UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Flujo máximo en redes con ganancias

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

ACTUARIA

PRESENTA:

CINTIA CARRILLO RIVERA

DIRECTORA DE TESIS:

DRA. CLAUDIA ORQUÍDEA LÓPEZ SOTO

CIUDAD UNIVERSITARIA, CD.MX., 2017

1. Datos de la alumna
Carrillo
Rivera
Cintia
59 16 67 06
Universidad Nacional Autónoma de México
Facultad de Ciencias
Actuaría
309549673

2. Datos de la tutora
Dra.
Claudia Orquídea
López
Soto

3. Datos de la sinodal 1
M. en I.O.
María del Carmen
Hernández
Ayuso

4. Datos de la sinodal 2
Dra.
Bibiana
Obregón
Quintana

5. Datos del sinodal 3
M. en I.
Adrián
Girard
Islas

6. Datos de la sinodal 4
Mat.
Ana Lilia
Anaya
Muñoz

7. Datos del trabajo escrito
Flujo máximo en redes con ganancias
98 p
2017

Agradecimientos

A mis padres, Angélica y Ricardo, les agradezco inmensamente haberme apoyado a lo largo de mi trayectoria académica. Agradezco sus años de esfuerzo para ayudarme a cumplir este objetivo. También les agradezco el haberme enseñado a confiar en mí, a dar mi máximo en todo lo que hago, a levantarme cuando me caigo y, sobre todo, a ser perseverante, porque sólo así se llega al éxito.

A mi hermana Diana, le doy las gracias por soportarme en mis momentos de estrés, por su apoyo incondicional y por ser mi cómplice en todo.

Quiero agradecer de manera especial a la Dra. Claudia Orquídea López Soto por ser mi Directora de Tesis, por su paciencia, por dedicarme su tiempo en horarios no laborales y por ayudarme a que este trabajo quedara lo mejor posible. Asimismo, le agradezco la invitación para participar en las Jornadas estudiantiles de Teoría de Juegos, Teoría de Redes y Programación Lineal llevadas a cabo en mayo de 2017.

Agradezco a mis sinodales, la M. en I.O. María del Carmen Hernández Ayuso, la Dra. Bibiana Obregón Quintana, el M. en I. Adrián Girard Islas y la Mat. Ana Lilia Anaya Muñoz por sus acertadas correcciones y comentarios que ayudaron a enriquecer tanto la parte escrita como el programa que conforman esta tesis.

Dios mueve al jugador, y éste, la pieza. ¿Qué Dios detrás de Dios
la trama empieza de polvo y tiempo y sueño y agonía?
- Jorge Luis Borges

Índice general

Lista de figuras VIII

Introducción 1

1. Rutas más cortas 3
1.1. Introducción . . . 3
1.1.1. Conceptos de Teoría de Gráficas . . . 3
1.2. Planteamiento del problema . . . 5
1.3. Algoritmo de Dijkstra . . . 7
1.3.1. Algoritmo . . . 7
1.3.2. Ejemplo . . . 8
1.4. Algoritmo de Ford . . . 11
1.4.1. Ejemplo . . . 12
1.5. Algoritmo de Floyd . . . 13
1.5.1. Ejemplo . . . 14

2. Flujos 17
2.1. Introducción . . . 17
2.2. Planteamiento del problema . . . 18
2.3. Algoritmo de Ford y Fulkerson . . . 19
2.3.1. Algoritmo . . . 21
2.3.2. Ejemplo . . . 21
2.4. Principales variantes del problema . . . 24
2.4.1. Redes con más de un vértice de origen y más de un vértice terminal . . . 24
2.4.2. Redes con cota inferior y superior de flujo . . . 25

3. Flujo máximo en redes con ganancias 31
3.1. Introducción . . . 31
3.2. Planteamiento del problema . . . 32
3.2.1. Ejemplo . . . 33

3.3. Cadenas Aumentantes 35
3.4. Ciclos activos 37
3.5. Algoritmo 39
3.5.1. Algoritmo 39
3.6. Ejemplo 42

4. Manual del usuario **51**
4.1. Variables 51
4.2. Funciones 52
4.3. Estructura del código 54
4.4. Instrucciones de uso 55

Conclusiones **58**

A. Resultados del ejemplo Cap. 2 **61**

B. Código en R **67**
B.1. Funciones 67
B.2. Procedimiento: Flujo máximo en redes con ganancias 80

Bibliografía **85**

Índice de figuras

1.1. Ejemplo de gráfica 3
1.2. Ejemplo: Circuitos negativos 7
1.3. Ejemplo 1: Rutas más cortas 9
1.4. Ruta más corta del vértice 1 al 5 10
1.5. Arborescencia de rutas más cortas 10
1.6. Red con arcos con $d_{ij} > 0$ y $d_{ij} < 0$ 12
1.7. Solución al ejemplo del algoritmo de Ford 13

2.1. Red inicial 22
2.2. Red con un flujo inicial factible de cero 22
2.3. Flujo correspondiente a la primer iteración 23
2.4. Flujo correspondiente a la segunda iteración 23
2.5. Ejemplo 1 24
2.6. Solución al Ejemplo 1 25
2.7. Red cuyos arcos tienen cota inferior y superior 26
2.8. Red marginal de la Figura 2.3 27
2.9. Flujo máximo sobre la red marginal 28
2.10. Red original actualizada con el flujo inicial factible 28
2.11. Red original con el flujo óptimo 29

3.1. Red con (q_{ij}, g_{ij}) 33
3.2. Patrón de flujo no óptimo 34
3.3. Patrón de flujo óptimo y no máximo 34
3.4. Patrón de flujo óptimo máximo 34
3.5. Ejemplo de cadena aumentante 36
3.6. Patrón de flujo asociado a la Figura 3.5 36
3.7. Ejemplo 3(b) Flujo actualizado 37
3.8. Red original 42
3.9. Red auxiliar 43
3.10. Flujo resultante de la primera iteración 45
3.11. R' de la segunda iteración 45
3.12. Flujo resultante de la segunda iteración 47

3.13. Capacidades de R' 47
3.14. Primera actualización del flujo 48
3.15. Red auxiliar 48
3.16. Segunda actualización del flujo 49
3.17. Red auxiliar de la Figura 3.12 49
3.18. Flujo máximo óptimo 50

4.1. Flujo resultante de la primer iteración 58
4.2. Flujo máximo óptimo 58

Introducción

La Investigación de Operaciones es una disciplina que surge en la segunda guerra mundial, debido a la necesidad de resolver problemas estratégicos y tácticos a través del manejo adecuado y efectivo de los recursos para las operaciones militares. Para ello se formaron equipos interdisciplinarios de especialistas que aplicaban su conocimiento y el método científico para solucionar dichos problemas. Tiempo después, dada la expansión de los mercados en la industria, las grandes empresas recurrieron nuevamente a los grupos interdisciplinarios, con el fin de hallar estrategias de mercado que les permitieran obtener la mayor utilidad posible, sujeta a un nivel de competitividad creciente.

Actualmente, existen procedimientos y técnicas estándares para resolver problemas como: manejo de inventarios, minimización de tiempos, asignación de recursos, maximización de utilidades, distribución de productos y programación de proyectos, entre otros, que tienen su fundamento en la Investigación de Operaciones utilizando herramientas tales como la computación. Debido a la versatilidad de su campo de aplicación, la Investigación de Operaciones forma parte de carreras como Ingeniería Industrial, Sistemas, Civil, Economía, Actuaría, Finanzas, etc.

Una parte importante de problemas de optimización puede analizarse mediante la representación gráfica de los mismos, a través de redes formadas por puntos y líneas que los unen y que representan una relación entre ellos. Por ejemplo, si se desea encontrar el camino más corto para llegar de una ciudad a otra pasando por distintas carreteras y pueblos; las ciudades y pueblos serían representados con puntos, las carreteras con líneas y la relación que establece cada una de las líneas es la distancia entre el par de puntos que une.

El proceso de optimización a través del uso de redes, varía dependiendo de los atributos cuantitativos o cualitativos que se pretendan maximizar o minimizar, y dada la facilidad de su planteamiento, la teoría de redes puede aplicarse a una gran variedad de situaciones. Entre las más comunes se encuentran el ya mencionado problema de rutas más cortas y el problema del flujo máximo. Este último, se re-

fiere al traslado de la mayor cantidad de un cierto producto de un punto a otro y está sujeto a las distintas variables y restricciones que se requieran en dicho proceso.

Ahora bien, si suponemos que en cada etapa de un proceso puede haber una ganancia o pérdida adicional, la tarea de obtener la mayor utilidad final se vuelve un tanto más compleja. A este problema se le denomina flujo máximo en redes con ganancias. Entre sus aplicaciones más usuales se encuentran: problemas de transporte, maximización de reacciones químicas, control de la corriente que circula a través de estructuras eléctricas, diversificación de portafolios financieros, etc.

En este contexto, el objetivo principal de esta tesis es, describir los elementos que componen el algoritmo de flujo máximo en redes con ganancias propuesto en el libro *Graph Theory. An Algorithmic Approach* por Nicos Christofides, así como su justificación.

Al ser una parte fundamental para la solución de dicho problema, resulta de gran importancia hacer una descripción de los algoritmos más comunes para resolver el problema de rutas más cortas. En el primer capítulo se tratan las principales variantes del problema y algunos métodos de solución, haciendo un especial énfasis en el propuesto por Floyd en 1962, ya que fue utilizado como subrutina principal para resolver y programar el problema de flujo en redes con ganancias.

El segundo capítulo contiene una breve introducción al problema del flujo y sus aplicaciones a diversos campos de estudio. Además se describe y ejemplifica el algoritmo de Ford- Fulkerson, el cual es uno de los más eficientes para solucionar este tipo de problemas.

El tercer capítulo detalla los elementos y etapas del algoritmo presentado por Nicos Christofides en 1975, para resolver el problema del flujo máximo en redes con ganancias. Dada la utilidad del algoritmo y su fácil implementación en diversas áreas de conocimiento, se realizó un programa en R que muestra el comportamiento del flujo sobre la red en cada paso del algoritmo hasta alcanzar su máximo.

El último capítulo es un manual del usuario del programa desarrollado en el que se describen las variables, funciones, notación requerida y la forma de interpretar los resultados de cada etapa.

Capítulo 1

Rutas más cortas

1.1. Introducción

Este capítulo tiene como propósito mostrar tres de los métodos más comunes para solucionar el problema de rutas más cortas y algunas de las variantes del mismo. Sin embargo, antes de abordar el problema de rutas más cortas, es necesario considerar los conceptos de la siguiente sección.

1.1.1. Conceptos de Teoría de Gráficas

Una gráfica G es una colección de puntos o vértices representado por el conjunto $X = \{x_1, x_2, ..., x_n\}$ y una colección de líneas representada por el conjunto $A = \{a_1, a_2, ..., a_m\}$ que pueden unir a todos o algunos de los vértices de X. De esta forma, denotamos a G como la pareja (X, A).

Cuando las líneas de A tienen dirección, generalmente representadas con flechas, se les llama arcos se dice que G es una gráfica dirigida. Si las líneas no tienen dirección se les llama aristas y se dice que la gráfica G es no dirigida.

Denotaremos a los arcos como el par (x_i, x_j), donde x_i y x_j representan al vértice inicial y final del arco respectivamente y la dirección del arco se asume del primer al segundo vértice. Por ejemplo, en la gráfica de la Figura 1.1, la pareja (x_1, x_2) denota al arco a_1.

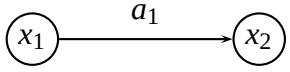


Figura 1.1: Ejemplo de gráfica

Para la gráfica dirigida $G = [X, A]$ con $x_i, x_j \in X$, el vértice x_j es sucesor del vértice x_i si el arco $(x_i, x_j) \in A$. Por otro lado, decimos que el vértice x_j es predecesor de x_i si $(x_j, x_i) \in A$. Al conjunto de vértices sucesores de x_i se le denota como $\Gamma^+(x_i)$. De manera análoga, al conjunto de predecesores de x_i se le denota $\Gamma^-(x_i)$

Definimos una gráfica parcial de $G = [X, A]$ como aquella constituida por todos los vértices de X y algunos arcos o aristas de A. La denotamos como $G_p = [X, A_p]$, donde $A_p \subset A$.

Resulta intuitivo pensar que pueden formarse secuencias de arcos o aristas en una gráfica. En particular, en una gráfica dirigida, un camino es una secuencia de arcos $\{a_1, a_2, ..., a_n\}$, donde el vértice final del arco a_i es el vértice inicial de a_{i+1}.

Un camino $a_1, a_, ..., a_n$ es un camino simple cuando no se recorre el mismo arco más de una vez, es decir, $a_i \neq a_j$ si $i \neq j$. Decimos que un camino es elemental cuando $x_i \neq x_j$ si $i \neq j$, es decir, si no se pasa por el mismo vértice más de una vez.

Una cadena es una secuencia de aristas $\{a_1, a_2, ..., a_n\}$ donde el vértice final de a_i es el vértice inicial de a_{i+1}. Notemos esta definición es análoga a la de un camino para una gráfica no dirigida. Por lo tanto, las definiciones de cadena simple y elemental son análogas a las de camino simple y elemental, respectivamente. Cabe mencionar que un camino o cadena puede ser representado por la secuencia de vértices, arcos o aristas que lo componen, o por una combinación de ambos. Nosotros, denotaremos las cadenas y caminos por los vértices que los integran. Además, si para todo par de vértices $x_i, x_j \in X$ existe una cadena que los une, decimos que la gráfica $G = [X, A]$ es conexa. [1]

Un circuito es un camino $\{x_1, x_2, \ldots, x_a\}$, en el que el vértice inicial x_1 coincide con el vértice final x_a. Y un ciclo es una cadena $\{x_1, x_2, \ldots, x_a\}$ donde $x_1 = x_a$.

Un árbol es una gráfica conexa y acíclica, comúnmente denotada como $T = [X, A]$. En una gráfica dirigida $G = [X, A]$, decimos que $r \in X$ es raíz de G si existe un camino de r a x para todo $x \in X$. Definimos una arborescencia como un árbol que tiene una raíz.

Finalmente, una red se define como una gráfica que tiene asociada una función real f definida sobre sus arcos o aristas, o sobre sus vértices, o ambos, y sus propiedades se extienden de las mismas definidas para gráficas. Denotaremos una red

[1] Para mayor información sobre las propiedades de las gráficas, se recomienda consultar [1]

como $R = [X, A, f]$, donde $X = \{x_1, x_2, ..., x_n\}$ representa el conjunto de vértices, $A = \{a_1, a_2, ..., a_m\}$ es el conjunto de arcos o aristas que unen a todos o algunos de los vértices de X y f es la función mencionada anteriormente.[2]

Algunos de los problemas de optimización que se pueden plantear como un problema de rutas más cortas son: de transporte, de planeación logística, cadenas de producción, reacciones químicas, entre otros, pueden ser representados mediante una red y resolverse utilizando alguno de los algoritmos que se verán más adelante.

A continuación se expondrán tres algoritmos utilizados para resolver el problema de rutas más cortas. En primer lugar, el algoritmo de Dijkstra se utiliza para hallar el camino más corto de un vértice origen a un vértice final donde la distancia entre cualquier par de vértices es positiva. El segundo algoritmo, que es el de Ford, tiene el mismo propósito que el anterior, sin embargo, este admite que el número asociado a los arcos sea negativo.

Por último, veremos el algoritmo de Floyd, el cual se utiliza para calcular el camino más corto entre cualquier par de vértices de la red. Además facilita la identificación de circuitos negativos, por lo que fue utilizado como subrutina principal en la programación y desarrollo del algoritmo presentado por Nicos Christofides para resolver el problema de flujo máximo en redes con ganancias.[3]

1.2. Planteamiento del problema

Consideremos el siguiente problema: La panadería “El Pan”desea transportar sus productos desde su ubicación en la ciudad S a la distribuidora de la ciudad T, atravesando en el camino por diferentes poblados. Por cuestiones de ahorro de insumos, los fabricantes desean que en este traslado se recorra la menor distancia posible.

Este problema puede plantearse sobre una red $R = [X, A, d]$ donde S y T son los vértices inicial y final y el resto de los poblados está representado por vértices, de forma que $X = \{S, x_2, x_3, ..., T\}$, los arcos de A representan los caminos entre los distintos poblados y d representa la distancia que miden dichos caminos. Nuestro objetivo sería encontrar el camino de menor distancia de S a T.

[2] Para mayor información sobre las definiciones anteriores, se recomienda consultar la referencia [2].

[3] El algoritmo de Nicos Christofides se encuentra desarrollado en el Capítulo 3.

Adicionalmente, si los fabricantes desearan dejar sus productos en cada poblado antes de llegar al centro de distribución de la ciudad T, el problema consistiría en encontrar el camino más corto que comience en la ciudad S, recorra cada poblado y finalice en la ciudad T.

Por otro lado, si los fabricantes desearan desplazarse entre las panaderías de algunos poblados a repartir sus productos recorriendo la menor distancia posible, el problema consistiría en encontrar el camino más corto entre todo par de vértices.

Es importante tomar en cuenta que, para que cualquiera de las situaciones planteadas tenga solución, se requiere que exista un camino del vértice inicial al final, del inicial a todos los demás vértices, o entre todo par de vértices, según sea el caso.

Para retomar el planteamiento del problema inicial, se tiene la red $R = [X, A, d]$ donde los vértices de X representan al conjunto de poblados y a las ciudades S y T. Los arcos de A representan los caminos que existen entre los poblados considerados. Finalmente, d_{ij} es un número real asociado a cada arco de A que representa la distancia entre el poblado i y el poblado j. Resulta lógico pensar que no existen distancias negativas, por lo que el número asociado a cada arco debe ser positivo.

Sin embargo, la función asociada a cada arco no siempre representa distancia y es necesario considerar que en algunos problemas, esta función puede tomar valores negativos. Cuando esto sucede, al buscar un camino del vértice inicial al vértice final, puede incurrirse en circuitos negativos[4]. Si esto sucede, decimos que el problema es no acotado, ya que siempre podrá encontrarse un camino de menor longitud[5] en tanto contenga más repeticiones del circuito negativo, por lo tanto, el problema no tiene solución.

Por ejemplo, si consideramos la red de la Figura 1.2, un camino del vértice 1 al 4 es $\{1,2,3,4\}$ de longitud 0 y otro camino más corto es $\{1,2,3,1,2,3,4\}$ de longitud -1. De hecho, cualquier camino que contenga los arcos $(1,2),(2,3),(3,4)$ más n veces el circuito $\{3,1,2,3\}$ con $n > 0$ será de longitud $(1-3+2)+(1+1-3)n = -n$. Notemos que si n tiende a infinito, la longitud del camino tiende a menos infinito, por lo tanto, el problema es no acotado y el camino más corto de 1 a 4 no existe.

[4]Como se mencionó previamente, un circuito es un camino $\{x_1, x_2, \ldots, x_a\}$, en el que el vértice inicial x_1 coincide con el vértice final x_a. Se dice que es negativo si cada vez que se recorre, se obtiene un valor menor.

[5]Obtenemos la longitud de un camino, sumando las d_{ij} de todos los arcos que lo componen.

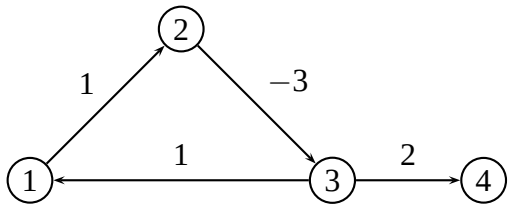


Figura 1.2: Ejemplo: Circuitos negativos

1.3. Algoritmo de Dijkstra

Este método fue propuesto por Edsger Dijkstra en el año 1959 y se utiliza para resolver el problema de rutas más cortas con costos no negativos entre dos vértices de una red, y de un vértice inicial a todos los demás en la red, en cuyo caso el resultado sería una arborescencia de rutas más cortas.[6]

Dada la facilidad para el planteamiento de este problema, existen diversas aplicaciones, entre ellas, el problema de transporte, reducción de costos, insumos o tiempos, etc.

El algoritmo de Dijkstra se basa en la asignación de etiquetas temporales y permanentes a los vértices. Las primeras, sirven para indicar al vértice predecesor y la menor distancia en un camino posible del vértice inicial a este. Las etiquetas permanentes sirven para indicar al predecesor en el camino seleccionado y la distancia total recorrida del vértice inicial al vértice en cuestión. Las etiquetas temporales pueden mejorar en cada iteración del algoritmo hasta obtener el camino más corto posible, la cual está marcado con etiquetas permanentes.

El algoritmo finaliza cuando el vértice final cuenta con una etiqueta permanente, en el caso del problema de rutas más cortas, o cuando todos los vértices de la red tienen etiqueta permanente, en este caso se habrá encontrado la arborescencia de rutas más cortas de raíz S. A continuación se presenta una breve explicación del algoritmo de Dijkstra, y dos ejemplos de aplicación.

1.3.1. Algoritmo

Sea la red $R = [X, A, d]$ donde X y A, representan, respectivamente, el conjunto de vértices y arcos que componen a la red R y d_{ij} es un número asociado a cada

[6]Una arborescencia de rutas más cortas de R es aquella arborescencia donde el único camino del vértice inicial s a todo vértice $x_i \in X$, es un camino más corto de s a x_i. Para mayor información sobre la caracterización de una arborescencia, se recomienda consultar [2].

arco que representa la distancia del vértice x_i al x_j. Ahora, se muestran los pasos para calcular el camino más corto del vértice inicial s al vértice final t.

1. En este paso ningún vértice tiene etiquetas. La variable $d(x_i)$ denota la distancia más corta del vértice inicial s al vértice $x_i \in X$ y $a(x_i)$ representa al vértice predecesor de x_i. Inicializamos el algoritmo asignando las siguientes etiquetas temporales: $d(s) = 0$ y $d(x_i) = \infty$ para todo $x \neq s$. Sea $y = s$.

2. Para todo $x_i \in \Gamma^+(y)$ con etiqueta temporal, calculamos

$$d(x_i) = \text{mín}\{d(x_i), d(y) + d(y, x_i)\} \tag{1.1}$$

   Si $d(x_i) = \infty$, terminar, pues no existen caminos de s a algún vértice. En caso de que $d(x_i)$ cambie, asignarle una etiqueta permanente al mínimo de los $d(x_i)$ calculados y $a(x_i) = y$. Para el siguiente paso, sea $y = x_i$.

3. Repetir el paso 2 hasta que suceda:

   - El vértice final t recibe etiqueta permanente, en este caso se habrá encontrado el camino más corto de s a t. Dicho camino está formado por los vértices con etiqueta permanente y se puede recuperar con los antecesores $a(x_i)$ calculados en cada iteración.
   - Si se desea encontrar la arborescencia de rutas más cortas, el algoritmo termina cuando todos los vértices tengan etiqueta permanente y se recupera con los antecesores $a(x_i)$ de cada iteración.

Para recuperar el camino más corto utilizamos los antecesores de cada iteración que fueron denotados con $a(x)$, comenzando con la última iteración, es decir con el vértice t y terminando en s. En orden inverso se obtiene el camino de s a t.

1.3.2. Ejemplo

Se desea determinar el camino más corto del vértice 1 al 5 en la siguiente red utilizando el algoritmo de Dijkstra:

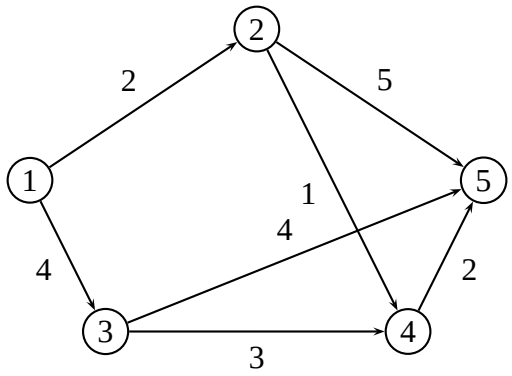


Figura 1.3: Ejemplo 1: Rutas más cortas

Paso 1. Iniciamos con el vértice s etiquetado de manera permanente con $d(1) = 0$, $d(x_i) = \infty$ para $x \neq 1$. Sea $y = 1$.

Paso 2. Recalculamos las etiquetas para los vértices en $\Gamma^+(y)$con etiqueta temporal como:
$d(2) = \min\{d(2), d(1) + d(1,2)\} = \min\{\infty, 0+2\} = 2$
$d(3) = \min\{d(3), d(1) + d(1,3)\} = \min\{\infty, 0+4\} = 4$
La distancia mínima de los vértices etiquetados es $d(2) = 2$, por lo tanto, el vértice 2 recibe etiqueta permanente, ahora $y = 2$ y $a(2) = 1$.

Paso 3. Como 5 no ha sido etiquetado de manera permanente, se repite el Paso 2.

Paso 2. Realculamos las etiquetas para los vértices en $\Gamma^+(2)$ con etiqueta temporal:
$d(4) = \min\{d(4), d(2) + d(2,4)\} = \min\{\infty, 2+1\} = 3$
$d(5) = \min\{d(5), d(2) + d(2,5)\} = \min\{\infty, 2+5\} = 7$
El vértice 4 recibe etiqueta permanente, $y = 4$ y $a(4) = 2$.

Paso 3. El vértice 5 no ha recibido etiqueta permanente, por lo tanto se repite el paso 2.

Paso 2. Calculamos la etiqueta para el vértice 5 en $\Gamma^+(4)$ con etiqueta temporal:
$d(5) = \min\{d(5), d(4) + d(4,5)\} = \min\{\infty, 3+2\} = 5$
El vértice 5 recibe etiqueta permanente y $a(5) = 4$.

Paso 3. Como $t = 5$ ha sido etiquetado de manera permanente, terminar. Finalmente, el camino más corto está marcado con etiquetas permanentes y se obtiene a partir de los antecesores de la siguiente manera: El vértice final es 5, ahora $a(5) = 4$, luego $a(4) = 2$ y $a(2) = 1$ que es el vértice inicial. Por lo tanto el camino más corto del vértice 1 al 5 es $1 \to 2 \to 4 \to 5$ de longitud $d = 5$. Podemos verla marcada en la Figura 1.4.

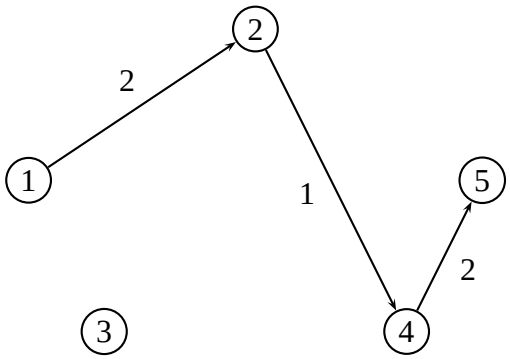


Figura 1.4: Ruta más corta del vértice 1 al 5

Ahora bien, si deseáramos encontrar la arborescencia de rutas más cortas, repetiríamos el algoritmo hasta que todos los vértices estén etiquetados de forma permanente. Para este ejemplo, continuaríamos con la siguiente iteración:

Paso 2. Recalculamos las etiquetas para los vértices en $\Gamma^+(1)$con etiqueta temporal como:
$d(3) = \min\{d(3), d(1)+d(1,3)\} = \min\{\infty, 0+4\} = 4$ El vértice 3 recibe etiqueta permanente, ahora $y = 3$ y $a(3) = 1$.

Paso 3. Como todos los vértices han recibido etiqueta permanente, hemos obtenido la arborescencia de rutas más cortas de raíz 1. Para recuperarla, basta con incluir a la solución anterior que $a(3) = 1$. La arborescencia de rutas más cortas se muestra en la Figura 1.5.

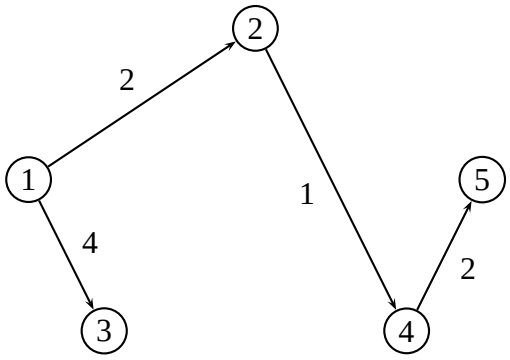


Figura 1.5: Arborescencia de rutas más cortas

1.4. Algoritmo de Ford

Este algoritmo es una modificación del algoritmo de Dijkstra y puede utilizarse cuando los arcos tienen asociados números negativos. La modificación al algoritmo anterior es la siguiente:

1. Se aplica la Ecuación (1.1) del paso 2 a todos los vértices sucesores del vértice que se está examinando, no sólo a los que tienen etiqueta permanente.

2. Si la etiqueta permanente de un vértice x_i mejora, se borran las etiquetas de los vértices tales que $x_i \in \Gamma^+(x_i)$.

3. El algoritmo se termina cuando todos los vértices estén etiquetados y $d(x_i)$ no puede ser mejorada para ningún vértice. [7]

El algoritmo a seguir es el siguiente:

1. En este paso ningún vértice tiene etiquetas. Definimos las variables $d(x_i)$ y $a(x_i)$ como en el algoritmo anterior. Iniciamos con $d(s) = 0$ y $d(x_i) = \infty$ para todo $x \neq s$. Sea $y = s$.

2. Para todo $x_i \in \Gamma^+(y)$ calculamos:

$$d(x_i) = \text{mín}\{d(x_i), d(y) + d(y, x_i)\} \tag{1.2}$$

   Si $d(x_i) = \infty$, terminar, pues no existen caminos de s a algún vértice. En caso de que $d(x_i)$ cambie, asignarle una etiqueta permanente al mínimo de los $d(x_i)$ calculados y $a(x_i) = y$. Para el siguiente paso, sea $y = x_i$.

3. Repetir el paso 2 hasta que suceda:

   - Todos los vértices han sido etiquetados de manera permanente y $d(x_i)$ no puede mejorar para ningún x_i
   - En caso de cambiar $d(x_i)$ para algún $x_i \in X$, se borran las etiquetas de todos los vértices que salen del mismo.

Para recuperar el camino más corto se utilizan los antecesores marcados en cada iteración con $a(x_i)$ siguiendo el procedimiento del algoritmo anterior.

[7] La prueba de que ésta modificación funciona puede consultarse en la referencia [3]

1.4.1. Ejemplo

Consideremos la red de la Figura 1.6 y apliquemos el algoritmo de Ford.

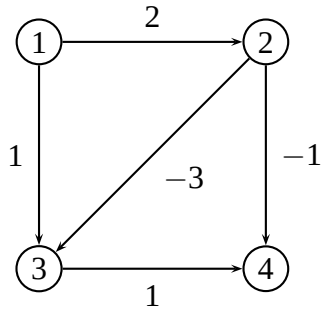


Figura 1.6: Red con arcos con $d_{ij} > 0$ y $d_{ij} < 0$

Paso 1. Inicialmente, el vértice 1 está etiquetado de manera permanente con $d(1) = 0$, $d(x_i) = \infty$ para $x_i \neq 1$. Sea $y = 1$.

Paso 2. Recalculamos la etiqueta de todos los vértices sucesores de $y = 1$:
$d(2) = \min\{d(2), d(1) + d(1,2)\} = \min\{\infty, 0+2\} = 2$
$d(3) = \min\{d(3), d(1) + d(1,3)\} = \min\{\infty, 0+1\} = 1$
La etiqueta mínima de los vértices examinados es $d(3) = 1$, el vértice 3 recibe etiqueta permanente. Sea $y = 3$ y $a(3) = 1$.

Paso 3. Como no todos los vértices tienen etiqueta permanente, se repite el paso 2.

Paso 2. Calculamos $d(x_i)$ para los sucesores del vértice 3:
$d(4) = \min\{d(4), d(3) + d(3,4)\} = \min\{\infty, 1+1\} = 2$
El vértice 4 ha sido etiquetado. Sea $y = 4$ y $a(4) = 3$.

Paso 3. Como no todos los vértices han sido etiquetados, se repite el paso anterior.

Paso 2. Calculamos la etiqueta para el vértice 2:
$d(2) = \min\{d(2), d(1) + d(1,2)\} = \min\{\infty, 0+2\} = 2$
Sea $y = 2$ y $a(2) = 1$.

Paso 3. Todos los vértices han sido etiquetados. Sin embargo, hace falta comprobar si es posible mejorar $d(x_i)$ para algún x. Por lo tanto repetimos el paso anterior utilizando la última etiqueta asignada.

Paso 2. Calculamos $d(x_i)$ para los vértices sucesores de $y = 2$:
$d(3) = \min\{d(3), d(2) + d(2,3)\} = \min\{1, 2-3\} = -1$
$d(4) = \min\{d(4), d(2) + d(2,4)\} = \min\{2, 2-1\} = 1$

La etiqueta mínima de los vértices examinados es $d(3) = -1$, por lo tanto mejora la etiqueta del vértice 3. Sea $y = 3$ y $a(3) = 2$.

Paso 3. Como mejoró la etiqueta del vértice 3, se borran las etiquetas de todos los vértices que salen del mismo. En este caso, se borra la etiqueta del vértice 4, por lo que es necesario repetir el paso anterior.

Paso 2. Calculamos $d(x_i)$ para los vértices sucesores de $y = 3$:
$d(4) = \min\{d(4), d(3) + d(3,4)\} = \min\{2, -1+1\} = 0$
El vértice 4 ha sido etiquetado. Sea $a(4) = 3$.

Paso 3. Todos los vértices han sido etiquetados, además no es posible reducir $d(x_i)$ para ningún x. Por lo tanto, termina el algoritmo y el camino más corto es $1 \to 2 \to 3 \to 4$ con $d = 0$. En la Figura 1.7 se muestra el camino más corto del vértice 1 al 4.

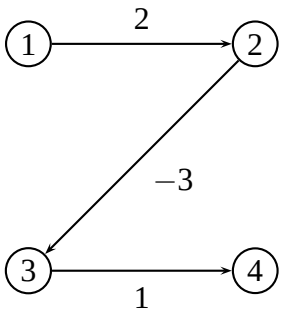


Figura 1.7: Solución al ejemplo del algoritmo de Ford

1.5. Algoritmo de Floyd

El algoritmo de Floyd se utiliza para calcular el camino más corto entre cualquier par de vértices de una red $R = [X, A, d]$. A grandes rasgos, el algoritmo funciona como sigue: Sea una red $R = [X, A, d]$ con $X = \{1, 2, 3, \cdots, n\}$, en cada iteración $k = \{1, \cdots, n\}$, el algoritmo calcula el camino más corto del vértice i pasando por k llegando a j para cualquier par de vértices de la red. La distancia de los caminos más cortos se va actualizando en cada iteración sobre una matriz C de dimensiones $n \times n$. Al mismo tiempo el camino más corto que deba seguirse se va actualizando en una matriz Z de dimensiones $n \times n$, de tal manera que al finalizar el algoritmo será posible recuperar la distancia mínima entre todo par de nodos con su respectivo camino más corto.

Al inicializar el algoritmo se construyen las matrices C y Z de la siguiente manera: Es condición necesaria que no existan bucles[8] en la red, por lo que $C[i,i] = 0$ $\forall x_i$, además si $(x_i, x_j) \notin A$ entonces $C[i,j] = \infty$ y si $(x_i, x_j) \in A$, entonces, $C[i,j] = d(x_i, x_j)$. Continuamos con la inicialización de la matriz de predecesores, en este caso $z[i,j] = i \; \forall x_i, x_j \in X$.

En cada iteración k donde $k = \{1, 2, \cdots, n\}$ se calcula la distancia más corta entre todo par de vértices (x_i, x_j) como $C[i,j] = \text{mín}\{C[i,j], C[i,k] + C[k,j]\}$. Si $C[i,j] \neq \infty$ y fue modificada, es decir, si ocurre que $C[i,j] = C[i,k] + C[k,j]$, entonces $z[i,j] = z[k,j]$.

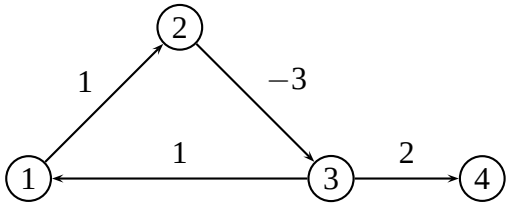
Es importante mencionar que este algoritmo se detiene una vez finalizadas las k iteraciones, es decir, cuando se ha calculado la ruta más corta entre todo par de vértices de la red. O bien, se detiene cuando existen circuitos negativos sobre la red, en este caso, si $C[i,i] < 0$ ó $z[i,i] \neq i$ para algún $i \in \{1, ..., n\}$, entonces existe un circuito negativo sobre la red y no es posible obtener una solución.

Ahora bien, al finalizar las n iteraciones del algoritmo se puede recuperar el camino más corto de i a j utilizando la matriz de antecesores, z, como sigue: Supongamos que $z[i,j] = b$, entonces b es el nodo inmediato anterior antes de llegar de i a j. Siguiendo esto, si $z[i,b] = a$, entonces a es el vértice inmediato anterior antes de llegar de i a b. Este proceso se repite hasta llegar a $z[i,k] = i$.

La ruta obtenida va de j a i, sin embargo al leerla en orden inverso se obtiene la trayectoria deseada.

1.5.1. Ejemplo

Para ejemplificar la forma en la que este algoritmo detecta circuitos negativos, se le aplicará a la Red de la Figura 1.2:



Las matrices iniciales asociadas a la red R (Figura 1.2) son:

[8] Se dice que hay un bucle en la red $R = [X, A]$ si $(x_i, x_i) \in A$ para algún vértice $x_i \in X$.

$$C = \begin{bmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & -3 & \infty \\ 1 & \infty & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix} \qquad z = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

A continuación se muestran los cálculos para la matriz C en la primera iteración.

- $C[1,1] = \min\{C[1,1], C[1,1]+C[1,1]\} = \min\{0,0\} = 0$
- $C[1,2] = \min\{C[1,2], C[1,1]+C[1,2]\} = \min\{1,0+1\} = 1$
- $C[1,3] = \min\{C[1,3], C[1,1]+C[1,3]\} = \min\{\infty,0+\infty\} = \infty$
- $C[1,4] = \min\{C[1,4], C[1,1]+C[1,4]\} = \min\{\infty,0+\infty\} = \infty$
- $C[2,1] = \min\{C[2,1], C[2,1]+C[1,1]\} = \min\{\infty,\infty+0\} = \infty$
- $C[2,2] = \min\{C[2,2], C[2,1]+C[1,2]\} = \min\{0,\infty+1\} = 0$
- $C[2,3] = \min\{C[2,3], C[2,1]+C[1,3]\} = \min\{-3,\infty\} = -3$
- $C[2,4] = \min\{C[2,4], C[2,1]+C[1,4]\} = \min\{\infty,\infty\} = \infty$
- $C[3,1] = \min\{C[3,1], C[3,1]+C[1,1]\} = \min\{1,1+0\} = 1$
- $C[3,2] = \min\{C[3,2], C[3,1]+C[1,2]\} = \min\{\infty,1+1\} = 2$
- $C[3,3] = \min\{C[3,3], C[3,1]+C[1,3]\} = \min\{0,1+\infty\} = 0$
- $C[3,4] = \min\{C[3,4], C[3,1]+C[1,4]\} = \min\{2,1+\infty\} = 2$
- $C[4,1] = \min\{C[4,1], C[4,1]+C[1,1]\} = \min\{\infty,\infty+0\} = \infty$
- $C[4,2] = \min\{C[4,2], C[4,1]+C[1,2]\} = \min\{\infty,\infty+1\} = \infty$
- $C[4,3] = \min\{C[4,3], C[4,1]+C[1,3]\} = \min\{\infty,\infty\} = \infty$
- $C[4,4] = \min\{C[4,4], C[4,1]+C[1,4]\} = \min\{0,\infty\} = 0$

Entonces la matriz actualizada es:

$$C = \begin{bmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & -3 & \infty \\ 1 & 2 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

Como cambió la entrada $C[3,2]$, entonces $z[3,2] = z[1,2] = 1$ y la matriz correspondiente es:

$$z = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Pasamos a la segunda iteración. El cálculo de C es:

- $C[1,1] = \text{mín}\{C[1,1], C[1,2] + C[2,1]\} = \text{mín}\{0, 1+\infty\} = 0$
- $C[1,2] = \text{mín}\{C[1,2], C[1,2] + C[2,2]\} = \text{mín}\{1, 1+0\} = 1$
- $C[1,3] = \text{mín}\{C[1,3], C[1,2] + C[2,3]\} = \text{mín}\{\infty, 1-3\} = -2$
- $C[1,4] = \text{mín}\{C[1,4], C[1,2] + C[2,4]\} = \text{mín}\{\infty, 1+\infty\} = \infty$
- $C[2,1] = \text{mín}\{C[2,1], C[2,2] + C[2,1]\} = \text{mín}\{\infty, 0+\infty\} = \infty$
- $C[2,2] = \text{mín}\{C[2,2], C[2,2] + C[2,2]\} = \text{mín}\{0, 0+0\} = 0$
- $C[2,3] = \text{mín}\{C[2,3], C[2,2] + C[2,3]\} = \text{mín}\{-3, 0-3\} = -3$
- $C[2,4] = \text{mín}\{C[2,4], C[2,2] + C[2,4]\} = \text{mín}\{\infty, 0+\infty\} = \infty$
- $C[3,1] = \text{mín}\{C[3,1], C[3,2] + C[2,1]\} = \text{mín}\{1, 2+\infty\} = 1$
- $C[3,2] = \text{mín}\{C[3,2], C[3,2] + C[2,2]\} = \text{mín}\{2, 2+0\} = 2$
- $C[3,3] = \text{mín}\{C[3,3], C[3,2] + C[2,3]\} = \text{mín}\{0, 2-3\} = -1$

Como cambiaron las entradas C[1,3] y C[3,3], ahora $z[1,3] = z[2,3] = 2$ y $z[3,3] = z[2,3] = 2$. Las matrices correspondientes a esta iteración son:

$$C = \begin{bmatrix} 0 & 1 & -2 & \infty \\ \infty & 0 & -3 & \infty \\ 1 & 2 & -1 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix} \qquad z = \begin{bmatrix} 1 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 1 & 2 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Como $C[3,3] < 0$ y $z[3,3] \neq 3$, existe un circuito negativo en la red. Para recuperarlo utilizaremos la matriz z comenzando en $z[3,3] = 2$, luego $z[3,2] = 1$ y $z[3,1] = 3$, entonces el circuito negativo es $\{3,1,2,3\}$ y por lo tanto no se puede obtener una solución óptima.

En el siguiente capítulo se expone el problema del flujo máximo y algunas variantes del problema.

Capítulo 2

Flujos

2.1. Introducción

En este capítulo planteamos el problema en el que se desea transportar cierta cantidad de la mercancía de una fábrica a un centro de distribución y que en el proceso de transporte se puede pasar por diferentes puntos de la ciudad. Además existe una capacidad máxima de mercancía que puede transportarse entre cada par de puntos de la ciudad. A dicha mercancía le llamaremos flujo y un planteamiento de este problema es el siguiente:

Se desea conocer la cantidad máxima de flujo que puede pasar a través de la red *R=[X,A,q]*, de una fuente específica, el vértice *s*, a una terminal, el vértice *t*. En este sentido, se supone también que si $x_i, x_j \in X$ y $(x_i, x_j) \in A$, entonces el arco (x_i, x_j) tiene asociada una capacidad q_{ij} que representa el límite máximo de flujo que puede pasar del vértice x_i al x_j.

Uno de los algoritmos más utilizados para resolver este tipo de problemas es el de Ford y Fulkerson (1957) y a partir de este surgen diversas variantes. Entre las más comunes se encuentran:

(i) Cuando existen varias fuentes $(s_1, s_2, s_3, ..., s_m)$ o varias terminales $(t_1, t_2, t_3, ..., t_n)$ en la red $R = [X, A, q]$.

(ii) Cuando además de capacidades, los arcos tienen asociada una cota inferior r_{ij}. En este caso desea determinarse un flujo factible, es decir, un flujo que cumpla con las condiciones de límite inferior y superior para cada arco. Si además se asocia costo c_{ij} a cada unidad de flujo que pasa por el arco (x_i, x_j), el problema es encontrar un *flujo a costo mínimo*.

En este capítulo se muestra el desarrollo y fundamentos del algoritmo básico de

Ford y Fulkerson y una breve descripción de los casos (i) y (ii). Posteriormente, se profundizará en otra variante del problema llamada *flujo en redes con ganancias* con base en el algoritmo presentado por Nicos Christofides.

2.2. Planteamiento del problema

Considérese la red $R = [X, A, q]$ con un vértice inicial (origen) s y uno terminal t. Además, a cada arco $(x_i, x_j) \in A$ se le asigna una cantidad ξ_{ij} denominada flujo, este es factible si se satisface que:

$$\sum_{x_j \in \Gamma^+(x_i)} \xi_{ij} - \sum_{x_k \in \Gamma^-(x_i)} \xi_{ki} = \begin{cases} v \text{ si } x_i = s \\ -v \text{ si } x_i = t \\ 0 \text{ si } x_i \neq s \text{ o t} \end{cases} \tag{2.1}$$

Y además cumple que para todo $(x_i, x_j) \in A$, $0 \leq \xi_{ij} \leq q_{ij}$.

A la expresión (2.1) se le conoce como ecuaciones de conservación de flujo y establecen que el flujo que entra a cada vértice es igual al flujo que sale del mismo, excepto para s y t. Además se debe cumplir que $0 \leq \xi_{ij} \leq q_{ij} \forall (x_i, x_j) \in A$, es decir, que el flujo a través de cada arco debe ser menor o igual a la capacidad del mismo. En este sentido, el objetivo del problema general de flujo es encontrar una distribución del flujo tal que se maximice:

$$\begin{aligned} v &= \sum_{x_j \in \Gamma^+(s)} \xi_{sj} - \sum_{x_i \in \Gamma^-(s)} \xi_{is} \\ &= \sum_{x_k \in \Gamma^-(t)} \xi_{kt} - \sum_{x_l \in \Gamma^+(t)} \xi_{tl} \end{aligned} \tag{2.2}$$

El Teorema del *Flujo máximo- corte mínimo* constituye una herramienta de optimalidad para este problema, ya que sirve para determinar si el flujo sobre una red es máximo. Sin embargo, antes de citarlo es necesario conocer lo siguiente:

Una cortadura $(X_0 \to \tilde{X_0})$ es un conjunto de arcos que separa a s de t si $s \in X_0$ y $t \in \tilde{X_0}$. Además si quitamos este conjunto de arcos de la red, no existe cadena alguna del vértice inicial s al vértice final t. El valor de esta cortadura es la suma de las capacidades de todos los arcos de A, cuyos vértices iniciales están en X_0 y los finales están en $\tilde{X_0}$, entonces

$$v(X_0 \to \tilde{X_0}) = \sum_{(x_i, x_j) \in (X_0 \to \tilde{X_0})} q_{ij} \tag{2.3}$$

TEOREMA: FLUJO MÁXIMO - CORTE MÍNIMO
El valor del flujo máximo del vértice s al vértice t es igual al valor de la cortadura mínima que separa a s de t. [1]

De esta forma la cortadura mínima es la cortadura donde $v(X_0 \to \tilde{X_0})$ es mínimo.

El problema del flujo máximo tiene aplicaciones en diversos contextos, por ejemplo, se utiliza para optimizar procesos de fabricación, sistemas de comunicación y programación logística. También es utilizado como subrutina para resolver problemas más complejos, entre ellos, comprobar la existencia de una solución factible en el problema de flujo a costo mínimo.

La estructura del problema de rutas más cortas es complementaria a la estructura del problema de flujo máximo y ambas son la base de diversos algoritmos. En este sentido, los algoritmos de rutas más cortas pueden ser vistos como flujo máximo-costo mínimo sin considerar las capacidades de los arcos, mientras que los algoritmos de flujo máximo pueden verse como de flujo máximo- costo mínimo sin considerar los costos de los arcos.

2.3. Algoritmo de Ford y Fulkerson

El algoritmo de Ford y Fulkerson resuelve el problema general del flujo máximo iniciando con un flujo factible conocido, por ejemplo un flujo igual a cero en todos los arcos de la red, y recursivamente se construyen secuencias de flujo de mayor valor hasta llegar al máximo. Antes de conocer más acerca de dicho algoritmo, es necesario tomar en consideración las siguientes definiciones:

Sea $R = [X, A, q]$, consideremos una cadena C del vértice inicial s al vértice final t y sea $(x_i, x_j) \in A$ un arco que forma parte de C. Definimos los siguientes subconjutos de C:

- $(x_i, x_j) \in C^+$ si tiene sentido de s a t
- $(x_i, x_j) \in C^-$ si tiene sentido de t a s

Decimos que C es aumentante si $\xi_{ij} < q_{ij}$ para todos los arcos $(x_i, x_j) \in C^+$ y $\xi_{ij} > 0$ para todo $(x_i, x_j) \in C^-$. Se denomina cadena aumentante debido a que

[1] La justificación y demostración de este teorema se encuentra en [4]

puede enviarse flujo a través de ella, alcanzando con esto un flujo factible de mayor valor.[2]

En caso de que sea posible incrementar el valor de la cadena aumentante en δ unidades, debe cumplirse que $\xi_{ij} + \delta \leq q_{ij}$ para todos los arcos en C^+ y $\xi_{ij} - \delta \geq 0$ para todo arco en C^-. Al buscar el flujo máximo en R, el objetivo es que δ tenga el mayor valor posible. En este sentido, la capacidad incremental δ, es la máxima cantidad de flujo que puede enviarse de s a t a través de la cadena aumentante. Se calcula con la Fórmula 2.4:

$$\delta = \text{mín}\{\min_{(i,j)\in C^+}\left(q_{ij} - \xi_{ij}\right), \min_{(i,j)\in C^-}\left(\xi_{ij}\right)\} \tag{2.4}$$

Después de tener un flujo inicial factible, se utiliza una subrutina de etiquetado para encontrar una cadena aumentante del vértice s al vértice t. Si la cadena aumentante existe, se actualiza el flujo que puede pasar a través de ella, aumentando así el flujo que pasa por la red. Una vez actualizado el flujo, se repite este procedimiento hasta que no se encuentre ninguna cadena aumentante del vértice s al vértice t. En este caso, ξ es el flujo máximo.

Consideremos el problema del flujo máximo en la red $R = [X, A, q]$ y donde cada vértice $x_i \in X$ tiene uno de los siguientes estados:

- Etiquetado y escaneado: El vértice ha sido etiquetado y todos sus vértices adyacentes han sido examinados.

- Etiquetado y no escaneado: El vértice ha sido etiquetado y no todos sus vértices adyacentes han sido examinados.

- Vértice no etiquetado.

Las etiquetas de cada vértice son de la forma $(+x_j, \delta)$ o $(-x_j, \delta)$. Cuando la primera coordenada es de la forma $+x_j$, el flujo puede incrementar a través del arco (x_j, x_i). Cuando la primera coordenada es del tipo $-x_j$, el flujo puede disminuir a través del arco (x_i, x_j). Para ambos casos, δ es la cantidad de flujo que puede pasar de s a x_i a través de la cadena aumentante que se está construyendo. De esta forma, al etiquetar x_i, se está construyendo una cadena aumentante de flujo de s a x_i.

[2] Se recomienda consultar el libro de la referencia [2] para mayor información sobre las cadenas aumentantes.

2.3.1. Algoritmo

A continuación se presenta el algoritmo Ford y Fulkerson. Para inicializar el algoritmo, todos los vértices están sin etiquetar.

1. Iniciar con un flujo factible sobre la red, éste puede ser cero. Etiquetamos s con $(+s, \delta(s) = \infty)$

2. Elegir un vértice x_i no etiquetado y no examinado cambiar su status por $(\pm x_k, \delta(x_i))$.

   - Para todos los vértices $x_j \in \Gamma^+(x_i)$ que no estén etiquetados y con $\xi_{ij} < q_{ij}$, etiquetar $(+x_i, \delta(x_j))$, con $\delta(x_j) = \text{mín}\{delta(x_i), q_{ij} - \xi_{ij}\}$.
   - Para todos los vértices $x_j \in \Gamma^-(x_i)$ que no estén etiquetados y con $\xi_{ji} > 0$, etiquetar $(-x_i, \delta(x_j))$, con $\delta(x_j) = \text{mín}\{delta(x_i), \xi_{ij}\}$.

   Ahora el vértice x_i está etiquetado y escaneado y x_j está etiquetado y no escaneado.

3. Repetir el paso 2 hasta que ocurra:

   - Si t es etiquetado, pasar a 4.
   - Si t no está etiquetado y ya no puede colocarse otra etiqueta, terminar el algoritmo. En este caso ξ es el flujo máximo. Sea X_0 es el conjunto de nodos etiquetados y $\tilde{X_0}$ es el conjunto de vértices sin etiquetar, entonces los arcos de la forma $(X_0 \to \tilde{X_0})$ representa la cortadura mínima.

4. Actualización del flujo. Sea $x = t$

5. Si la etiqueta de x es de la forma $(+z, \delta(x))$, el flujo pasa de ξ_{zx} a $\xi_{zx} + \delta(t)$. Si la etiqueta de x es de la forma $(-z, \delta(x)$, el flujo pasa de ξ_{xz} a $\xi_{xz} - \delta(t)$.

6. Si $z = s$, borra todas las etiquetas y repetir el algoritmo con el flujo actualizado. Si $z \neq s$, sea $x = z$ y regresar al paso 5.[3]

2.3.2. Ejemplo

Encontrar el flujo máximo utilizando el algoritmo de Ford y Fulkerson sobre la red de la Figura 2.1. A cada arco se asocia q_{ij}.

[3]Para mayor detalle sobre el algoritmo de Ford y Fulkerson consultar las referencias [5], [2], [1].

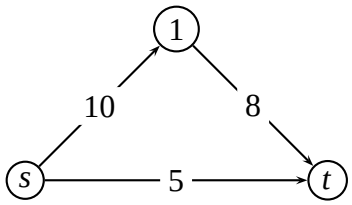


Figura 2.1: Red inicial

Paso 1. Iniciamos el algoritmo con un flujo factible de cero sobre la red. A cada arco de la Figura 2.2 se le asocia (ξ_{ij}, q_{ij}).

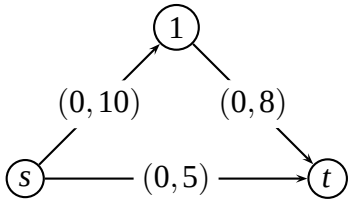


Figura 2.2: Red con un flujo inicial factible de cero

Paso 2. Asignamos al vértice s la etiqueta $(+s,\infty)$ y etiquetamos a los vértices $x_j \in \Gamma^+(s)$ que no estén etiquetados y con $\xi_{sj} < q_{sj}$:

- $(+s,\delta(1))$, con $\delta(1) = \min\{\delta(s), q_{s1} - \xi_{s1}\} = \min\{\infty, 10-0 = 10$. Entonces la etiqueta de 1 es $(+s,10)$.
- $(+s,\delta(t))$ con $\delta(t) = \min\{\delta(s), q_{st} - \xi_{st}\} = \min\{\infty, 5-0 = 5$. Y la etiqueta de t es $(+s,5)$ Decimos que s está etiquetado y escaneado. Los vértices 1 y t están etiquetados y no escaneados.

Paso 3. Como t ha sido etiquetado, pasar a 4.

Paso 4. Actualización del flujo.

Paso 5. Si la etiqueta de x es de la forma $(+z,\delta(x))$, el flujo pasa de ξ_{zx} a $\xi_{zx} + \delta(t)$. Entonces el flujo que atravieza por el arco (s,t) es $\xi_{st} + \delta(t) = 0 + 5$.

Paso 6. Se borran todas las etiquetas y se repite el algoritmo con el flujo actualizado mostrado en la red de la Figura2.3.

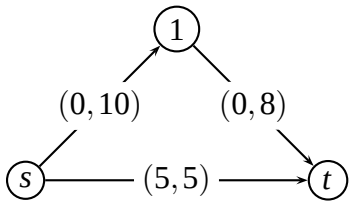


Figura 2.3: Flujo correspondiente a la primer iteración

Paso 2. Etiquetamos s con $(+s,\infty)$ y a los vértices $x_j \in \Gamma^+(s)$ que no estén etiquetados y con $\xi_{sj} < q_{sj}$:

- $(+s,\delta(1))$, con $\delta(1) = \text{mín}\{\delta(s), q_{s1} - \xi_{s1}\} = \text{mín}\{\infty, 10-0 = 10$ y la etiqueta es $(+s,10)$. El vértice s está etiquetado y escaneado. Los vértices 1 y 2 están etiquetados y no escaneados.

Paso 3. Repetir el paso 2.

Paso 2. Etiquetamos a los vértices $x_j \in \Gamma^+(1)$ que no estén etiquetados y con $\xi_{1j} < q_{1j}$:

- $(+1,\delta(t))$, con $\delta(t) = \text{mín}\{\delta(1), q_{1t} - \xi_{1t}\} = \text{mín}\{\infty, 8-0 = 8$ y la etiqueta es $(+1,8)$. El vértice 1 está etiquetado y escaneado.

Paso 3. Como t ha sido etiquetado, pasar a 4.

Paso 4. Actualización del flujo.

Paso 5. El flujo que atravieza por el arco (s,t) es $\xi_{st} + \delta(t) = 0+5$.

Paso 6. Se repite el algoritmo con el flujo actualizado mostrado en la red de la Figura2.4.

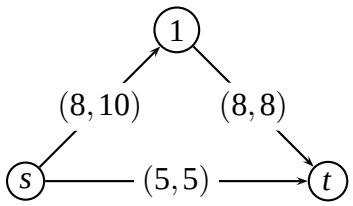


Figura 2.4: Flujo correspondiente a la segunda iteración

Notemos que en la siguiente iteración del algoritmo, ya no es posible incrementar el flujo de s a t. Por lo tanto se ha llegado al corte mínimo formado por los arcos $(X_0 \to \tilde{X_0}) = \{(1,t),(s,t)\}$. Además, se cumple que $v(X_0 \to \tilde{X_0}) = 8+5 = 13$ es igual al valor del flujo obtenido, por lo tanto el valor del flujo máximo es $\xi = 13$.

2.4. Principales variantes del problema

Algunas de las variantes más comunes del problema son:

2.4.1. Redes con más de un vértice de origen y más de un vértice terminal

Cuando existen varias fuentes $(s_1, s_2, s_3, ..., s_m)$ o varias terminales $(t_1, t_2, t_3, ..., t_n)$ en la red $R = [X, A, q]$ se construye una red auxiliar $R' = [X', A', q']$, donde:

$X' = \{s,t\} \cup X$
$A' = A \cup \{\{(s,s_i)|i = 1,2...m\} \cup \{(t_j,t)|j = 1,2...n\}\}$
$q'_{ij} = \begin{cases} q_{ij} \text{ si } (x_i,x_j) \in A \\ \infty \text{ si } i = s \text{ ó } j = t \end{cases}$

En este caso el problema se reduce a encontrar la máxima cantidad de flujo que puede pasar del vértice s al t en R', que es el flujo máximo en la red original R. En la siguiente red (Figura 2.5) se muestra un ejemplo, el número asociado a cada arco representa su capacidad q_{ij}.

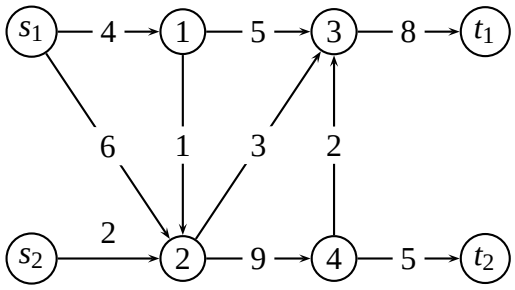


Figura 2.5: Ejemplo 1

Notemos que la red anterior tiene dos vértices iniciales, s_1 y s_2, y dos vértices finales, t_1 y t_2. Al agregar vértices ficticios de origen y destino con capacidad ∞ en los arcos (s', s_1), $(s's_2)$, (t_1, t') y (t_2, t'), es posible usar el algoritmo de Ford y Fulkerson con el cual se obtiene la solución que se muestra en la Figura 2.6. A

cada arco se asocia (ξ_{ij}, q_{ij}).

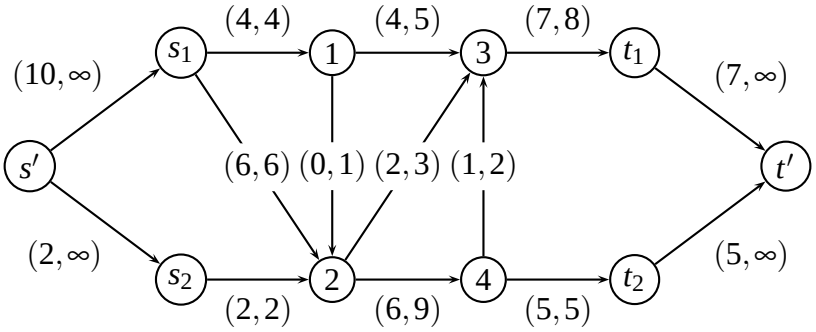


Figura 2.6: Solución al Ejemplo 1

En la siguiente iteración del algoritmo de Ford y Fulkerson, el vértice s' recibe la etiqueta permanente $(+s', \infty)$ y sus sucesores, los vértices s_1 y s_2, reciben la etiqueta temporal $(+s', \infty)$. Luego ningún vértice en $\Gamma^+(s_1)$ puede ser etiquetado pues $\xi_{ij} = q_{ij}$. Lo mismo sucede para el vértice en $\Gamma^+(s_2)$.

Por lo tanto, termina el algoritmo y se llega al corte formado por los arcos $(X_0 \to \tilde{X_0}) = \{(s_1,1),(s_1,2),(s_2,2)\}$. Efectivamente, se cumple que $v(X_0 \to \tilde{X_0}) = 4 + 6 + 2 = 12$ es igual al valor del flujo obtenido, por lo tanto el flujo máximo es $\xi = 12$. Fácilmente puede verse que al eliminar los vértices ficticios s' y t' y los arcos que los unen a R, el resultado obtenido es el mismo para la red original.

2.4.2. Redes con cota inferior y superior de flujo

En este caso, un flujo factible es aquel que satisface los límites de capacidad de cada arco. Sea la red $R = [X, A, q, r]$, donde r_{ij} representa el flujo mínimo que puede circular a través del arco (x_i, x_j). En este tipo de redes, además de las ecuaciones de conservación de flujo, se debe cumplir que $\forall (x_i, x_j) \in A, r_{ij} \leq \xi_{ij} \leq q_{ij}$. Por otro lado, si $r_{ij} = 0$ para todos los arcos $(x_i, x_j) \in A$, entonces nos encontramos en el caso anterior.

La Figura 2.7 muestra un ejemplo de este tipo de red. A cada arco le asociamos $(r_{ij}, q_{i,j})$.

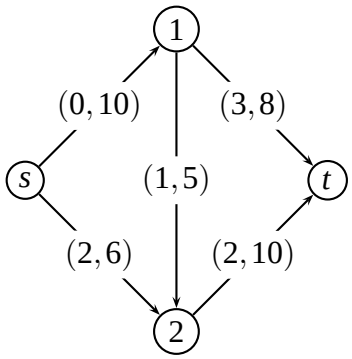


Figura 2.7: Red cuyos arcos tienen cota inferior y superior

Para resolver este tipo de problemas, es necesario comprobar la existencia de un flujo factible, pues un flujo de cero para todos los arcos de la red puede no resultar factible, dado que los límites inferiores de los arcos pueden ser mayores a cero. Para encontrar el flujo factible se construye una red auxiliar $R' = [X', A', q']$, constituida de la siguiente manera:

$X' = X \cup \{s', t'\}$
$A' = A \cup (t,s) \cup \{(s', x_j), (x_i, t') \text{ para } (x_i, x_j) \in A \text{ con } r_{ij} \neq 0)\}$
$$q' = \begin{cases} q'_{ij} = q_{ij} - r_{ij} \text{ para } (x_i, x_j) \in A \\ q'_{ij} = r_{ij} \text{ para } \{(s', x_j), (x_i, t')\} \in A \\ q'_{ts} = \infty \end{cases}$$

Notemos que $r_{ij} = 0$ para todos los arcos de R'. En caso de que un vértice tenga más de un arco entrante, se suman todas las cotas inferiores de flujo para formar un sólo arco auxiliar del tipo (s', x_j). Se procede de forma análoga para construir los arcos (x_i, t').

Con ayuda del algoritmo de Ford y Fulkerson se calcula el flujo máximo de s' a t'. Cuando este es de valor $v' = \sum r_{ij}$ donde $r_{ij} \neq 0$, se dice que existe un flujo factible de valor ξ'_{ts} en la red original R.

Una vez calculado v', se actualiza el flujo en la red original como $\xi_{ij} = \xi'_{ij} + r_{ij}$ para todos los arcos donde $r_{ij} \neq 0$ y $\xi_{ij} = \xi'_{ij}$ para los arcos donde $r_{ij} = 0$ y se procede a encontrar el flujo máximo del vértice s al vértice t utilizando el algoritmo de Ford y Fulkerson con una modificación que consiste en asignar etiquetas $[-j, \xi_i]$ a los vértices no etiquetados con $\xi_{ij} > r_{ij}$, donde $\xi_i = min\{\xi_j, \xi_{ij} - r_{ij}\}$

Como el flujo a través de cada arco tiene límite inferior y superior, también se

modifica la forma de calcular la capacidad de la cortadura mínima a:

$$v(X_0 \to \tilde{X_0}) = \sum_{(x_i,x_j)\in(X_0\to\tilde{X_0})} q_{ij} - \sum_{(x_i,x_j)\in(\tilde{X_0}\to X_0)} r_{ij} \tag{2.5}$$

En la Figura 2.8 se muestra la red auxiliar R' correspondiente al problema planteado en la Figura 2.7. A cada arco se asocia q'_{ij}.

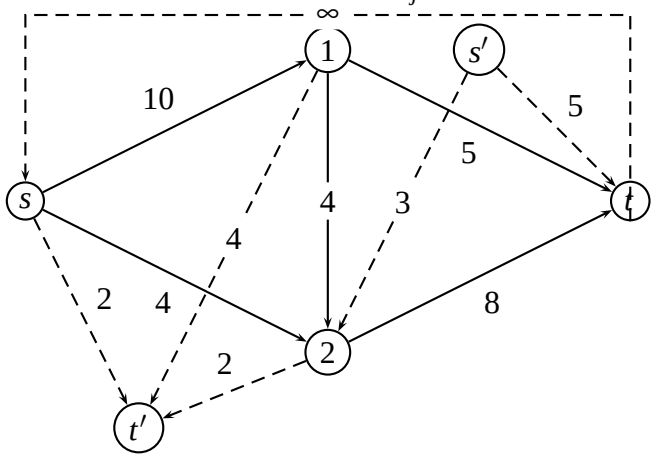


Figura 2.8: Red marginal de la Figura 2.3

Al aplicar el algoritmo de Ford y Fulkerson se obtiene la Figura 2.9. A cada arco se asocia (ξ'_{ij}, q'_{ij}). Notemos que $v' = 8$ que es igual a la suma de todas las cotas inferiores. Por lo tanto, existe un flujo factible de valor $\xi'_{ts} = 6$ en la red original R.

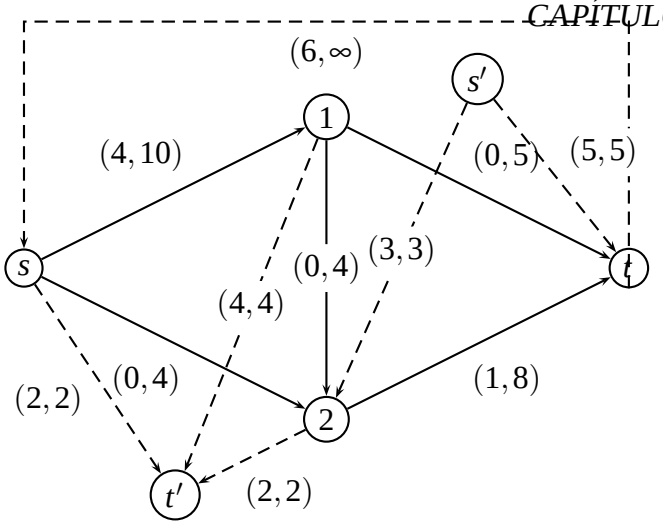


Figura 2.9: Flujo máximo sobre la red marginal

Posteriormente se actualiza el flujo sobre la red original obteniendo un flujo factible de valor $v = 6$, el cual se muestra en la Figura 2.10. A cada arco se asocia la triada $(r_{ij}, \xi_{ij}, q_{ij})$.

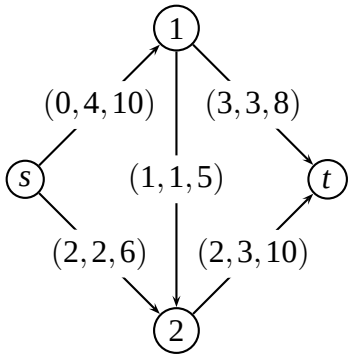


Figura 2.10: Red original actualizada con el flujo inicial factible

Maximizamos el flujo de la red de la Figura 2.10 utilizando el algoritmo de Ford y Fulkerson con la variante previamente mencionada. El flujo obtenido en la última iteración se muestra en la Figura 2.11. Los arcos se definen igual que en la red anterior.

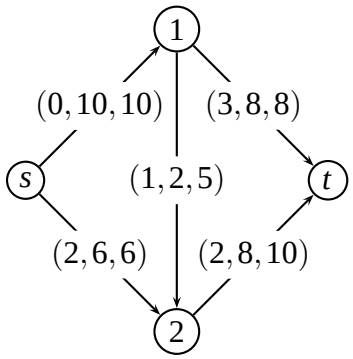


Figura 2.11: Red original con el flujo óptimo

En la siguiente iteración del algoritmo, el vértice s recibe la etiqueta permanente $(+s,\infty)$ y ningún vértice en $\Gamma^+(s)$ puede ser etiquetado pues $\xi_{ij} = q_{ij}$.

Por lo tanto, termina el algoritmo y se llega al corte formado por los arcos $(X_0 \to \tilde{X_0}) = \{(s,1),(s,2)\}$. Se cumple que el valor del corte mínimo es $v(X_0 \to \tilde{X_0}) = 16$ es igual al valor del flujo obtenido, por lo tanto el flujo máximo es de valor $v = 16$.

En el siguiente capítulo se profundizará sobre la tercer variante del problema mencionada anteriormente: flujo máximo en redes con ganancias.

Capítulo 3

Flujo máximo en redes con ganancias

3.1. Introducción

En las variantes del problema de flujo máximo revisadas hasta ahora, para todos los vértices distintos del origen y el terminal, se cumplen las ecuaciones de conservación de flujo.

Si en este contexto, suponemos que a cada arco de la red se le asocia un multiplicador positivo, entonces la cantidad de flujo que ingresa a un arco es distinta a la que sale del mismo, es decir, no se cumplen las ecuaciones de conservación del flujo, y en este caso, la tarea consiste en maximizar la cantidad de flujo que puede circular a través de la red sujeto a las distintas capacidades de cada arco y considerando el impacto de la ganancia en el flujo. A este problema se le denomina *flujo máximo en redes con ganancias* y el método para solucionarlo puede variar dependiendo de las variables y restricciones que se agreguen al planteamiento.

Diversos autores como Jean F. Maurras, William S. Jewell , Richard C. Grinold y Nicos Christofides[1], han desarrollado algoritmos para solucionar este tipo de problemas. Algunos de ellos utilizan como subrutina el método simplex para determinar soluciones factibles y para la eliminación de circuitos negativos, así como modificaciones del algoritmo de Ford y Fulkerson para maximizar el flujo a través de la red sujeto al costo por cada unidad de flujo con el que se esté trabajando.

Este problema tiene una amplia variedad de aplicaciones, por ejemplo: en trans-

[1]Las publicaciones correspondientes a estos autores se encuentran en la Bibliografía [6], [7], [8], [1]

porte, para maximizar reacciones químicas, para controlar la corriente que circula a través de estructuras eléctricas, para diversificar portafolios financieros, etc.

Como se mencionó en el capítulo anterior, en este capítulo se describirán los elementos y etapas que componen al algoritmo que presentó Nicos Christofides en 1975 para resolver el problema del flujo máximo en redes con ganancias.

3.2. Planteamiento del problema

De manera intuitiva, se desea maximizar la cantidad de flujo que circula sobre una red en la que cada arco tiene asociado un multiplicador que incrementa o disminuye la cantidad de flujo que pasa a través del mismo. Esta ganancia puede tomar cualquier valor, pero para estos efectos se considerará no negativa.

Siguiendo la notación del capítulo anterior, tenemos el siguiente planteamiento: Se desea pasar la mayor cantidad de flujo ξ_{ij} que circula en la red $R = [X, A, q, g]$, donde X es el conjunto de vértices de R, A es el conjunto de arcos, q_{ij} representa la capacidad del arco (x_i, x_j) y $g_{ij} > 0$ representa la ganancia del mismo.

Como el flujo que entra por un arco es diferente al flujo que sale del mismo, es necesario marcar esa distinción con la notación ξ^e_{ij} para el flujo entrante y ξ^o_{ij} para el flujo que sale luego de ser multiplicado por la ganancia, es decir, el flujo saliente por el arco x_{ij} es:

$$\xi^o_{ij} = \xi^e_{ij} * g_{ij} \tag{3.1}$$

Notemos que para este tipo de redes, la capacidad de cada arco aplica únicamente para el flujo entrante, así

$$\xi^e_{ij} \leq q_{ij} \tag{3.2}$$

Definimos al patrón de flujo Ξ como el conjunto de flujo entrante y flujo saliente en cada arco de la red $R = [X, A, q, g]$, es decir, $\Xi = \{(\xi^e_{ij}, \xi^o_{ij}) \forall (x_i, x_j) \in A$ y decimos que es factible si además cada vértice de R cumple con las siguientes condiciones de continuidad:

$$\sum_{x_j \in \Gamma^+(x_i)} \xi^e_{ij} - \sum_{x_k \in \Gamma^-(x_i)} \xi^o_{ki} = \begin{cases} v_s \text{ si } x_i = s \\ -v_t \text{ si } x_i = t \\ 0 \text{ si } x_i \neq s \text{ o t} \end{cases} \tag{3.3}$$

Las ecuaciones de continuidad del problema de flujo con ganancias son similares a las ecuaciones de conservación de flujo, con la diferencia de que se hace una distinción entre el flujo entrante y saliente de cada arco, ya que no necesariamente es el mismo.

Decimos que un patrón de flujo factible Ξ es **máximo** si produce el mayor valor de v_t y decimos que es **óptimo** si para cualquier flujo factible $\Xi' = (\xi'^{e}_{ij}, \xi'^{o}_{ij})$ ocurre sólo alguna de las siguientes:

i. $v_t \leq v'_t$ cuando $v_s = v'_s$

ii. $v_s \geq v'_s$ cuando $v_t = v'_t$

En otras palabras, un flujo es **óptimo** si el mismo flujo de entrada, produce el mayor valor de salida v_t. O si un menor valor de entrada produce el mismo valor de salida que un patrón de flujo diferente con v_s mayor.

3.2.1. Ejemplo

Veamos la diferencia entre un patrón de flujo no óptimo, óptimo y óptimo máximo. A cada arco de la siguiente red (Figura 3.1) se asocia (q_{ij}, g_{ij}).

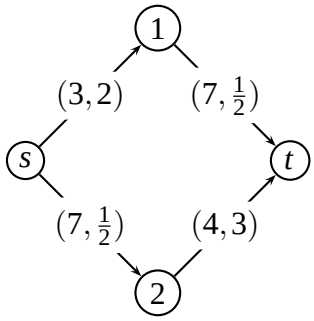


Figura 3.1: Red con (q_{ij}, g_{ij})

La siguiente red (Figura 3.2) muestra un patrón de flujo Ξ no óptimo. Notemos que $v_s = 3$ y $v_t = 4$.

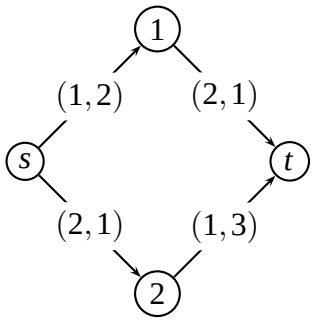


Figura 3.2: Patrón de flujo no óptimo

En la Figura 3.3 tenemos un patrón de flujo Ξ' óptimo y no máximo. Efectivamente, con el mismo valor de entrada, $v'_s = 3$, se produce un mejor valor de salida, $v'_t = 4.5$.

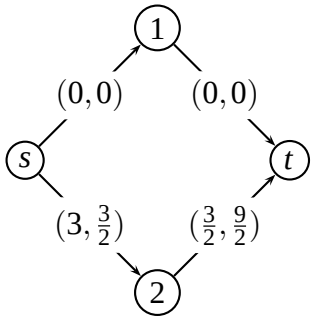


Figura 3.3: Patrón de flujo óptimo y no máximo

La Figura 3.4 muestra un patrón de flujo óptimo máximo con $v_s = 10$ y $v_t = \frac{27}{2}$

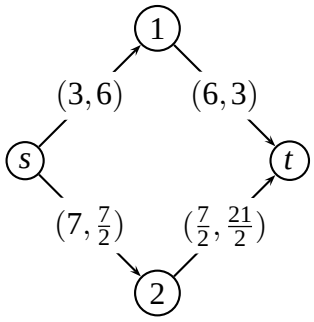


Figura 3.4: Patrón de flujo óptimo máximo

En este sentido, resolver el problema del flujo máximo en redes con ganancias, se refiere a encontrar el patrón de flujo Ξ óptimo máximo, es decir, un patrón de flujo que sea óptimo y además que produzca el valor de v_t máximo.

3.3. Cadenas Aumentantes

En este tipo de redes, se define una cadena aumentante del vértice x_1 al vértice x_t, como aquella en la que puede enviarse flujo a través de R del vértice x_1 al vértice x_t. Notemos que esta definición es análoga a la que se utiliza para el algoritmo de Ford y Fulkerson.

De esta forma, si tenemos la cadena aumentante $S = \{x_1, x_2, ..., x_t\}$, con los subconjuntos C^+ y C^- definidos en el capítulo anterior, entonces, para todos los arcos $(x_i, x_j) \in C^+$ se tiene que $\xi^e_{i,i+1} < q_{i,i+1}$ y para los arcos $(x_j, x_i) \in C^-$ se tiene que $\xi^e_{i-1,i} > 0$.

Calcularemos la ganancia de la cadena aumentante $(x_1, x_2, ..., x_t)$, denotada como $g(S)$ como:

$$g(S) = \prod_{(x_i,x_j)\in C^+} g_{ij} \prod_{(x_j,x_i)\in C^-} 1/g_{ji} \tag{3.4}$$

Capacidad de una cadena

La capacidad incremental de una cadena S es la máxima cantidad de flujo que puede circular a través de ella hasta que, algún $(x_i, x_j) \in C^+$ se sature o la capacidad incremental de uno de los arcos $(x_j, x_i) \in C^-$ se reduzca a cero.

Si S es una cadena simple sobre la cual se introduce un flujo δ empezando en x_{i1}, entonces el flujo entrante a x_{ip} en el arco (x_{ip}, x_{ip+1}) es:

$$\delta_{i_p,i_{p+1}} = \delta \prod_{(x_i,x_j)\in C_p^+} g_{ij} \prod_{(x_j,x_i)\in C_p^-} 1/g_{ij} = \delta \cdot g(S_p) \tag{3.5}$$

donde C_p^+ y C_p^- son subconjuntos de la subcadena $S_p = (x_{i1}, x_{i2}, ..., x_{ip})$ y $g(S_p)$ es la ganancia de S_p. Notemos que al hacer $\delta = 1$ en la expresión 3.5, se describe el comportamiento de una unidad de flujo al circular por la cadena S comenzando en el nodo x_{i1} hasta llegar al nodo x_{ip}. Por esta razón, para obtener la ganancia de S deben multiplicarse las ganancias de los arcos en C_p^+ y dividirse las ganancias de los arcos en C_p^-. Cuando $\delta \neq 1$ estamos calculando la ganancia final de introducir δ unidades de flujo en S. En este sentido, un arco $(x_{i_p}, x_{i_{p+1}}) \in C^+$ está

saturado cuando $\xi^e_{i_p i_{p+1}} + \delta_{i_p i_{p+1}} = q_{i_p i_{p+1}}$ y la capacidad incremental de los arcos $(x_{i_{p+1}}, x_{i_p}) \in C^-$ se reduce a cero cuando $\delta_{i_p i_{p+1}} = \xi^o_{i_{p+1},i_p} = g_{i_{p+1},i_p}\xi^e_{i_{p+1},i_p}$. Por lo tanto, la capacidad incremental, $q(S)$, de la cadena S es:

$$q(S) = \text{mín}\left[\min_{(x_{i_p},x_{i_{p+1}})\in C^+}\left\{\frac{q_{i_p,i_{p+1}} - \xi^e_{i_p,i_{p+1}}}{g(S_p)}\right\}, \min_{(x_{i_{p+1}},x_{i_p})\in C^-}\left\{\frac{g_{i_{p+1},i_p}\xi^e_{i_{p+1},i_p}}{g(S_p)}\right\}\right] \tag{3.6}$$

Tomemos la cadena de la Figura 3.5 como ejemplo. El primer número asociado a cada arco es su capacidad y el segundo es la ganancia.

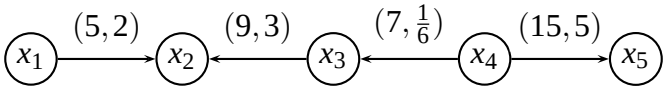


Figura 3.5: Ejemplo de cadena aumentante

En la Figura 3.6 se muestra el patrón de flujo $(\xi^e_{i,j}, \xi^o_{i,j})$ asociado a la cadena aumentante.

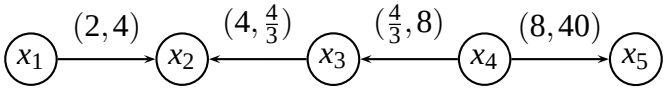


Figura 3.6: Patrón de flujo asociado a la Figura 3.5

Primero calculamos la ganancia en cada subcadena al introducir δ a través de x_1, como:

$$g(S_1) = \delta$$

$$g(S_2) = g(x_1)g_{12} = 2\delta$$

$$g(S_3) = \frac{g(x_2)}{g_{32}} = \frac{2\delta}{3} = \frac{2}{3}\delta$$

$$g(S_4) = \frac{g(x_3)}{g_{43}} = \frac{\frac{2\delta}{3}}{\frac{1}{6}} = \frac{12}{3}\delta = 4\delta$$

$$g(S_5) = g(x_4)g_{45} = 4\delta(5) = 20\delta$$

Ahora calculamos $\delta_{i,j}$ para cada arco de la cadena utilizando la Fórmula 3.5:

$$\delta_{1,2} = \frac{q_{1,2} - \xi^e_{1,2}}{g(S_1)} = \frac{5-2}{1} = 3$$

$$\delta_{3,2} = \frac{g_{3,2}(\xi^e_{3,2})}{g(S_2)} = \frac{3(4/3)}{2} = 2$$

$$\delta_{4,3} = \frac{\frac{1}{6}(8)}{\frac{2}{3}} = \frac{\frac{4}{3}}{\frac{2}{3}} = 2$$

$$\delta_{4,5} = \frac{15-8}{4} = \frac{7}{4}$$

Por la ecuación 3.6, tenemos: $q(S) = \text{mín}\left[\text{mín}\left\{\delta_{1,2}, \delta_{4,5}\right\}, \text{mín}\left\{\delta_{3,2}, \delta_{4,3}\right\}\right] = \text{mín}\left\{\frac{7}{4}, 2\right\} = \frac{7}{4}$ es igual al máximo valor de δ que puede ingresar a la cadena. Una vez actualizado el flujo, se obtiene el patrón de la Figura 3.7:

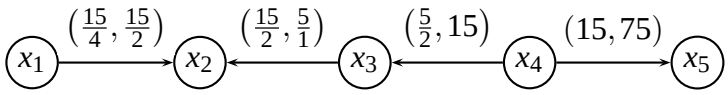


Figura 3.7: Ejemplo 3(b) Flujo actualizado

Como $\xi_{45} = q_{45} = 15$, se dice que el acro $(4,5)$ está saturado y ya no es posible transmitir flujo a través de él. En particular, no es posible transmitir flujo de x_1 a x_5, por lo tanto, la ganancia final de la cadena aumentante es $g = \xi^o_{4,5} = 75$.

3.4. Ciclos activos

Como un ciclo es una cadena cuyo vértice inicial y final es el mismo, podemos obtener su ganancia y capacidad siguiendo el mismo procedimiento que para la cadena. Decimos que un ciclo Φ es activo respecto al vértice final t si:

(i) Su ganancia es mayor que 1.

(ii) Su capacidad incremental $q(\Phi)$ es distinta de cero.

(iii) Existe algún vértice $x_i \in \Phi$ tal que existe una cadena aumentante de x_i a t. Además, si t forma parte de Φ, no es necesario buscar una cadena aumentante.

Del inciso (i) de la definición anterior se entiende que al circular flujo a través del ciclo activo, se "crea" nuevo flujo y por (iii), este nuevo flujo puede ser transmitido hasta t. Por lo tanto, un ciclo activo es aumentante, ya que es posible circular flujo a través de él y transmitirlo al nodo final t.

Decimos que un ciclo activo se ha desactivado cuando su capacidad incremental es igual a cero o cuando la capacidad incremental de las cadenas aumentantes

de $x_i \in \Phi$ a t es igual a cero, es decir, cuando no hay cadenas aumentantes en Φ de x_i a t. En otras palabras, un ciclo activo deja de serlo cuando deja de ser aumentante, pues ya no es posible incrementar el flujo a través de él ni llevarlo hasta t.

Los siguientes teoremas describen las propiedades de un patrón de flujo óptimo y uno óptimo máximo. Su demostración formal se encuentra en la referencia [9].

- *Teorema:* Un patrón de flujo Ξ es óptimo si y sólo si no exiten ciclos activos de s con respecto a t.

  Supongamos que v_t es el flujo resultante del Ξ. Notemos que si existe un ciclo activo en Ξ, se genera nuevo flujo de salida v'_t mayor a v_t para un mismo v_s. Por lo tanto Ξ no es óptimo.

- *Teorema:* Sea Ξ un patrón de flujo óptimo, si se incrementa el flujo a través de la cadena de s a t con la mayor ganancia, el flujo resultante también es óptimo.

  Supongamos que q es la cadena aumentante de s a t de mayor ganancia en Ξ. Sin perder factibilidad, incrementamos flujo a través de ella y obtenemos el patrón de flujo factible $\Xi' > \Xi$ en el que $v'_s > v_s$ y $v'_t > v_t$. Por lo tanto, Ξ' es óptimo.

- *Teorema:* El patrón de flujo Ξ es máximo y óptimo si la cortadura $(X_o \to X)$ que separa a s de t está saturada y no hay ciclos activos respecto a t y respecto a los vértices de $(X_o \to X)$.

  Por el primer teorema, si el patrón de flujo Ξ no tiene ciclos activos, entonces es óptimo. Por otro lado, como la cortadura que separa a s de t está saturada, ya no es posible incrementar el flujo de s a t. Entonces decimos que Ξ produce el mayor valor de v_t, es decir, que es máximo. Por lo tanto Ξ es máximo y óptimo.

El objetivo del siguiente algoritmo es obtener un patrón de flujo Ξ óptimo máximo. Para lograrlo se apoya de los teoremas anteriores de la siguiente manera:

- Primero, se busca obtener un patrón de que no contenga ciclos activos de forma que no sea posible ”generar”flujo sobre la red.

- Luego, se incrementa el flujo a través de la cadena aumentante de s a t de mayor ganancia. Por los dos primeros teoremas, se tendría un patrón de flujo óptimo.

- Si esto se repite hasta que no sea posible incrementar flujo, entonces la cortadura que separa a s de t estará saturada y como Ξ ya no tiene ciclos activos, utilizando el tercer teorema, Ξ es óptimo máximo.

3.5. Algoritmo

A continuación se describe el algoritmo para calcular el flujo máximo en redes con ganancias que presentó Nicos Christofides en su libro *Graph Theory. An Algorithmic Approach*.

De manera breve, este algoritmo se divide en dos etapas. La primera consiste en establecer un flujo inicial v_s, localizar ciclos activos y cadenas aumentantes que unen algún vértice del ciclo con el nodo final de la red y, posteriormente, desactivarlos. Para ello, se debe saturar o reducir a cero el flujo de alguno de los arcos que componen al ciclo o a la cadena aumentante. Esta etapa concluye cuando todos los ciclos han sido desactivados. En la segunda etapa se maximiza el flujo resultante de la primera etapa hasta lograr el valor del flujo de salida v_t deseado o alcanzar el valor máximo óptimo.

Para desactivar los ciclos, se circula una cantidad δ a través del ciclo activo y de una cadena aumentante que va de algún vértice del ciclo al vértice final de la red, de tal forma que se sature o se reduzca a cero la capacidad incremental de alguno de los arcos considerados.

En la siguiente sección se enumeran los pasos del algoritmo del flujo máximo en redes con ganancias. Notemos que los primeros cuatro pasos corresponden a la primera etapa y los restantes a la segunda. Después del algoritmo se justifican algunas sugerencias para realizar los cálculos en el menor número de iteraciones posibles.

3.5.1. Algoritmo

1. Iniciamos con cualquier flujo factible sobre la red $R = [X, A, q, g]$. Puede utilizarse un flujo de cero a través de cada arco.

2. Encontrar un ciclo activo $\Phi \in R$ respecto a t.

3. Sea el vértice $x_i \in \Phi$ tal que existe una cadena aumentante de x_i a t, nótese que x_i puede ser t. Empezando en x_i, sin perder la factibilidad del flujo, circular una cantidad de flujo δ a través del ciclo activo y llevar el flujo

excedente de x_i a t. Es importante escoger un δ tal que la capacidad incremental del ciclo activo, $q(\Phi)$ o de la cadena aumentante, $q(S)$, se reduzca a cero.

4. Actualizar el flujo Ξ y repetir los pasos 2 y 3 hasta desactivar todos los ciclos de la red, en este caso, pasar al paso 5. (En esta etapa Ξ es el flujo óptimo con $v_s = 0$).

5. Encontrar la cadena aumentante de s a t con la mayor ganancia. Mandar flujo a través de esta cadena hasta que su capacidad incremental se reduzca a cero.

6. Actualizar Ξ y repetir 5 hasta alcanzar el valor óptimo requerido del flujo de salida v_t, o hasta que no puedan encontrarse cadenas aumentantes de s a t, en este caso, el flujo encontrado será el óptimo máximo.

Para conservar la factibilidad en cada paso del algoritmo, deben cumplirse las ecuaciones de continuidad de flujo en todo momento. Ya que garantizan que la suma del flujo entrante de cada arco (x_i, x_j) es igual a la suma del flujo saliente de todos los arcos cuyo vértice final es x_i.

Para los pasos 2 y 5 pueden utilizarse algoritmos de rutas más cortas. Para este efecto, definimos la red incremental $R' = [X', A', q', c']$, donde $X' = X$ y A' es tal que

$$(x_i, x_j) \in A' \text{ si } 0 \leq \xi^e_{ij} < q_{ij}$$

con costo incremental

$$c'_{ij} = -log(g_{ij})$$

y capacidad

$$q'_{ij} = q_{ij} - \xi^e_{ij}$$

Y decimos que los arcos

$$(x_j, x_i) \in A' \text{ si } 0 < \xi^e_{ij} \leq q_{ij}$$

con costo incremental

$$c'_{ji} = -log(\tfrac{1}{g_{ij}})$$

y capacidad

$$q'_{ji} = \xi^e_{ij} g_{ij}$$

Así, un circuito en R' corresponde a un ciclo en R. Cuando dicho circuito tiene costo negativo, en la red original R obtenemos un ciclo Φ con $g(\Phi) > 1$ a través del cual es posible incrementar el flujo, es decir, obtenemos un ciclo activo. Esto puede verse al tomar logaritmos en ambos lados de la Ecuación 3.4 aplicada al ciclo activo Φ:

$$\begin{aligned}
\log(g(\Phi)) &= \log\left(\prod_{(x_i,x_j)\in C^+} g_{ij} \prod_{(x_j,x_i)\in C^-} 1/g_{ji}\right) \\
&= \sum_{(x_i,x_j)\in C^+} \log\left(g_{ij}\right) + \sum_{(x_j,x_i)\in C^-} \log\left(1/g_{ji}\right) \\
&= \sum_{(x_i,x_j)\in C^+} -\left(c'_{ij}\right) + \sum_{(x_j,x_i)\in C^-} -\left(c'_{ji}\right) \\
&= -\left(\sum_{(x_i,x_j)\in C^+} c'_{ij} + \sum_{(x_j,x_i)\in C^-} c'_{ij}\right)
\end{aligned}$$

De esta forma, si $g(\Phi) > 1$ entonces $\log(g(\Phi)) > 0$, lo cual implica que el costo del ciclo en R' debe ser negativo. Por el lado contrario, si $g(\Phi) \leq 1$, entonces el costo del ciclo $\Phi \in R$ es no negativo. En este caso, Φ no es un ciclo activo y por lo tanto, no es aumentante. Esto último se debe a que alguno de sus arcos $(x_i,x_j) \in C^+$ se encuentra saturado o alguno de sus arcos $(x_i,x_j) \in C^-$ tiene flujo inicial $\xi_{ij} = 0$.[2]

Notemos que para resolver este tipo de problemas resulta conveniente utilizar el algoritmo de Floyd, pues es una herramienta sencilla que sirve para encontrar el camino más corto entre todo par de vértices y para la detección de circuitos negativos.

En línea con el párrafo anterior, podemos utilizar el algoritmo de Floyd para localizar ciclos activos del paso 2, tomando como variable de distancia al costo incremental que cada arco tiene asociado. De esta forma, se estarían localizando de manera eficiente los circuitos de costo negativo.

La red con la que se trabaja en el paso 5, ya no tiene circuitos de costo negativo, pues fueron desactivados en el segundo paso. Nuevamente se utiliza R', ahora considerando las capacidades, para encontrar el camino más corto de s a t y el resultado equivale a la cadena aumentante con mayor ganancia en R.

[2] Para más información, se recomienda consultar [1]

Cabe destacar que, en el paso 3, si al transmitir flujo a través de un ciclo Φ, éste no se desactiva pero se reduce a cero la capacidad incremental de la cadena aumentante, en la siguiente iteración del paso 2, deberá tomarse el mismo ciclo activo Φ y otra cadena aumentante que vaya de cualquier vértice en Φ a t. Este procedimiento se repite hasta que el ciclo quede desactivado y se localice un ciclo activo diferente.

Dada la utilidad de este algoritmo y su fácil implementación en diversas áreas de conocimiento, se realizó un programa en R que muestra el comportamiento del flujo sobre la red en cada iteración hasta alcanzar su máximo, siguiendo el algoritmo previamente mencionado.

3.6. Ejemplo

En esta sección se muestra un ejemplo de aplicación del algoritmo desarrollado anteriormente. Los resultados fueron obtenidos con el programa anexo.[3] En la Figura 3.8 se muestra la red inicial, el número asociado a cada arco es (q_{ij}, g_{ij}) y el valor del flujo actual es cero.

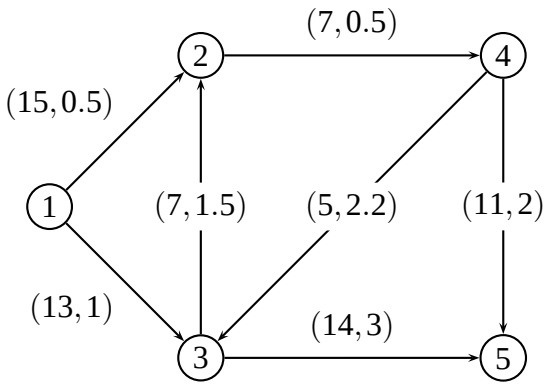


Figura 3.8: Red original

Iniciamos con un flujo factible $\Xi = 0$ en todos los arcos de la red. A continuación se construye una red auxiliar R', definida en la sección anterior, para localizar ciclos activos. A cada arco de la red de la Figura 3.9 se le asocia $c'_{ij} = -log(g_{ij})$.

[3] El código del programa se encuentra en el Apéndice B y los resultados de este ejemplo se muestran en el Apéndice A.

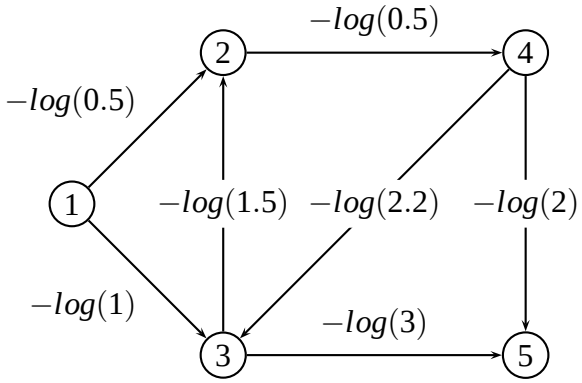


Figura 3.9: Red auxiliar

Como se explicó previamente, un ciclo activo en R, equivale a un circuito de costo negativo en R', por lo que se utilizará el algoritmo de Floyd revisado en el Capítulo 1 para encontrar un circuito de costo negativo en R'.

Las matrices iniciales asociadas a R' (Figura 3.9) son:

$$C = \begin{bmatrix} 0 & -log(0.5) & -log(1) & \infty & \infty \\ \infty & 0 & \infty & -log(0.5) & \infty \\ \infty & -log(1.5) & 0 & \infty & -log(3) \\ \infty & \infty & -log(2.2) & 0 & -log(2) \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

$$z = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

En la tercera iteración llegamos a la siguientes matrices:

$$z = \begin{bmatrix} 1 & 3 & 1 & 2 & 3 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 2 & 3 \\ 4 & 3 & 4 & 2 & 3 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & -0.90 & -0.50 & 0.28 & -1.59 \\ \infty & -0.50 & -0.09 & 0.69 & -1.19 \\ \infty & -0.90 & -0.50 & 0.28 & -1.59 \\ \infty & -1.19 & -0.78 & -0.50 & -1.88 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Como $z[4,4] \neq 4$ y además $C[2,2], C[3,3], C[4,4]$ son menores a cero, existe un circuito negativo en la red. Para recuperarlo, utilizaremos la matriz z de la siguiente forma: $z[4,4] = 2$, $z[4,2] = 3$ y $z[4,3] = 4$. Por lo tanto, el circuito negativo es $\phi = \{4,3,2,4\}$ y corresponde a un ciclo activo en la red original (Figura 3.8) con la cadena aumentante asociada $S = (4,5)$.

Primero calculamos la ganancia de cada subcadena del ciclo activo al circular una cantidad de flujo δ a través del mismo y transmitir el flujo excedente por la cadena aumentante , iniciando en el vértice $x_{i_1} = 4$:

$$g(S_4) = \delta$$

$$g(S_3) = g(S_4)g_{43} = 2.2\delta$$

$$g(S_2) = g(S_3)g_{32} = (2.2\delta)(1.5) = 3.3\delta$$

$$g(S_4) = g(S_2)g_{24} = (3.3\delta)(0.5) - \delta = 0.65\delta$$

$$g(S_5) = g(S_4)g_{45} = (0.65\delta)(2) = 1.3\delta$$

Al llegar al nodo inicial x_{i_1} después de haber transmitido flujo a través del ciclo activo, vuelve a calcularse la ganancia de la subcadena correspondiente, pero esta vez restando la cantidad de flujo δ que había ingresado inicialmente al ciclo. De esta forma estamos asegurando que el flujo excedente de enviar una unidad de flujo por el ciclo activo, es el que circula a través de la cadena aumentante.

Ahora calculamos $\delta_{i,j}$ para cada arco en ϕ y para los arcos en S utilizando la Fórmula 3.5:

$$\delta_{4,3} = \frac{5-0}{1} = 5$$

$$\delta_{3,2} = \frac{7-0}{2.2} = \frac{35}{11}$$

$$\delta_{2,4} = \frac{7-0}{3.3} = \frac{70}{33}$$

$$\delta_{4,5} = \frac{11-0}{0.65} = \frac{120}{13}$$

Y $\min \delta_{ij} = \min\{\delta_{4,3}, \delta_{3,2}, \delta_{2,4}, \delta_{4,5}\} = \delta_{2,4} = \frac{70}{33}$ por lo tanto se satura el arco $(2,4)$ y el ciclo ϕ queda desactivado. En la Figura 3.10 se muestra el flujo actualizado. A cada arco se asocia (ξ^e_{ij}, ξ^o_{ij}).

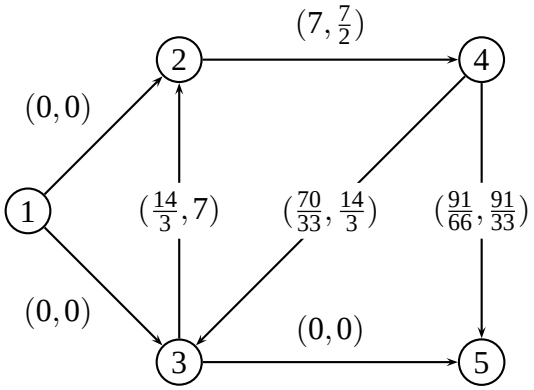


Figura 3.10: Flujo resultante de la primera iteración

Nuevamente se construye la red auxiliar (Figura 3.11) en busca de un circuito de costo negativo. Las líneas punteadas de R' representan a los arcos de A' que no forman parte de A.

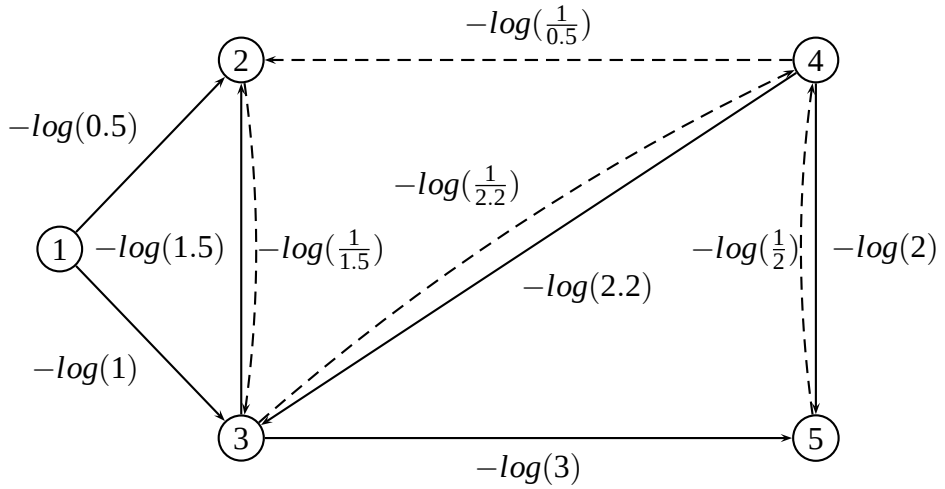


Figura 3.11: R' de la segunda iteración

Al aplicar el algoritmo de Floyd sobre la red de la Figura 3.11, en la cuarta iteración se llega a las matrices:

$$z = \begin{bmatrix} 1 & 3 & 4 & 3 & 3 \\ 2 & 3 & 4 & 3 & 3 \\ 3 & 3 & 4 & 3 & 3 \\ 4 & 3 & 4 & 3 & 3 \\ 5 & 3 & 4 & 5 & 3 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & -0.40 & -1.11x10^{-}16 & 0.78 & -1.09 \\ \infty & -2.22x10^{-}16 & 0.40 & 1.19 & -0.69 \\ \infty & -0.40 & -1.11x10^{-}16 & 0.78 & -1.09 \\ \infty & -1.19 & -0.78 & -1.11x10^{-}16 & -1.88 \\ \infty & -0.50 & -0.09 & 0.69 & -1.19 \end{bmatrix}$$

Al utilizar el último renglón de la matriz z se encuentra el circuito negativo, o ciclo activo, $\phi = \{5,4,3,5\}$, y como el vértice final $x_t = 5$ es parte del ciclo activo, no se requiere encontrar una cadena aumentante en esta iteración. A continuación se calcula la ganancia de cada subcadena de ϕ al circular sobre el mismo una cantidad incremental de flujo δ:

$$g(S_5) = \delta$$

$$g(S_4) = \frac{g(S_5)}{g_{54}} = \frac{\delta}{2} = 0.5\delta$$

$$g(S_3) = g(S_4)g_{43} = (0.5\delta)(2.2) = 1.1\delta$$

$$g(S_5) = g(S_3)g_{35} = (1.1\delta)(3) = 3.3\delta$$

Ahora calculamos $\delta_{i,j}$ para cada arco en ϕ utilizando la Fórmula 3.5:

$$\delta_{4,5} = \frac{91}{33}$$

$$\delta_{4,3} = \frac{5-\frac{70}{33}}{0.5} = \frac{190}{33}$$

$$\delta_{3,5} = \frac{14-0}{1.1} = \frac{140}{11}$$

Seleccionamos $\text{mín}\,\delta_{ij} = \text{mín}\left\{\delta_{4,5}, \delta_{4,3}, \delta_{3,5}\right\} = \delta_{4,5} = \frac{91}{33}$ por lo tanto se reduce a cero el flujo en $(4,5)$ y como la capacidad incremental del ciclo ϕ es igual a cero, el ciclo queda desactivado. La Figura 3.12 muestra el flujo actualizado correspondiente. A cada arco se asocia (ξ^e_{ij}, ξ^o_{ij}).

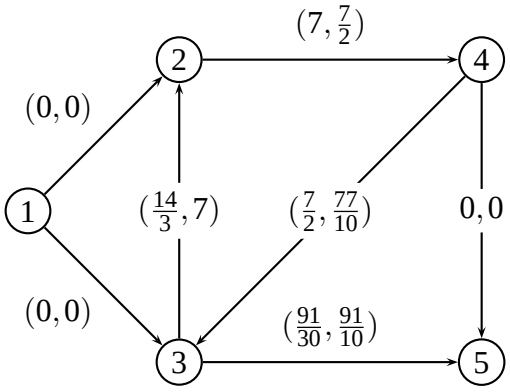


Figura 3.12: Flujo resultante de la segunda iteración

Al aplicar el algoritmo de Floyd sobre la red auxiliar R' asociada a la red anterior, no se detectaron circuitos negativos. Por lo tanto, ahora deben localizarse cadenas aumentantes del vértice inicial 1 al vértice final 5.

Para llevar acabo esto, se utilizan algoritmos de rutas más cortas sobre la red incremental R'. Por comodidad, se utilizará nuevamente el algoritmo de Floyd. La red auxiliar correspondiente se muestra en la Figura 3.13. A cada arco se asocia q_{ij}

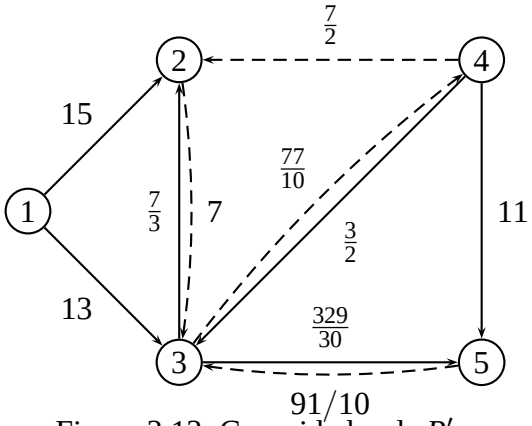


Figura 3.13: Capacidades de R'

En primer lugar, se encuentra la cadena $S = \{1,3,5\}$. La ganancia de cada subcadena de S al introducir una cantidad incremental δ de flujo sobre la misma es: $g(S_1) = \delta$, $g(S_3) = \delta$ y $g(S_5) = 3\delta$, y con capacidad incremental δ_{ij} de cada

arco $\delta_{13} = 13$ y $\delta_{35} = \frac{329}{30}$.

Por lo tanto, la capacidad incremental de la cadena aumentante es $q(S) = \min\{\delta_{13}, \delta_{35}\} = \delta_{35} = \frac{329}{30}$. La actualización del flujo se muestra en la red de la Figura 3.14, donde el número asociado a cada arco representa el patrón de flujo entrante y saliente, (ξ^e_{ij}, ξ^o_{ij}).

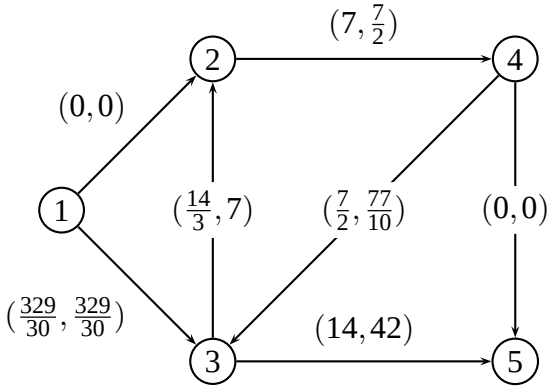


Figura 3.14: Primera actualización del flujo

Una vez construida la red auxiliar, mostrada a continuación, se buscan nuevamente cadenas aumentantes del vértice 1 al vértice 5. El número asociado a cada arco de R' (Figura 3.15) representa su capacidad, q'_{ij}.

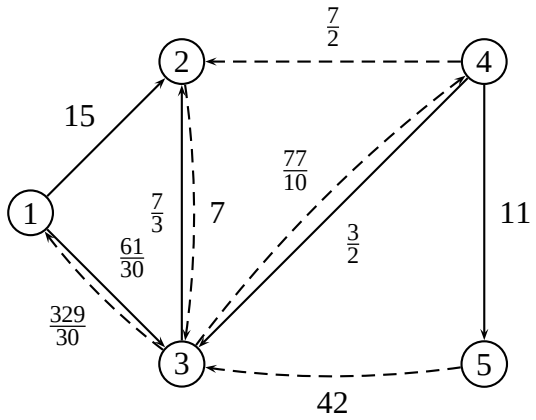


Figura 3.15: Red auxiliar

Esta vez, el algoritmo de Floyd nos da la cadena $S = \{1,3,4,5\}$. La ganancia de cada subcadena de S al introducir una cantidad de flujo δ sobre la misma es: $g(S_1) = \delta$, $g(S_3) = \delta$, $g(S_4) = \frac{11}{5}\delta$ y $g(S_5) = \frac{22}{5}$. Con capacidad incremental de cada arco δ_{ij} igual a:

$\delta_{13} = \frac{61}{30}$

$\delta_{34} = \frac{77}{10}$

$\delta_{35} = 5$

Entonces la capacidad incremental de la cadena es $q(S) = \text{mín}\{\delta_{13}, \delta_{34}, \delta_{45}\} = \delta_{13} = \frac{61}{30}$, notemos que el arco $(1,3)$ queda saturado. La actualización del flujo se encuentra en la Figura 3.16.

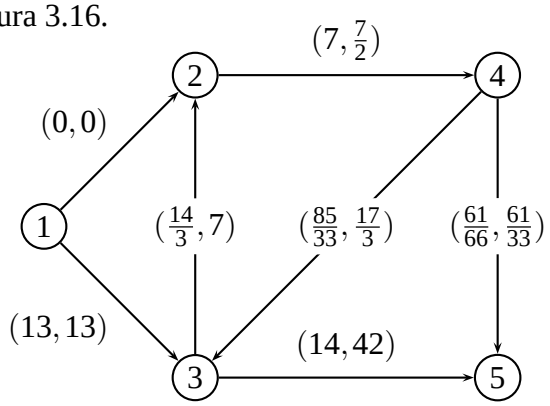


Figura 3.16: Segunda actualización del flujo

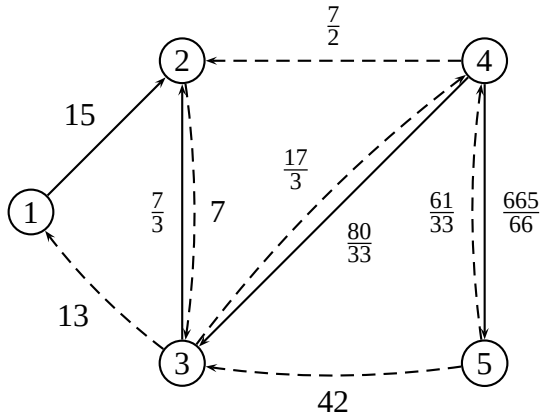


Figura 3.17: Red auxiliar de la Figura 3.12

Al aplicar el algoritmo de Floyd sobre la red de la Figura 3.17, se encuentra la cadena aumentante $S = \{1,2,3,4,5\}$. La ganancia de cada subcadena de S al introducir una cantidad adicional de flujo δ comenzando en $x_{i_1} = 1$ es: $g(S_1) = \delta$, $g(S_2) = 0.5\delta$, $g(S_3) = \frac{1}{3}\delta$, $g(S_4) = \frac{5}{33}\delta$ y $g(S_5) = \frac{10}{33}\delta$.

Ahora se calcula la capacidad incremental de cada arco: $\delta_{12} = 15$, $\delta_{23} = 14$, $\delta_{34} = 17$ y $\delta_{45} = 66.5$. Por lo tanto la capacidad incremental de la cadena aumentante es $q(S) = 14$ y el flujo a través del arco $(3,2)$ queda reducido a cero. La Figura 3.18 ilustra la actualización del flujo.

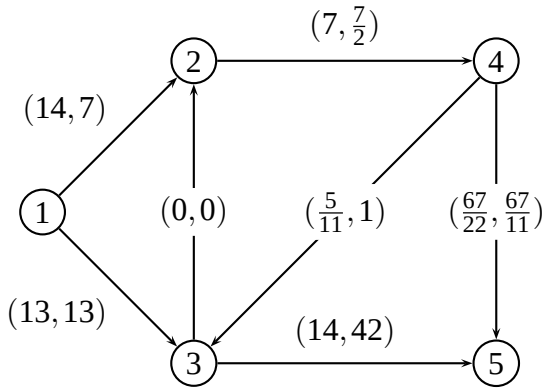


Figura 3.18: Flujo máximo óptimo

Como en R' ya no existen cadenas aumentantes de 1 a 5, el flujo actual es el máximo óptimo y la ganancia final de la red es $\frac{529}{11} \approx 48.091$.

Capítulo 4

Manual del usuario

A continuación se describen las variables, funciones y estructura del programa diseñado en R para resolver el problema del flujo máximo en redes con ganancias.

4.1. Variables

- vertices: Es el número de vértices de la red.
- Redq: Matriz de *nxn*, donde **n** es el número de vértices de la red. El número asociado la coordenada (i, j) es la capacidad del arco $(i, j) \in A$, es decir, q_{ij}.
- Redg: Matriz de dimensiones *nxn*, donde *n* es el número de vértices de la red. El número asociado la coordenada (i, j) representa la ganancia $g_{ij} > 0$ del arco $(i, j) \in A$.
- FEntrante: Matriz de *nxn*, donde *n* es el número de vértices de la red, que representa el patrón de flujo entrante de cada arco (i, j) de la red $R = [X, A, q, g]$.
- FSaliente: Matriz de dimensiones análogas a la anterior que representa el patrón de flujo saliente de la red $R = [X, A, q, g]$.
- RedAuxCostos: Matriz que servirá para calcular la red auxiliar de costos a partir de la matriz de ganancias, siguiendo las condiciones que marca el algoritmo.
- C,z,k: Matrices del algoritmo de Floyd, cuya función es la misma que en algoritmo definido en el Capítulo 1, el cual sirve como subrutina principal del algoritmo del flujo máximo en redes con ganancias.

- Circuito: Matriz que sirve para guardar el circuito negativo encontrado en el algoritmo de Floyd. Utiliza como auxiliares a las variables *c1,c2,c3,auxiliar*.
- Cadena: Es una matriz de $(n+1)x1$ en la que se calcula el camino más corto del inicio del ciclo activo al vértice final de la red.
- CicloCadena: Matriz de dimensión $(n+1)x1$ en la que se concatena el ciclo activo con la cadena aumentante.
- gnodo: Matriz auxiliar de dimensiones $(n+1)x3$ en la que se calcula la ganancia de cada subcadena tomando como base CicloCadena y la capacidad incremental de cada arco.
- delta: Variable numérica que representa el mínimo de las capacidades incrementales de cada arco.
- RedAuxCapacidades: Matriz de *nxn* que sirve para calcular la capacidad incremental de cada arco de la red en la segunda etapa del algoritmo.
- CadenaAumentante: Matriz de $(n+1)x1$ en donde se recupera en orden inverso el camino más corto del vértice inicial al final que se calculó con el algoritmo de Floyd. Se utiliza en la segunda parte del algoritmo.
- gnodo2: Esta matriz es análoga a gnodo, pero se utiliza cuando la red ya no tiene ciclos activos y sólo buscamos maximizar el flujo a través de ella.
- g: Variable numérica donde se guarda el valor de la ganancia final de la red.
- iteracion: Variable numérica que sirve como contador y auxiliar para algunas funciones.
- Parte: Variable numérica que sirve para indicar la etapa del algoritmo que se está calculando.

Las variables no mencionadas en esta lista fueron utilizadas como contadores o auxiliares en las funciones que componen el código.

4.2. Funciones

- AuxiliarCostos: Esta función sirve para construir la red auxiliar de costos que utilizaremos para determinar ciclos activos sobre la red original.
- CFloyd: Función complementaria del algoritmo de Floyd para calcular la matriz C de dimensiones *nxn* definida como en el Capítulo 1.

- zFloyd: Función complementaria del algoritmo de Floyd para calcular la matriz *z* de dimensiones *nxn* definida como en el Capítulo 1.

- Floyd: Función que realiza los cálculos del algoritmo de Floyd para calcular el camino más corto entre todo par de vértices de una red.

- RecuperaCiclo: Sirve para localizar los circuitos negativos detectados en el algoritmo de Floyd.

- RecuperaCA: Una vez localizado el circuito negativo, esta función encuentra una cadena aumentante de algún vértice del ciclo al vértice final de la red.

- Validar: Esta función valida que la cadena aumentante no utilice más de un nodo del ciclo activo y no contenga vértices repetidos.

- Concatenar: Función que concatena el circuito negativo, que en este caso es un ciclo activo, con la cadena aumentante calculados en las funciones RecuperaCiclo y RecuperaCA.

- FunGnodo: Esta función calcula la ganancia de cada subcadena del camino que seguiría el flujo a través del ciclo activo y la cadena aumentante y da estimaciones de delta para cada vértice.

- CalculaDelta: Esta función sirve como auxiliar para separar los cálculos de delta propuestos en la función FunGnodo y seleccionar el mínimo valor calculado. [1]

- Resumen: Esta función nos muestra el resultado de cada iteración.

- RedCapacidades: Esta función nos ayuda a construir la red auxiliar de capacidades para la etapa del algoritmo en la que deben encontrarse cadenas aumentantes del inicio al final de la red para maximizar el flujo circulante a través de la misma.

- RecuperaCA2: Esta función nos ayuda a recuperar la cadena aumentante del vértice inicial al final, siguiendo la ruta de la matriz *z* resultante del algoritmo de Floyd.

[1] En el algoritmo de flujo máximo en redes con ganancias, delta es la cantidad de flujo que circulará a través del ciclo activo y la cadena aumentante y cumple con las características de ser el máximo valor posible para saturar o reducir a cero la capacidad incremental de algún arco de la red.

- FunGnodo2: Función análoga a FunGnodo, se utiliza para calcular la ganancia de cada subcadena de la cadena aumentante resultante de la función RecuperaCA2.
- ganancia: Calcula la ganancia final de la red una vez que el flujo ha sido maximizado.

4.3. Estructura del código

El programa sigue la misma estructura del algoritmo de flujo máximo en redes con ganancias. Es por eso que el programa se encuentra dividido en dos etapas:

Etapa 1. Localizar y desactivar los ciclos activos de la red.

Etapa 2. Maximizar el flujo a través de la red resultante de la primera etapa.

En el código, la primer etapa está activa cuando Parte1=1, y la segunda etapa se activa cuando Parte1=0. El siguiente pseudocódigo muestra a grandes rasgos el procedimiento que sigue el programa para resolver el problema del flujo máximo con ganancias:

```
iteracion=1
While (iteracion<=n)
{
if (Parte1=1)
{
        Construir (Red_Auxiliar_de_Costos)
        Floyd(Red_Auxiliar_de_Costos)
        Localiza_Circuito_Negativo()
        Localiza_Cadena_Aumentante()
        CalculaGanancias()
        delta= min(g1,g2,...)
        if(is.na(delta)==TRUE || delta<=0)
                Parte1<-0
}
If(Parte1=0)
{
        Construir(Red_Auxiliar_Capacidades)
        Floyd(Red_Auxiliar_Capacidades)
Localiza_Cadena_Aumentante()
```

```
        if(Cadena_Aumentante != Error)
        {
CalculaGanancias_CA()
                delta=min(g1,g2,...)
        }
}
Actualiza_Flujo()

If(Parte1= 0 && Cadena_Aumentante== Error)
        ganancia_final()

iteracion=iteracion+1

}
```

4.4. Instrucciones de uso

Las únicas variables que el usuario debe definir para utilizar el programa son:

- $vertices < -n$, donde n es el número de vértices de la red.
- $Redq < -matrix(ncol = vertices, nrow = vertices)$
- $Redg < -matrix(ncol = vertices, nrow = vertices)$

Es necesario que el usuario considere los dominios de dichas variables, mencionados en el Capítulo 3, por ejemplo que $g_{ij} > 0 \forall (i, j)$. Entonces, si consideramos un arco que une a los vértices (1) y (2) en una red del tipo $R = [X, A, q, g]$, con capacidad 3 y ganancia 2, dicho arco se definiría como:

$Redq[1,2] < -3$
$Redg[1,2] < -2$

El usuario debe considerar que en las matrices que representan redes, tales como $FEntrante$ y $FSaliente$ las entradas $[i, j] = NA$ representan a los arcos (x_i, x_j) que no forman parte de la red. En la matriz *gnodo*, las entradas marcadas con *NA*, son simplemente valores vacíos y no se tomarán en consideración en la interpretación del resultado.

Para aprender a interpretar los resultados, tomaremos como base el resultado de la primer iteración del algoritmo, al ejecutarlo sobre la red del ejemplo de aplicación

del Capítulo 3.

```
———————— Iteracion= 1 ————————————

 gnodo=
     [,1]     [,2]     [,3]     [,4] [,5] [,6] [,7] [,8]
[1,]    4 3.000000 2.000000  4.00000  5.0   NA   NA   NA
[2,]    1 2.200000 3.300000  0.65000  1.3   NA   NA   NA
[3,]    5 3.181818 2.121212 16.92308   NA   NA   NA   NA

 delta= 2.121212

 FEntrante=
     [,1]     [,2]     [,3] [,4]     [,5]
[1,]   NA 0.000000 0.000000   NA       NA
[2,]   NA       NA       NA    7       NA
[3,]   NA 4.666667       NA   NA 0.000000
[4,]   NA       NA 2.121212   NA 1.378788
[5,]   NA       NA       NA   NA       NA

 FSaliente=
     [,1] [,2]     [,3] [,4]     [,5]
[1,]   NA    0 0.000000   NA       NA
[2,]   NA   NA       NA  3.5       NA
[3,]   NA    7       NA   NA 0.000000
[4,]   NA   NA 4.666667   NA 2.757576
[5,]   NA   NA       NA   NA       NA

 La ganancia final de la red R es: 2.757576
```

Primero se muestra la matriz **gnodo**. [2] Para lograr un mejor entendimiento, se explicará cada renglón de esta matriz de manera individual.

El primer renglón nos muestra los vértices del ciclo activo y de la cadena aumentante que fueron calculados con el algoritmo de Floyd. En este caso, se encontró el ciclo activo $\{4,3,2,4\}$ con la cadena aumentante $(4,5)$. Al concatenarlos se obtiene la secuencia de vértices $\{4,3,2,4,5\}$, que se muestra a continuación:

```
     [,1]     [,2]     [,3]     [,4] [,5] [,6] [,7] [,8]
```

[2] Por construcción del programa, la matriz gnodo cuenta con varias entradas vacías.

```
[1,]    4 3.000000 2.000000  4.00000  5.0   NA   NA   NA
```

El segundo renglón nos muestra el flujo de cada subcadena al circular una unidad delta de flujo a través del ciclo activo y transmitir el excedente a través de la cadena aumentante. Para dicho cálculo se considera el arco formado por la entrada *(gnodo[1,n],gnodo[1,n+1])*.

En el ejemplo, una unidad de flujo entra por el vértice 4, por lo que la primer entrada del segundo renglón es 1, y al pasar por el arco $(4,3)$, el flujo vale 2.2. A continuación se muestra el segundo renglón de la matriz *gnodo* correspondiente a la primer iteración.

```
[2,]    1 2.200000 3.300000  0.65000  1.3   NA   NA   NA
```

Por último, el tercer renglón de *gnodo* muestra la capacidad incremental de cada arco, es decir δ_{ij}. Por ejemplo, en esta iteración, la capacidad incremental del arco $(4,3)$ es 5, pues:

$\delta_{4,3} = \frac{4,3 - \xi^e_{4,3}}{g(4)} = \frac{5-0}{1} = 5$

El tercer renglón se muestra a continuación:

```
[3,]    5 3.181818 2.121212 16.92308   NA   NA   NA   NA
```

Como δ es el mínimo de las capacidades incrementales, la variable **delta** toma el valor más pequeño del tercer renglón de *gnodo*.

```
delta=  2.121212
```

Finalmente, las matrices **FEntrante** y **FSaliente**, muestran el patrón de flujo entrante y saliente correspondientes a la actualización de flujo.

```
FEntrante=
     [,1]     [,2]     [,3] [,4]     [,5]
[1,]   NA 0.000000 0.000000   NA       NA
[2,]   NA       NA       NA    7       NA
[3,]   NA 4.666667       NA   NA 0.000000
[4,]   NA       NA 2.121212   NA 1.378788
[5,]   NA       NA       NA   NA       NA

FSaliente=
```

```
      [,1] [,2]     [,3] [,4]     [,5]
[1,]   NA    0 0.000000   NA       NA
[2,]   NA   NA       NA  3.5       NA
[3,]   NA    7       NA   NA 0.000000
[4,]   NA   NA 4.666667   NA 2.757576
[5,]   NA   NA       NA   NA       NA
```

Y visto sobre la red original, la Figura 4.1 muestra el patrón de flujo actualizado.

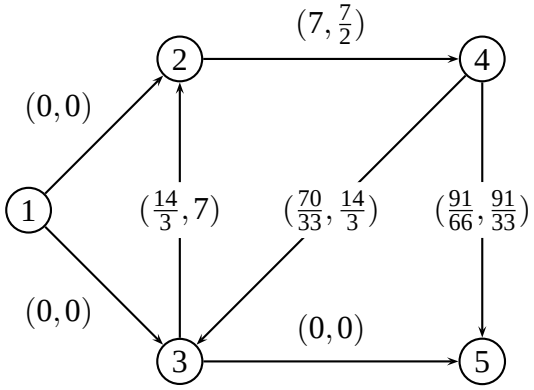


Figura 4.1: Flujo resultante de la primer iteración

Al finalizar la ejecución del programa, se obtiene la red de la Figura 4.2 con un flujo máximo óptimo de 48.091.

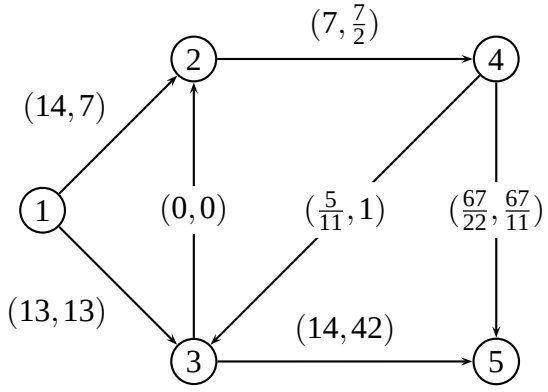


Figura 4.2: Flujo máximo óptimo

Conclusiones

Existe una gran variedad de algoritmos para solucionar problemas de optimización basados en la representación gráfica de los mismos mediante el uso de redes. Entre los más comunes se encuentran el de rutas más cortas y el de flujo máximo. Como su nombre lo indica, en el problema de rutas más cortas el objetivo es encontrar un camino tal que la distancia recorrida para llegar de un punto inicial a uno final, sea la mínima posible. Por otro lado, en el problema de flujo se busca que al final de un proceso se haya logrado obtener la mayor cantidad de alguna materia o producto.

Al estudiar las variantes más comunes del problema de flujo, surgió la pregunta ¿qué sucede cuando en un proceso podemos tener ganancias o pérdidas antes de llegar al resultado final? En este caso, el problema de maximizar la ganancia final se torna más complejo. Por ejemplo, supongamos que se desea maximizar la ganancia de un portafolio de inversión al final de N periodos, dados diversos instrumentos de inversión y la posibilidad de otorgar créditos. Este problema puede plantearse sobre una red de la siguiente manera: los distintos periodos se representan con nodos, los posibles movimientos de efectivo con arcos, las tasas de interés y rendimiento con ganancias y el límite máximo de efectivo en cada operación es la capacidad de cada arco. A este problema se le llama flujo máximo en redes con ganancias y es el tema principal de esta tesis.

En el primer capítulo se mostraron algunos de los métodos más utilizados para resolver el problema de rutas más cortas haciendo énfasis en el método propuesto por Floyd, el cual ayuda a la detección de circuitos negativos al mismo tiempo que calcula el camino más corto entre todo par de vértices de la red. De manera particular, este algoritmo fue utilizado como subrutina para el cálculo del flujo máximo en redes con ganancias del que trata esta tesis.

En el segundo capítulo se incluyó introducción al problema del flujo y algunas de sus variantes, por ejemplo, cuando además de restricciones de capacidad, cada arco tiene una cota inferior. Este capítulo constituyó la base para poder estudiar la

variante del problema de flujo que se expone en el capítulo 3.

Entre los métodos de solución del problema del flujo en redes con ganacias, se encontró el propuesto por Jean Francois Maurras en su artículo *Optimization of the flow through networks with gains*, en el que resuelve el problema mediante una modificación a los cálculos del algoritmo simplex dual. Otro método fue propuesto por William S. Jewell en *Optimal flow through networks with gains* el cual encuentra la solución mediante la maximización del flujo del problema dual.

El objetivo principal de esta tesis es estudiar el problema del flujo en redes con ganancias utilizando el método presentado en 1975 por Nicos Christofides en el libro *Graph Theory. An Algorithmic Approach*. Por este motivo, el tercer capítulo se enfocó en dicho método. El algoritmo se divide en dos etapas: en la primera se busca establecer un flujo sobre la red que no contenga ciclos activos, es decir, ciclos por los que pueda circularse flujo de manera continua obteniendo cada vez una ganancia mayor. Para lograr lo anterior se utiliza un algoritmo de rutas más cortas que permita la detección de circuitos negativos sobre una red auxiliar de costos. En la segunda etapa se maximiza el flujo resultante.

En términos de tiempos de ejecución, podría resultar costoso realizar los cálculos que requieren los algoritmos propuestos por Jean F. Maurras y William S. Jewell para el problema de redes con ganancias, es por eso que se pensó que sería más eficiente resolver este problema con el algoritmo de Nicos Christofides, ya que utiliza como subrutina principal uno de rutas más cortas. Sin embargo, al utilizar este método se tiene la desventaja de que la ganancia debe ser mayor a cero en cada arco de la red, por lo que su aplicación queda restringida a ciertos problemas que cumplan con esta condición.

En los cursos de Teoría de Redes a nivel licenciatura, se imparte una introducción al problema del flujo y algunas de sus variantes, por lo que otro objetivo de este trabajo es constituir una herramienta para el estudio de la variante en la que los arcos tienen ganancias mayores a cero. Además se realizó un código del algoritmo de Nicos Christofides en el lenguaje de programación R, que fue utilizado para comprobar los cálculos del ejemplo expuesto y se incluyó junto con un manual del usuario para que alumnos y profesores puedan implementarlo con facilidad.

Apéndice A

Resultados del ejemplo Cap. 2

Como se ha mencionado anteriormente, debido a que cada arco está sujeto a una ganancia, el flujo que entra por un arco es distinto del que sale, por lo tanto el patrón de flujo está dividido en dos partes: entrante y saliente, es decir, para cada arco $(i, j) \in A$, tenemos $\xi = (\xi^e_{ij}, \xi^o_{ij})$.

El ciclo activo resultante de la primer iteración del algoritmo fue $\{4,3,2,4\}$ con la cantidad incremental $\delta = \frac{70}{33}$. Al circular dicho valor por el ciclo activo se obtuvo el patrón de flujo de las matrices *FEntrante* y *FSaliente*.

Notemos que la coordenada (i, j) corresponde al arco $(i, j) \in X$. Y el valor asociado a cada coordenada es el valor del flujo correspondiente en cada arco. Los arcos que no existen en la red se muestran con *NA*. Cabe mencionar que el resultado que arroja el programa incluye también a la matriz *gnodo*, pero para facilitar la lectura, se incluyen únicamente el cálculo de delta y los patrones de flujo.[1]

```
———————————— Iteracion=  1 ————————————————

 gnodo=
     [,1]     [,2]     [,3]     [,4] [,5] [,6]
[1,]    4 3.000000 2.000000  4.00000  5.0   NA
[2,]    1 2.200000 3.300000  0.65000  1.3   NA
[3,]    5 3.181818 2.121212 16.92308   NA   NA

 delta=  2.121212
```

[1] Las redes y cálculos asociados a esta sección se muestran en el ejemplo correspondiente del Capítulo 3

```
 FEntrante=
     [,1]      [,2]      [,3] [,4]      [,5]
[1,]   NA 0.000000 0.000000   NA        NA
[2,]   NA        NA        NA    7        NA
[3,]   NA 4.666667        NA   NA 0.000000
[4,]   NA        NA 2.121212   NA 1.378788
[5,]   NA        NA        NA   NA        NA

 FSaliente=
     [,1] [,2]      [,3] [,4]      [,5]
[1,]   NA    0 0.000000   NA        NA
[2,]   NA   NA        NA  3.5        NA
[3,]   NA    7        NA   NA 0.000000
[4,]   NA   NA 4.666667   NA 2.757576
[5,]   NA   NA        NA   NA        NA

 La ganancia final de la red R es: 2.757576
```

En la segunda iteración se obtiene el ciclo activo $\{5,4,3,5\}$ y se calcula un factor incremental $\delta = \frac{91}{33}$. Al circular delta a través del ciclo activo se obtiene el siguiente patrón:

```
 ---------- Iteracion=  2 ----------

 gnodo=
         [,1]     [,2]     [,3] [,4] [,5] [,6]
[1,] 5.000000 4.000000  3.00000  5.0   NA   NA
[2,] 1.000000 0.500000  1.10000  2.3   NA   NA
[3,] 2.757576 5.757576 12.72727   NA   NA   NA

 delta=  2.757576

 FEntrante=
     [,1]      [,2] [,3] [,4]      [,5]
[1,]   NA 0.000000  0.0   NA        NA
[2,]   NA        NA   NA    7        NA
[3,]   NA 4.666667   NA   NA 3.033333
[4,]   NA        NA  3.5   NA 0.000000
[5,]   NA        NA   NA   NA        NA
```

```
 FSaliente=
     [,1] [,2] [,3] [,4] [,5]
[1,]   NA    0  0.0   NA   NA
[2,]   NA   NA   NA  3.5   NA
[3,]   NA    7   NA   NA  9.1
[4,]   NA   NA  7.7   NA  0.0
[5,]   NA   NA   NA   NA   NA

 La ganancia final de la red R es: 9.1
```

Para la tercera iteración no se localizan ciclos activos, por lo que ahora entra en acción la segunda parte del algoritmo que es maximizar el flujo sobre la red a través de cadenas aumentantes que van del nodo 1 al 5 de R'. La primer cadena es $\{1,3,5\}$ con capacidad incremental $\delta = \frac{329}{30}$. Al circular delta por la cadena aumentante se obtiene el siguiente patrón de flujo:

```
———————— Iteracion=  3 ————————————————

 gnodo=
     [,1]     [,2] [,3] [,4] [,5] [,6]
[1,]    1  3.00000    5   NA   NA   NA
[2,]    1  1.00000    3   NA   NA   NA
[3,]   13 10.96667   NA   NA   NA   NA

 delta=  10.96667

 FEntrante=
     [,1]     [,2]     [,3] [,4] [,5]
[1,]   NA 0.000000 10.96667   NA   NA
[2,]   NA       NA       NA    7   NA
[3,]   NA 4.666667       NA   NA   14
[4,]   NA       NA  3.50000   NA    0
[5,]   NA       NA       NA   NA   NA

 FSaliente=
     [,1] [,2]     [,3] [,4] [,5]
[1,]   NA    0 10.96667   NA   NA
[2,]   NA   NA       NA  3.5   NA
[3,]   NA    7       NA   NA   42
```

```
[4,]    NA    NA  7.70000    NA     0
[5,]    NA    NA       NA    NA    NA

 La ganancia final de la red R es: 42
```

Al correr nuevamente el algoritmo para localizar cadenas aumentantes sobre la red R', se obtiene $\{1,3,4,5\}$ con capacidad incremental $\delta = \frac{61}{30}$. Una vez actualizando el flujo de la red original se obtiene el patrón:

```
 ———————— Iteracion=  4 ————————————————

 gnodo=
         [,1] [,2]       [,3]      [,4] [,5] [,6]
[1,] 1.000000  3.0  4.0000000 5.0000000   NA   NA
[2,] 1.000000  1.0  0.4545455 0.9090909   NA   NA
[3,] 2.033333  7.7 24.2000000        NA   NA   NA

 delta=  2.033333

 FEntrante=
     [,1]     [,2]      [,3] [,4]       [,5]
[1,]   NA 0.000000 13.000000   NA         NA
[2,]   NA       NA        NA    7         NA
[3,]   NA 4.666667        NA   NA 14.0000000
[4,]   NA       NA  2.575758   NA  0.9242424
[5,]   NA       NA        NA   NA         NA

 FSaliente=
     [,1] [,2]      [,3] [,4]       [,5]
[1,]   NA    0 13.000000   NA         NA
[2,]   NA   NA        NA  3.5         NA
[3,]   NA    7        NA   NA 42.000000
[4,]   NA   NA  5.666667   NA  1.848485
[5,]   NA   NA        NA   NA         NA

 La ganancia final de la red R es: 43.84848
```

La siguiente cadena aumentante que se obtuvo en la red auxiliar R' fue $\{1,2,3,4,5\}$ con capacidad incremental $\delta = 14$. Al actualiar el flujo de la red original se obtiene el patrón de las siguientes matrices:

```
————————— Iteracion=  5 ——————————

 gnodo=
     [,1] [,2]       [,3]       [,4]      [,5] [,6]
[1,]    1  2.0  3.0000000  4.0000000 5.0000000   NA
[2,]    1  0.5  0.3333333  0.1515152 0.3030303   NA
[3,]   15 14.0 17.0000000 66.5000000        NA   NA

 delta=  14

 FEntrante=
     [,1] [,2]       [,3] [,4]       [,5]
[1,]   NA   14 13.0000000   NA         NA
[2,]   NA   NA          NA    7         NA
[3,]   NA    0          NA   NA 14.000000
[4,]   NA   NA  0.4545455   NA  3.045455
[5,]   NA   NA          NA   NA         NA

 FSaliente=
     [,1] [,2] [,3] [,4]       [,5]
[1,]   NA    7   13   NA         NA
[2,]   NA   NA   NA  3.5         NA
[3,]   NA    0   NA   NA 42.000000
[4,]   NA   NA    1   NA  6.090909
[5,]   NA   NA   NA   NA         NA

 La ganancia final de la red R es: 48.09091
```

Cuando el algoritmo termina, manda dos veces el mismo resultado, ya que no fue posible maximizar el flujo sobre la red por lo tanto, en este resultado se repitien las matrices de la iteración anterior.

```
 gnodo=
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5   NA   NA   NA   NA
[2,]    1   NA   NA   NA   NA   NA
[3,]   NA   NA   NA   NA   NA   NA
```

```
 delta=  14

 FEntrante=
     [,1] [,2]        [,3] [,4]       [,5]
[1,]   NA   14 13.0000000   NA         NA
[2,]   NA   NA          NA    7         NA
[3,]   NA    0          NA   NA 14.000000
[4,]   NA   NA   0.4545455   NA  3.045455
[5,]   NA   NA          NA   NA         NA

 FSaliente=
     [,1] [,2] [,3] [,4]       [,5]
[1,]   NA    7   13   NA         NA
[2,]   NA   NA   NA  3.5         NA
[3,]   NA    0   NA   NA 42.000000
[4,]   NA   NA    1   NA  6.090909
[5,]   NA   NA   NA   NA         NA
```

Finalmente, se calcula la ganancia total de la red [2]:

```
La ganancia final de la red R es: 48.09091
```

[2]La red asociada al flujo máximo óptimo se encuentra en el Capítulo 3, Figura 3.18.

Apéndice B

Código en R

El siguiente código fue utilizado para resolver el problema del flujo máximo en redes con ganancias. Para un mayor entendimiento, se dividió el código en dos partes. La primera contiene las funciones que componen al programa y la segunda, el desarrollo del algoritmo.

Iniciamos el algoritmo declarando las variables de la red original:

```
vertices<- #Número de vértices que se necesitan
Redq<-matrix(ncol=vertices,nrow=vertices)
Redg<-matrix(ncol=vertices,nrow=vertices)
```

B.1. Funciones

```
########################################
CONSTRUYENDO LA RED AUXILIAR DE COSTOS
```

Construimos la red auxiliar para localizar circuitos de costo negativo, por lo tanto a cada arco se asocia c'_{ij}.

```
########################################
AuxiliarCostos<-function(RedAuxCostos)
{
for(i in 1:vertices)
{
        for(j in 1:vertices)
        {
        if(FEntrante[i,j]>=0 && FEntrante[i,j]<Redq[i,j] && !is.
            na(FEntrante[i,j])) #El arco está saturado
                RedAuxCostos[i,j]<--log(Redg[i,j])
        if(FEntrante[i,j]>0 && FEntrante[i,j]<=Redq[i,j] && !is.
            na(FEntrante[i,j]))
```

```
                RedAuxCostos[j,i]<--log(1/Redg[i,j])
        }
}
return(RedAuxCostos)
}

#####################################
ALGORITMO DE FLOYD
```

Utilizamos este algoritmo como subrutina auxiliar para resolver el problema de rutas más cortas.

```
#####################################

CFloyd<-function(RedAuxCostos)
{
for(i in 1:vertices)
{
        for(j in 1:vertices)
        {
        if(j!=i)
        C[i,j]<-RedAuxCostos[i,j]
        C[i,i]<-0
        if(is.na(C[i,j]))
        C[i,j]<-Inf
        }
}
return(C)
}

zFloyd<-function()
{
for(i in 1:vertices)
{
        for(j in 1:vertices)
        {
        z[i,j]<-i
        }
}
return(z)
}

########################################
ALGORITMO DE FLOYD
########################################

Floyd<-function(C,z)
{
for(k in 1:vertices)
{
```

```
for(i in 1:vertices)
{
        for(j in 1:vertices)
        {
                if(i!=k && j!=k)
                {
                w<- min(C[i,j],(C[i,k]+C[k,j]))
                C[i,j]<-w
                if(C[i,j]!=Inf)
                if(C[i,j]==(C[i,k]+C[k,j]))
                z[i,j]<- z[k,j]
                }
        }
        hayciclo<-0
        if(C[i,i]<0 && z[i,i]!=i)
        {
                hayciclo<-1
        }
}
cat("\n Para la iteración k=",k,"tenemos: \n ")
cat("C= \n")
print(C)
cat("z= \n")
print(z)
if(hayciclo==1)
        break

}
return(z)
}

########################################
```

Una vez detectado el circuito de costo negativo sobre la red auxiliar sede encontrarse una cadena aumentante del circuito al vértice final t. Esto, en la red original, equivale a encontrar un ciclo activo.
Las siguientes funciones tienen como objetivo recuperar el circuito de costo negativo de la matriz de predecesores y concatenarlo con una cadena aumentante que vaya de algún nodo del circuito a t.

RECUPERANDO EL CICLO ACTIVO

```
########################################

RecuperaCiclo<-function(Circuito,inicio)
{
Circuito<-matrix(ncol=vertices+2,nrow=1)
for(m in 1:vertices)
{
        if(z[m,m]!=m)
```

```
                {i<-m
                break}
}

if(inicio!=0)
{
        i<-inicio
}
j<-i
Circuito[1,1]<-i
for(c1 in 2:vertices)
{
        auxiliar<-z[i,j]
        Circuito[1,c1]<-z[i,j]
        if(auxiliar==i)
                break
        j<-auxiliar
}

for(i in 1:vertices+2)
{
        if(is.na(Circuito[1,i]))
                {auxiliar<-(i-1)
                break}
}

Circuito2<-matrix(ncol=vertices+2,nrow=1)
for(j in 1:(i-1))
{
        Circuito2[1,j]<-Circuito[1,i-1]
        i<-i-1
        if(i==0)
                break
}
for(m in 1:vertices+2)
{
        if(is.na(Circuito2[1,m]))
                break
}
indicador<-0
repetido<-0
for(j in 1:auxiliar)
{
        if(j<(auxiliar-2) && !is.na(Circuito2[1,j]) && !is.na(
            Circuito2[1,j+2]) && Circuito2[1,j]==Circuito2[1,j
            +2])
                {
                Circuito2[1,j]<-NA
                indicador<-1
```

```
                repetido<-repetido+1
                }
        if(repetido==2)
                {
                Circuito2[1,j+1]<-vertices
                Circuito2[1,j]<-Circuito[1,1]
                break
                }
}
if (indicador==1)
{
        Circuito3<-matrix(ncol=vertices+2,nrow=1)
        for(m in 1:vertices+2)
        {
        if(!is.na(Circuito2[1,m]))
                break
        }
        for(i in 0:vertices+1)
        {
        if((i+m-1)>vertices)
                break
        if(is.na(Circuito2[1,0+m-1]))
                Circuito2[1,i+1]<-Circuito2[1,m+i-1]
        }
}
if (indicador==0)
        Circuito<-Circuito2
if(Circuito[1,1]==Circuito[1,3])
        Circuito<-matrix(ncol=vertices+2,nrow=1)

return(Circuito)
}

########################################
```

RECUPERANDO LA CADENA AUMENTANTE

Se recupera la cadena aumentante utilizando la matriz de predecesores.

```
########################################

RecuperaCA<-function(Cadena,z)
{
i<-Circuito[1,1]
if(!is.na(Redq[i,vertices]) && FEntrante[i,vertices]<Redq[i,
    vertices])
{
        Cadena[1,1]<-i
        Cadena[1,2]<-vertices
}
```

```
else
{
Cadena[1,1]<-vertices
j<-vertices
FormaParte<-0
for(columna in 2:ncol(Cadena))
{
        if(Circuito[1,columna]==vertices && !is.na(Circuito[1,
            columna]))
        {
        FormaParte<-1
        print("El nodo final forma parte del ciclo activo")
        break
        }
}
if(FormaParte!=1)
{
for(columna in 2:ncol(Cadena))
{
        if(z[i,j]!=i)
        {
                Cadena[1,columna]<-z[i,j]
                j<-z[i,j]
        }
        if(z[i,j]==i)
                break

}
Cadena<-Voltea(Cadena,FormaParte)
}
}
return(Cadena)
}

##########################################
```

FUNCIÓN QUE VOLTEA LA CADENA

```
##########################################

Voltea<-function(Cadena,FormaParte)
{
        Cadena2<-matrix(ncol=vertices+1,nrow=1)
        if(FormaParte!=1)
        {
                for(i in 1:ncol(Cadena))
                {
                if(is.na(Cadena[1,i]))
                break
                }
```

```
                Cadena2[1,1]<-Circuito[1,1]
                for (j in 1:i)
                {
                if(i-j==0)
                        break
                Cadena2[1,j+1]<-Cadena[1,i-j]
                }
        }
return(Cadena2)
}

########################################
```

FUNCIÓN QUE VALIDA LA CADENA

```
########################################

Validar<-function(Cadena,z)
{
        #Cadena<-RecuperaCA(Cadena,z)
        Validador<-0
        for(j in 1:ncol(Cadena))
        {
                if(is.na(Cadena[1,j]))
                        break
        }
                if(j==1)
                        {
                        Validador<-0
                        }
                if(j>1 && !is.na(Cadena[1,j-1]) && Cadena[1,j
                    -1]!=vertices)
                        {
                        Validador<-0
                        }
                if(!is.na(Cadena[1,2]) && Cadena[1,2]==Circuito
                    [1,2])
                        {
                        Validador<-0
                        }
                if(!is.na(Cadena[1,2]) && !is.na(Circuito[1,3])
                   && Cadena[1,2]==Circuito[1,3])
                        {
                        Validador<-0
                        }
                if(j>1 && !is.na(Cadena[1,j-1]) &&  !is.na(
                    Circuito[1,3]) && Cadena[1,j-1]==vertices &&
                    !is.na(Cadena[1,2]) && Cadena[1,2]!=Circuito
                    [1,2] && Cadena[1,2]!=Circuito[1,3])
                        {
```

```
                        Validador<-1
                        }
                if(is.na(Cadena[1,2]) && Circuito[1,1]==vertices
                    && !is.na(Circuito[1,1]))
                        {
                        Validador<-1
                        }
        return(Validador)
}

########################################
CONCATENAMOS EL CICLO ACTIVO Y LA CADENA AUMENTANTE
########################################

Concatenar<-function(CicloCadena)
{
for(i in 1:ncol(Circuito))
{
        CicloCadena[1,i]<-Circuito[1,i]
        if(is.na(CicloCadena[1,i]))
        {
        j<-i-1
        break
        }
}
for(i in 1:vertices)
{
        if(is.na(Cadena[1,i]))
        break
        CicloCadena[1,j+i]<-Cadena[1,i+1]
}
return(CicloCadena)
}

################################################
```

El siguiente paso es calcular la cantidad de flujo delta que puede pasar a través del ciclo activo y la cadena aumentante.

CALCULANDO GANANCIAS

La matriz gnodo tiene los vértices del ciclo activo y de la cadena aumentante en el primer renglón, en el segundo renglón está la ganancia por cada subcadena del ciclo activo y la cadena aumentante y en el tercer renglón se muestra la capacidad incremental de cada arco.

```
################################################

FunGnodo<-function(gnodo)
{
```

```
for (j in 1:ncol(CicloCadena))
{

                gnodo[1,j]<-CicloCadena[1,j]
}

for(j in 1:ncol(CicloCadena))
{
        if(j==1)
        gnodo[2,1]<-1
        if(j>=2)
        {
        if (is.na(Redg[gnodo[1,j-1],gnodo[1,j]]))
                gnodo[2,j]<-(1/(Redg[gnodo[1,j],gnodo[1,j-1]]))*
                    gnodo[2,j-1]
        if (!is.na(Redg[gnodo[1,j-1],gnodo[1,j]]))
                gnodo[2,j]<-Redg[gnodo[1,j-1],gnodo[1,j]]*gnodo
                    [2,j-1]
        if(!is.na(gnodo[2,j]) && gnodo[1,j]==gnodo[1,1])
                gnodo[2,j]<-gnodo[2,j]-1
        }
}

################################
```

CALCULANDO LA CAPACIDAD INCREMENTAL DE CADA ARCO

```
################################

for(j in 1:(ncol(CicloCadena)-1))
{
        if (is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
                gnodo[3,j]<-FSaliente[gnodo[1,j+1],gnodo[1,j]]/
                    gnodo[2,j]
        if (!is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
                gnodo[3,j]<-(Redq[gnodo[1,j],gnodo[1,j+1]]-
                    FEntrante[gnodo[1,j],gnodo[1,j+1]])/gnodo[2,j
                    ]
}

return(gnodo)
}

#####################################
```

SELECCIONAMOS DELTA

El mínimo de las capacidades incrementales por arco

```
#####################################

CalculaDelta<-function(delta,gnodo)
```

```
{
        #El siguiente ciclo nos dice cuantos números antes de un
            NA
        for (i in 1:ncol(gnodo))
        {
        if(is.na(gnodo[3,i]))
        break
        }
        i<-i-1
        #La matriz auxiliarDelta sirve para seleccionar el menor
            delta posible.
        auxiliarDelta<-matrix(nrow=1,ncol=i)
        for(j in 1:ncol(auxiliarDelta))
        {
                auxiliarDelta[1,j]<-gnodo[3,j]
        }
        delta<-min(auxiliarDelta)
return(delta)
}

###################################
```

RESUMEN DE LA ITERACIÓN

```
###################################

Resumen<-function()
{
cat("\n gnodo= \n")
print(gnodo)
cat("\n delta= ",delta,"\n")
cat("\n FEntrante= \n")
print(FEntrante)
cat("\n FSaliente= \n")
print(FSaliente)
g<-ganancia(FSaliente)
cat("\n La ganancia final de la red R es:",g,"\n")
}
```

Volvemos a repetir desde Floyd para ver si la red aún tiene ciclos activos.
Una vez eliminados, sigue maximizar el flujo actual sobre la red del nodo inicial al final. Para lograrlo, utilizamos como subrutina el algoritmo de Floyd de Rutas más cortas.

```
########################################
```

RED AUXILIAR DE CAPACIDADES

```
########################################
```

```
RedCapacidades<-function(RedAuxCapacidades,FEntrante)
{
for(i in 1:vertices)
{
        for(j in 1:vertices)
        {
        if(FEntrante[i,j]>=0 && FEntrante[i,j]<Redq[i,j] && !is.
            na(FEntrante[i,j])) #El arco está saturado
                RedAuxCapacidades[i,j]<-Redq[i,j]-FEntrante[i,j]
        if(FEntrante[i,j]>0 && FEntrante[i,j]<=Redq[i,j] && !is.
            na(FEntrante[i,j]))
                RedAuxCapacidades[j,i]<-FEntrante[i,j]*Redg[i,j]
        }
}
return(RedAuxCapacidades)
}

#######################################
```

RECUPERANDO LA CADENA AUMENTANTE INVERSA DE S A T

```
#######################################

RecuperaCA2<-function(CadenaAumentante,z)
{
for(j in 0:vertices)
{
        if(j==0)
        valor<-z[1,vertices]
        if(j>0)
        valor<-z[1,valor]
        if(valor==1)
        {
                CadenaAumentante[1,j+1]<-z[1,valor]
                break
        }
        CadenaAumentante[1,j+1]<-valor

}
a<-CadenaAumentante[1,1]
b<-CadenaAumentante[1,2]
try(for(j in 3:vertices+1)
{
        if(!is.na(CadenaAumentante[1,j]) || CadenaAumentante[1,j
            ]==a || CadenaAumentante[1,j+1]==b)
        {
        CadenaAumentante[1,j]<-NA
        }
})
CadenaAumentante[1,vertices+1]<-NA
```

```
for(j in 1:vertices)
{
        if(j>1 && !is.na(CadenaAumentante[1,j]) &&
            CadenaAumentante[1,j]==CadenaAumentante[1,j-1])
        {
        CadenaAumentante[1,j]<-CadenaAumentante[1,j+1]
        }
}

for(j in 1:vertices)
{
        if(!is.na(CadenaAumentante[1,j]) &&  !is.na(
            CadenaAumentante[1,j+2]) && CadenaAumentante[1,j]==
            CadenaAumentante[1,j+2])
        {
        CadenaAumentante[1,j]<-NA
        for(ind in j:vertices)
        {
        CadenaAumentante[1,ind]<-CadenaAumentante[1,ind+1]
        }
        }
}
for(j in 1:vertices)
{
if(j>2 && is.na(CadenaAumentante[1,j]) && !is.na(
    CadenaAumentante[1,j-1])&& CadenaAumentante[1,j-1]!=1)
{
CadenaAumentante[1,j]<-1
break
}
}

##########################################
VOLTEAMOS LA CADENA AUMENTANTE
##########################################

CadenaAumentante2<-matrix(ncol=vertices+1,nrow=1)
for(i in 0:vertices)
{
        if(vertices-i>0 && !is.na(CadenaAumentante[1,vertices-i
            ]))
        break
}

if(!is.na(CadenaAumentante[1,vertices-i]))
CadenaAumentante[1,vertices-i+1]<-1
```

```
for(j in 0:vertices-i)
{
        if(vertices-i+1-j>0)
        CadenaAumentante2[1,j]<-CadenaAumentante[1,vertices-i-j
            +1]
}
CadenaAumentante2[1,vertices-i+1]<-vertices
CadenaAumentante<-CadenaAumentante2

return(CadenaAumentante)
}

##########################################
```

CAPACIDAD INCREMENTAL DE LA CADENA

La matriz gnodo2 es igual a gnodo que se utilizó para actualizar el flujo a través de los ciclos activos.

```
##########################################

FunGnodo2<-function(gnodo2)
{
gnodo2<-matrix(ncol=ncol(CadenaAumentante),nrow=3)
for (j in 1:ncol(CadenaAumentante))
{

                gnodo2[1,j]<-CadenaAumentante[1,j]
}
for(j in 1:ncol(CadenaAumentante))
{
        if(j==1)
        gnodo2[2,1]<-1
        if(j>=2)
        {
        if (is.na(Redg[gnodo2[1,j-1],gnodo2[1,j]]))
                gnodo2[2,j]<-(1/(Redg[gnodo2[1,j],gnodo2[1,j
                    -1]]))*gnodo2[2,j-1]
        if (!is.na(Redg[gnodo2[1,j-1],gnodo2[1,j]]))
                gnodo2[2,j]<-Redg[gnodo2[1,j-1],gnodo2[1,j]]*
                    gnodo2[2,j-1]
        if(!is.na(gnodo2[2,j]) && gnodo2[1,j]==gnodo2[1,1])
                gnodo2[2,j]<-gnodo2[2,j]-1
        }
}

########################################
```

CAPACIDAD INCREMENTAL DE LA CADENA AUMENTANTE

```
########################################
```

```
for(j in 1:(ncol(CadenaAumentante)-1))
{
        if (is.na(Redq[gnodo2[1,j],gnodo2[1,j+1]]))
                gnodo2[3,j]<-FSaliente[gnodo2[1,j+1],gnodo2[1,j
                    ]]/gnodo2[2,j]
        if (!is.na(Redq[gnodo2[1,j],gnodo2[1,j+1]]))
                gnodo2[3,j]<-(Redq[gnodo2[1,j],gnodo2[1,j+1]]-
                    FEntrante[gnodo2[1,j],gnodo2[1,j+1]])/gnodo2
                    [2,j]
}
return(gnodo2)
}

########################################
CALCULANDO LA GANANCIA FINAL
########################################

ganancia<-function(FSaliente)
{
        g<-0
        for(i in 1:vertices)
        {
        if(!is.na(FSaliente[i,vertices]))
                        g<-FSaliente[i,vertices]+g
        }
        return(g)
}
```

B.2. Procedimiento: Flujo máximo en redes con ganancias

```
iteracion<-1
Parte1<-1
for(iteracion in 1:15)
{
if(iteracion==1)
{
        FEntrante<-matrix(ncol=vertices,nrow=vertices)
        FSaliente<-matrix(ncol=vertices,nrow=vertices)
        for(i in 1:vertices)
        {
                for(j in 1:vertices)
                {
                if(!is.na(Redq[i,j]))
                        FEntrante[i,j]<-0
```

```
                FSaliente[i,j]<-FEntrante[i,j]*Redg[i,j]
                }
        }
}
if(Parte1==1)
{
        RedAuxCostos<-matrix(ncol=vertices,nrow=vertices)
        RedAuxCostos<-AuxiliarCostos(RedAuxCostos)
        C<-matrix(ncol=vertices,nrow=vertices)
        C<-CFloyd(RedAuxCostos)
        inicio<-0
        if(iteracion==2)
                inicio<-5

        if(iteracion==1)
        {
        z<-matrix(0,ncol=vertices,nrow=vertices)
        z<-zFloyd()
        z<-Floyd(C,z)
        Circuito<-matrix(ncol=vertices+1,nrow=1)
        Circuito<-RecuperaCiclo(Circuito,inicio)
        }

        if(iteracion>1 && cadenasaturada!=1)
        {
        z<-matrix(0,ncol=vertices,nrow=vertices)
        z<-zFloyd()
        z<-Floyd(C,z)
        Circuito<-matrix(ncol=vertices+1,nrow=1)
        Circuito<-RecuperaCiclo(Circuito,inicio)
        }

        if(iteracion>1 && cadenasaturada==1)
        {
        z<-Floyd(C,z)
        Circuito<-RecuperaCiclo(Circuito,inicio)
        }

        Cadena<-matrix(ncol=vertices+1,nrow=1)
        Cadena<-RecuperaCA(Cadena,z)
        Validador<-Validar(Cadena,z)
        contador<-1

        Circuito2<-matrix(ncol=vertices+1,nrow=1)
        Circuito2<-Circuito

        if(is.na(Circuito2[1,1]))
                Validador<-1
```

```
        while(Validador==0)
        {
        inicio<-Circuito2[1,1+contador]
        Circuito<-matrix(ncol=vertices+1,nrow=1)
        Circuito<-RecuperaCiclo(Circuito,inicio)
        Cadena<-matrix(ncol=vertices+1,nrow=1)
        Cadena<-RecuperaCA(Cadena,z)
        Validador<-Validar(Cadena,z)
        contador<-contador+1
        }

        if(Validador==1)
        {
        Cadena<-matrix(ncol=vertices+1,nrow=1)
        Cadena<-RecuperaCA(Cadena,z)
        }

        CicloCadena<-matrix(nrow=1,ncol=vertices+3)
        CicloCadena<-Concatenar(CicloCadena)

        gnodo<-matrix(ncol=ncol(CicloCadena),nrow=3)
        gnodo<-FunGnodo(gnodo)

        if(is.na(gnodo[3,1])==TRUE)
                Parte1<-0
        if(is.na(gnodo[3,1])!=TRUE)
        delta<-CalculaDelta(delta,gnodo)
        if(delta<=0)
                Parte1<-0
}

if(Parte1==0 )#Parte 2: Encontrar cadenas aumentantes de s a t y
    enviar la mayor cantidad de flujo sobre la red.
{
        RedAuxCapacidades<-matrix(ncol=vertices,nrow=vertices)
        RedAuxCapacidades<-RedCapacidades(RedAuxCapacidades,
            FEntrante)
        C<-CFloyd(RedAuxCapacidades)

        z<-matrix(0,ncol=vertices,nrow=vertices)
        z<-zFloyd()
        z<-Floyd(C,z)

        CadenaAumentante<-matrix(ncol=vertices+1,nrow=1)
        CadenaAumentante<-RecuperaCA2(CadenaAumentante,z)
        gnodo2<-matrix(ncol=ncol(CadenaAumentante),nrow=3)
        gnodo2<-FunGnodo2(gnodo2)

        if(is.na(gnodo2[3,1])!=TRUE)
```

```
                delta<-CalculaDelta(delta,gnodo2)
        gnodo<-gnodo2
        if(is.na(gnodo2[3,1])==TRUE)
        {
                Resumen()
                break
        }
}

for(j in 1:vertices) #Actualización del flujo
{
        #Para j=1, es decir, donde empieza la cadena
        if(j==1 && !is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
        {
                FEntrante[gnodo[1,j],gnodo[1,j+1]]<-FEntrante[
                    gnodo[1,j],gnodo[1,j+1]]+delta
                FSaliente[gnodo[1,j],gnodo[1,j+1]]<-FEntrante[
                    gnodo[1,j],gnodo[1,j+1]]*Redg[gnodo[1,j],
                    gnodo[1,j+1]]
                EsForward<-1
        }
        #Si es un arco en sentido contrario
        if(j==1 && is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
        {
                FEntrante[gnodo[1,j+1],gnodo[1,j]]<-FSaliente[
                    gnodo[1,j+1],gnodo[1,j]]-delta
                FSaliente[gnodo[1,j+1],gnodo[1,j]]<-FEntrante[
                    gnodo[1,j+1],gnodo[1,j]]/Redg[gnodo[1,j+1],
                    gnodo[1,j]]
                EsForward<-0
        }
        #Para un nodo distinto al inicial
        if(j>=2 && !is.na(Redq[gnodo[1,j],gnodo[1,j+1]]) && !is.
            na(gnodo[1,j]))
        {
                x<-0
                y<-0
                for(i in 1:vertices)
                {
                if(EsForward==1 && !is.na(FSaliente[i,gnodo[1,j
                    ]]) && FSaliente[i,gnodo[1,j]]!=FSaliente[
                    gnodo[1,j-1],gnodo[1,j]] && !is.na(FSaliente[
                    gnodo[1,j-1],gnodo[1,j]]))
                        x<-FSaliente[i,gnodo[1,j]]+x
                if(EsForward==0 && !is.na(FSaliente[i,gnodo[1,j
                    ]]) && FSaliente[i,gnodo[1,j]]!=FSaliente[
                    gnodo[1,j],gnodo[1,j-1]] && !is.na(FSaliente[
                    gnodo[1,j],gnodo[1,j-1]]))
                        x<-FSaliente[i,gnodo[1,j]]+x
```

```
            if(!is.na(FEntrante[gnodo[1,j],i]) && FEntrante[
                gnodo[1,j],i]!=FEntrante[gnodo[1,j],gnodo[1,j
                +1]] && !is.na(FEntrante[gnodo[1,j],gnodo[1,j
                +1]]))
                    y<-FEntrante[gnodo[1,j],i]+y
            }
            if(EsForward==1)
                    FEntrante[gnodo[1,j],gnodo[1,j+1]]<-x-y+
                        FSaliente[gnodo[1,j-1],gnodo[1,j]]
            if(EsForward==0)
                    FEntrante[gnodo[1,j],gnodo[1,j+1]]<-x-y
            if(is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
                    EsForward<-0
            if(!is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
                    EsForward<-1
            FSaliente[gnodo[1,j],gnodo[1,j+1]]<-FEntrante[
                gnodo[1,j],gnodo[1,j+1]]*Redg[gnodo[1,j],
                gnodo[1,j+1]]

    }
    #Si es un arco en sentido contrario  && gnodo[1,j]!=
        gnodo[1,1]
    if(j>=2 && is.na(Redq[gnodo[1,j],gnodo[1,j+1]]) && !is.
        na(gnodo[1,j]))
    {
            x<-0
            y<-0
            for(i in 1:vertices)
            {
            if(!is.na(FSaliente[i,gnodo[1,j]]) && !is.na(
                FSaliente[gnodo[1,j+1],gnodo[1,j]]))
                    x<-FSaliente[i,gnodo[1,j]]+x
            if(!is.na(FEntrante[gnodo[1,j],i]) && FEntrante[
                gnodo[1,j],i]!=FEntrante[gnodo[1,j+1],gnodo
                [1,j]]/Redg[gnodo[1,j+1],gnodo[1,j]] && !is.
                na(FEntrante[gnodo[1,j+1],gnodo[1,j]]/Redg[
                gnodo[1,j+1],gnodo[1,j]]))
                    y<-FEntrante[gnodo[1,j],i]+y
            }
            x<-x-FSaliente[gnodo[1,j+1],gnodo[1,j]]
            if(is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
                    EsForward<-0
            if(!is.na(Redq[gnodo[1,j],gnodo[1,j+1]]))
                    EsForward<-1
            if(EsForward==1)
                    FEntrante[gnodo[1,j+1],gnodo[1,j]]<-x/
                        Redg[gnodo[1,j+1],gnodo[1,j]]
            if(EsForward==0)
                    FEntrante[gnodo[1,j+1],gnodo[1,j]]<-(y-x
```

```
                        )/Redg[gnodo[1,j+1],gnodo[1,j]]
                FSaliente[gnodo[1,j+1],gnodo[1,j]]<-FEntrante[
                    gnodo[1,j+1],gnodo[1,j]]*Redg[gnodo[1,j+1],
                    gnodo[1,j]]
        }
}
for(j in 1:vertices)
{
        if(j<vertices && !is.na(Cadena[1,j]) && !is.na(Cadena[1,
            j+1]))
        {
        if(FEntrante[Cadena[1,j],Cadena[1,j+1]]==Redq[Cadena[1,j
            ],Cadena[1,j+1]])
        {
                cadenasaturada<-1
                break
        }
        if(FEntrante[Cadena[1,j],Cadena[1,j+1]]!=Redq[Cadena[1,j
            ],Cadena[1,j+1]])
                cadenasaturada<-0
        }
        if(j<vertices && is.na(Cadena[1,j]) && is.na(Cadena[1,j
            +1]))
                cadenasaturada<-1
}
cat("\n----------Iteracion=",iteracion,"----------\n
    ")
Resumen()
iteracion<-iteracion+1
readline(prompt="Presiona [enter] para continuar")
}
```

Bibliografía

[1] Christofides Nicos. *Graph Theory. An Algorithmic Approach*. Academic Press Inc., London, 1975.

[2] Hernández Ayuso Ma. del Carmen. *Introducción a la Teoría de Redes*. Sociedad Matemática Mexicana, México, 2 edition, 2005.

[3] Evans James R. and Minieka Edward. *Optimization Algorithms For Networks And Graphs*. Marcel Dekker, Inc., USA, 2 edition, 1992.

[4] Ahuja Ravindra K., Magnanti Thomas L., and Orlin James B. *Network Flows. Theory, Algorithms and Applications*. Prentice Hall, USA, 1993.

[5] Bondy J.A. and Murty U.S.R. *Graph Theory With Applications*. North-Holland, USA, 1976.

[6] Maurras Jean F. Optimization of the flow through networks with gains. In *Mathematical Programming 3*, pages 135–144. North-Holland, 1972.

[7] Jewell William S. Optimal flow through networks with gains. In *Operations Research*, volume 10, pages 476–499. INFORMS, Berkely, California, 1962.

[8] Grinold Richard C. Calculating maximal flows in a network with positive gains. In *Operatios Research*, volume 21, pages 528–541. INFORMS, Berkely, California, 1973.

[9] Onaga Kenji. Dynamic programming of optimum flows in lossy communication nets. *IEEE Transactions on Circuit Theory*, (2):282–287, September 1966.

[10] Bollobás Béla. *Modern Graph Theory*. Springer, USA, 1998.

[11] Bertsekas Dimitri P. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, USA, 1998.

[12] Bazaraa Mokhtar S., Jarvis John J., and Sherali Hanif D. *Linear Programming and Nerwork Flows*. John Wiley & Sons, Inc., USA, 4 edition, 2010.

[13] Prawda Juan. *Métodos y modelos de investigación de operaciones*, volume 1. Limusa, 2004.