(P)$ y $f(W)$ de todas las rutas P y ciclos W que contienen este arco. Formalizaremos esta observación definendo algunas notaciones nuevas: $\delta_{ij}(P)$ es igual a 1 si el arco (i,j) está contenido en la ruta P, y 0 en otro caso. Similarmente, $\delta_{ij}(W)$ es igual a 1 si (i,j) está contenido en el ciclo W, y 0 en otro caso. Entonces

$$x_{ij} = \sum_{P \in \mathcal{P}} \delta_{ij} \cdot f(P) + \sum_{W \in \mathcal{W}} \delta_{ij} \cdot f(W).$$

Por lo tanto cada flujo en rutas y ciclos determina un flujo en arcos único.
Ahora veamos lo contrario, es decir, como representar un flujo en arcos como un flujo en rutas y ciclos, si es que esto es posible. Con el siguiente teorema responderemos la pregunta.

Teorema Apéndice A.1 (Teorema de descomposición de Flujo) *Cada flujo expresado como rutas y ciclos tiene una representación única como flujos no negativos en arcos. Inversamente, cada flujo en arcos puede ser representado como un flujo a través de rutas y ciclos (aunque no necesariamente es única) con las siguientes propiedades:*

(a) Cada ruta dirigida con flujo positivo conecta a un nodo con déficit a un nodo con exceso.
(b) A lo más $n+m$ rutas y ciclos tienen flujo diferente de cero y a lo más m ciclos tienen flujo diferente de cero.

> *Demostración.* La primera parte del teorema se obtiene de las observaciones previas, por lo que sólo nos falta demostrar la segunda afirmación. Daremos una prueba algorítmica para mostrar como descomponer cualquier flujo a través de arcos x a un flujo a lo largo de rutas y ciclos. Supongamos que i_0 es un nodo con déficit. Entonces algún arco (i_0, i_1) lleva un flujo positivo. Si i_1 es un nodo con exceso, nos detenemos; en otro caso, la condición (a) sobre el nodo i_1 implica que algún arco (i_1, i_2) lleva un flujo positivo. Repetimos este argumento hasta que encontramos un nodo con exceso o hasta que llegamos un nodo previamente examinado.

Notemos que uno de esos casos ocurre en n pasos. En el primer caso obtenemos una ruta dirigida P desde un nodo con déficit i_0 hasta un nodo con exceso i_k, mientras que en el segundo caso obtenemos un ciclo dirigido W. En ambos casos la ruta o el ciclo consisten solamente de arcos con flujo positivo.
Si encontramos una ruta dirigida, hacemos:

$$f(P) = \min\{-e(i_0), e(i_k), \min\{x_{ij} : (i,j) \in P\}\},$$

$$e(i_0) = e(i_0) + f(P), \quad e(i_k) = e(i_k) - f(P) \quad y$$

$$x_{ij} = x_{ij} - f(P) \text{ para cada } (i,j) \in P.$$

Si obtenemos un ciclo dirigido W, hacemos:

$$f(W) = \min\{x_{ij} : (i,j) \in W\},$$

$$x_{ij} = x_{ij} - f(W) \text{ para cada } (i,j) \in W.$$

Repetimos este proceso con el problema redefinido hasta que el desequilibrio de todos los nodos sea cero. En ese momento seleccionamos cualquier nodo con al menos un arco saliente con flujo positivo como el nodo inicio, y repetimos el procedimiento, en este caso debemos encontrar un ciclo dirigido. Terminamos cuando $x = 0$ para el problema redefinido. Claramente, el flujo original es la suma de los flujo a través de las rutas y los ciclos encontrados por este método. Ahora observemos que cada vez que identificamos una ruta dirigida, reducimos la oferta/demanda de un nodo a cero o el flujo de un arco a cero, y cada vez que encontramos un ciclo dirigido, reducimos el flujo de un arco a cero. En consecuencia, la representación de flujo a lo largo de rutas y ciclos de un flujo x dado contiene a lo más $n + m$ rutas dirigidas y ciclos, y a lo más m de ellos son ciclos.
◇

Apéndice B. Algoritmo de búsqueda

Los algoritmos de búsqueda son técnicas para encontrar todos los nodos de una red que satisfacen una propiedad particular.

Para ilustrar las ideas básicas de los algoritmos de búsqueda, supongamos que queremos encontrar todos los nodos en una red $G(N, A)$ que son alcanzables a lo largo de rutas dirigidas desde un nodo especial s, llamado origen. El algoritmo hace un barrido desde el origen e identifica los nodos que son alcanzables desde el origen. En cada punto intermedio de su ejecución, el algoritmo designa a todos los nodos de la red uno de dos estados: *etiquetado* o *no etiquetado*. Los nodos etiquetados son nodos que ya han sido alcanzados por el nodo origen, los no etiquetados son los que aún no se han determinado. Notemos que si un nodo i está etiquetado, un nodo j no está etiquetado y la red contiene el arco (i, j), podemos etiquetar al nodo j; ya que este nodo es alcanzable desde el nodo origen vía una ruta dirigida del origen a i más el arco (i, j). Nos referiremos al arco (i, j) como un arco *admisible* si el nodo i está etiquetado y el nodo j no lo está, y nos referiremos como *no admisible* en otro caso.

Inicialmente, etiquetamos sólo el nodo s. Luego, examinando arcos admisibles, el algoritmo etiquetará nodos adicionales. El algoritmo termina cuando ya no hay arcos admisibles en la red. En la descripción algorítmica, $LISTA$ representa el conjunto de nodos etiquetados que el algoritmo va a examinar en el sentido de que algunos arcos admisibles podrían emanar de ellos. Cuando el algoritmo termina, éste ha etiquetado todos los nodos en G que son alcanzables desde s vía una ruta dirigida.

En el Algoritmo Apéndice B.1 mostramos una versión de un algoritmo de búsqueda el cual genera las etiquetas de distancia exactas para cada nodo. En esta variante, el nodo origen es el nodo t y la definición de arcos admisibles cambia un poco; en vez de ser arcos de la forma (i, j) donde i es un nodo etiquetado y j un nodo no etiquetado, un arco (i, j) es *admisible* si i no está etiquetado y j si lo está. Además, el conjunto $LISTA$ lo guardamos como una cola, y siempre tomamos el primer nodo de la cola. Es decir, el algoritmo selecciona los nodos etiquetados en orden PEPS. Estos cambios son los que nos dan como resultado un algoritmo para determinar las etiquetas de distancia exactas, $d(i)$.

Este algoritmo se ejecuta en un tiempo $O(m + n) = O(m)$. Cada iteración del ciclo while encuentra un arco admisible o no. En el primer caso, el algoritmo etiqueta un nuevo nodo y lo agrega a $LISTA$, y en el otro caso borra un nodo de $LISTA$. Ya que el algoritmo etiqueta cualquier nodo a lo más una vez, el ciclo while se ejecuta a lo más $2n$ veces. Ahora consideremos el tiempo que se necesita para identificar arcos admisibles. Por cada nodo i, el algoritmo revisa los arcos en $A'(i) = \{(j, i) : (j, i) \in A\}$

Algoritmo Apéndice B.1 Algoritmo de búsqueda para etiquetas de distancia exactas.

```
begin
  desetiqueta todos los nodos;
  etiqueta el nodo t;
  d(t) ← 0
  LISTA = {s}
  while LISTA ≠ ∅ do
     selecciona el primer nodo i en LISTA;
     if el nodo i es incidente a un arco admisible (j,i) then
        etiqueta j;
        d(j) = d(i) + 1
        agrega j al final de LISTA;
     else
        elimina i de LISTA;
     end if;
  end while;
end;
```

a lo más una vez. Por lo que, el algoritmo examina un total de $\sum_{i \in N} |A(i)| = m$ arcos y entonces termina en un tiempo de $O(m)$.

Índice de figuras

Lista de Algoritmos