UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

PROGRAMA DE MAESTRIA Y DOCTORADO EN INGENIERIA

FACULTAD DE INGENIERIA

ALGORITMO FORD Y FULKERSON MEJORADO

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERIA

INGENIERIA DE SISTEMAS - INVESTIGACION DE OPERACIONES

P R E S E N T A :

JUAN GONZÁLEZ OVIEDO

TUTOR:

DRA. IDALIA FLORES DE LA MOTA

AÑO

2008

JURADO ASIGNADO:

Presidente:	Dr. ACOSTA FLORES JOSÉ DE JESÚS
Secretario:	M.I. GODELIEVE WELLENS PURNAL ANN
Vocal:	Dra. FLORES DE LA MOTA IDALIA
1er Suplente:	Dra. SÁNCHEZ LARIOS HÉRICA
2do Suplente:	Dra. ELIZONDO CORTÉS MAYRA

Lugar donde se realizó la tesis:

DEPARTAMENTO DE SISTEMAS DE LA DIVISIÓN DE INGENIERÍA MECÁNICA E INDUSTRIAL DE LA FACULTAD DE INGENIERÍA DE LA UNAM.

TUTOR DE TESIS:

Dra. IDALIA FLORES DE LA MOTA

_____

FIRMA

DEDICATORIAS

Agradecimientos:

- Con todo mi amor y respeto, como resultado del esfuerzo y dedicación a lo largo de sus vidas: para mis padres, Francisca Oviedo y Roberto González, quienes con su apoyo, cariño y confianza, me han convertido en persona de provecho, ayudándome a culminar una meta más en mi vida.

- A mis hermanos Azucena, Roberto y Noe; como muestra de seguir luchando para alcanzar lo que anhelamos.

- Para mis amigos José Armando Arroyo, Alberto Muñoz, Claudia Isabel Vargas, Janet Domínguez, Edgar López, Miguel Ochoa, Martha Amor, Paula de la Lama, Nayelli Manzanares, Patricia Olivares, Raúl Pulido y Rubén Téllez; por todos sus buenos consejos en los momentos que los he necesitado y tener la dicha de conservar su amistad sincera y con quienes tuve el privilegio de compartir un salón de clases.

- A todos los profesores que han participado directa e indirectamente en mi formación y elaboración de este trabajo.

- Con admiración y respeto a la Dra. Idalia Flores de la Mota por la dirección de la Tesis y por su apoyo durante y después de la maestría.

- A la M.I. Ann Godelieve Wellens Purnal, por la revisión de esta tesis; así como por sus comentarios y finas sugerencias.

- A la Dra. Hérica Sánchez Larios, por su amistad, por la revisión de esta tesis; así como por sus comentarios y finas sugerencias.

- A la Dra. Mayra Elizondo Cortés, por brindarme su amistad y su confianza para trabajar en el laboratorio, así como también por los valiosos consejos y el apoyo durante y después de la maestría.

- Al Dr. José de Jesús Acosta Flores, por la revisión de esta tesis; así como por las herramientas aprendidas en la toma de decisiones y la creatividad aplicada en este trabajo.

- A la Universidad Autónoma de México y a la Facultad de Ingeniería, por brindarme la oportunidad de realizarme como profesionista.

- Al CONACYT por su apoyo.

ALGORITMO FORD Y FULKERSON MEJORADO

Noviembre del 2008

Indice Pág.

RESUMEN

En este trabajo se propone una mejora al algoritmo de Ford-Fulkerson que soluciona el problema del flujo clásico de red, como el de flujo máximo y el de transporte. Se realiza un análisis del algoritmo de Ford-Fulkerson en el cual se determina su pseudocódigo así como su complejidad computacional; además se dan las bases para analizar la eficiencia de cualquier algoritmo a través de su complejidad, y se muestra una aplicación del problema de flujo máximo, así como el uso de software para resolver problemas de este tipo.

INTRODUCCIÓN

Una red de flujo es un grafo dirigido $G = (V, E)$ donde cada arco (u, v) perteneciente a E tiene una capacidad no negativa. Se distinguen dos nodos: la fuente o nodo s, y el sumidero o nodo s'. Si existen múltiples fuentes y sumideros, el problema se puede simplificar añadiendo una fuente común y un sumidero común.

Es habitual que por las redes circulen flujos. Una red suele ser el soporte, físico o abstracto, por el que circula uno o varios bienes que, de forma general, son ofertados y demandados por algunos puntos localizados en la red. Las conexiones en la misma, bajo factibilidad del problema, aseguran la satisfacción de los requerimientos establecidos.

Se puede considerar un grafo como una red con un flujo, donde un nodo fuente produce o introduce en la red, cierta cantidad de algún tipo de material, y un nodo sumidero lo consume. Cada arco, por tanto, puede considerarse como un conducto que tiene cierta capacidad de flujo. De igual modo que en redes eléctricas (Ley de Kirchhoff), la suma de flujos entrantes a un nodo, debe ser igual a la suma de los salientes (principio de conservación de energía), excepto para el nodo fuente y el nodo sumidero.

Desde una perspectiva global, el concepto de red es de importancia relevante en la concepción y el desenvolvimiento de multitud de problemas de la vida real. La idea de red aparece en procesos naturales, trasciende a fenómenos de índole organizativo y económico, sustentando estructuras de relevancia decisiva en las formas modernas de vivir.

El concepto de grafo resulta de la abstracción de situaciones reales en las que aparecen ciertos lugares o puntos de conexiones entre ellos. Por tanto, un grafo queda definido por un conjunto de vértices o nodos y un conjunto de áreas o aristas que conectan sus vértices. Cuando los elementos de un grafo tienen asociados valores numéricos o magnitudes (pesos, distancias, costos, capacidades, disponibilidades, ofertas, demandas, etc.), aparece el concepto de red.

Ejemplos reales de redes resultan cotidianos y son de una importancia capital en el mundo actual:

- Redes de comunicaciones de todo tipo (terrestres, aéreas, telefónicas, etc.).
- Distribución de bienes (eléctricas, de aguas, de gas, etc.).
- Organización y gestión (servicios, sanidad, seguridad, etc.).

Existen muchas redes por las que circula algún tipo de flujo y, por ello, muchos de los problemas de optimación inherentes son problemas de flujo en redes.

Los problemas de flujos en redes, a pesar de tener antecedentes históricos como el problema de los puentes de Konigsberg, tratado por Euler en el siglo XVIII, o problemas de flujos eléctricos, analizados por Gustav Kirchhoff en el siglo XIX, son estudiados como tales a partir de la década de los años cincuenta en el marco de la investigación operativa.

Los algoritmos clásicos para resolver los problemas de flujo máximo son métodos que se basan en un análisis iterativo basándose en la mínima cota del conjunto de arcos que componen al problema y así se propone un flujo factible, el cual vuelve al problema muy iterativo.

El algoritmo Ford y Fulkerson mejorado es un método intuitivo de flujo máximo que se utiliza para resolver el problema clásico de transporte. Supera considerablemente al método tradicional, tanto en teoría como en la eficiencia computacional.

El problema de flujo máximo es uno de los problemas de optimación en redes más extensamente estudiado y tiene numerosas aplicaciones. Este problema fue introducido por Fulkerson y Dantzig en 1955 y resuelto por vez primera por Ford y Fulkerson con su conocida algoritmo de *caminos incrementales*. Posteriormente Dinic en 1970 introduce el concepto de redes de caminos mínimos, llamadas redes estratificadas.

En 1972, Edmonds y Karp sugieren dos implementaciones polinomiales del algoritmo de Ford y Fulkerson. La primera envía flujo a lo largo de caminos de mayor capacidad residual, la segunda envía flujo a lo largo de caminos mínimos. Hasta este momento, todos los algoritmos de flujo máximo son algoritmos de caminos incrementales. En 1974, Karzanov introduce el concepto de *preflujo* que utiliza sobre redes estratificadas. A partir de entonces, numerosos autores han diseñado algoritmos que incorporando nuevas e importantes ideas, resuelven el problema intentando mejorar la complejidad computacional asociada.

La importancia de los problemas de flujo máximo en redes constituye un campo de trabajo científico, inmerso en la investigación operativa, que reúne a estudiantes e investigadores alrededor de una disciplina de contenido intelectual con un amplio rango de aplicabilidad. Existen miles de aplicaciones en campos como química, física, redes de computadores, economía y ciencias de la gestión, ingenierías de telecomunicaciones y muchas otras ramas de las ingenierías, política y sistemas sociales, planificación, sanidad, etc.

La modificación propuesta sirve para refinar el algoritmo en redes coloreadas, el de etiquetado y los casos de distribución factible, del algoritmo de flujo máximo de Ford-Fulkerson. Con la propuesta de un flujo inicial x el cual utiliza una propuesta de capas estos métodos se vuelven más eficientes puesto que requieren de menos iteraciones para llegar a un flujo óptimo.

Lo propuesta surgió por la deficiencia que existe en los algoritmos cuando inician, pues lo hacen, tomando la cota mínima de los arcos existentes en la red, lo que es conveniente en el caso general donde se desconoce la cantidad inicial en el nodo fuente, pero en casos particulares, como se conoce la cantidad en el nodo fuente, se evita utilizar el criterio de la cota mínima y asi mejorar la eficiencia.

También existe otra deficiencia del método tracional, el cual es la forma de obtener el flujo máximo una vez que el algoritmo termina, pues no toma en cuenta los datos que ya obtuvo y utiliza el teorema de flujo máximo corte mínimo para su obtención.

Esta mejora es aplicable tanto al algoritmo de Ford y Fulkerson de redes coloreadas, al algoritmo de etiquetado que son los casos generales donde no se conoce la cantidad inicial en el nodo fuente y a los casos particulares en redes coloreadas donde si se conoce la cantidad inicial en el nodo fuente.

OBJETÍVO

Modificar el algoritmo de Ford-Fulkerson para resolver el problema de flujo máximo de forma más eficiente que los algoritmos Ford-Fulkerson convencionales.

Este trabajo está estructurado de la siguiente manera:

En el capítulo 1 se muestra al lector el estado del arte incluyendo una reseña histórica del algoritmo de flujo máximo (Ford-Fulkerson) y los avances de los últimos 5 años. Se explican los orígenes del algoritmo así como lo más actual. Además se anexa información de los métodos existentes para resolver el problema de flujo máximo desde el primer método hasta el más actual, así como también, se mencionan los casos particulares de distribución factible del algoritmo de Ford-Fulkerson en redes coloreadas.

En el capítulo 2 se dan las bases teóricas para poder analizar un algoritmo en cuanto a su bondad y así determinar su complejidad computacional, la cual servirá en la práctica para resolver un problema de flujo máximo en un tiempo factible. Así mismo, se agrega el pseudocódigo del algoritmo Ford-Fulkerson y se realiza un pequeño análisis de su complejidad computacional, así como los algoritmos que resuelven el problema de flujo máximo, desde el primero hasta el mas actual, donde se menciona la complejidad de cada uno.

El capítulo 3 representa el capítulo central de la tesis, y muestra la propuesta de mejora del algoritmo de flujo máximo (Ford-Fulkerson). Se realizan las modificaciones al algoritmo existente, así como a la primera modificación realizada por Edmonds y Karp. Los algoritmos, tanto el de etiquetado como el de redes coloreadas, se vuelven más eficientes en cuestión al número de iteraciones que requiere; además, cada algoritmo modificado se ilustra con un ejemplo numérico y se compara con el algoritmo original. También se proponen modificaciones a los casos particulares del algoritmo de Ford-Fulkerson en redes coloreadas.

En el capítulo 4 se muestra una aplicación en la cual se resuelve un problema de transporte de la leche LICONSA en el Distrito Federal en el cual se requiere maximizar el flujo de camiones de las fábricas a los almacenes, utilizando el caso de distribución factible ya mejorado del algoritmo Ford Fulkerson en redes coloreadas y se compara con el método original. Además, se genera un modelo de programación lineal para este problema en especial, y se utiliza un software (LINDO) para resolver el problema de flujo máximo, esto con la finalidad de validar la solución óptima. Es importante destacar que cada capítulo, al final, cuenta con sus propias notas importantes donde se manejan conceptos fundamentales y ejemplos que dan una mejor perspectiva al lector.

Además, el trabajo cuenta con conclusiones y recomendaciones así como un glosario donde se define los conceptos que se manejan en la tesis.

La siguiente figura 0.1 muestra los algoritmos de Ford-Fulkerson de flujo máximo que son susceptibles de mejorarse, puesto que presentan anomalías las cuales afectan la eficiencia de cada algoritmo y con ello la complejidad computacional.

FORMULACIÓN DE LA PROBLEMÁTICA

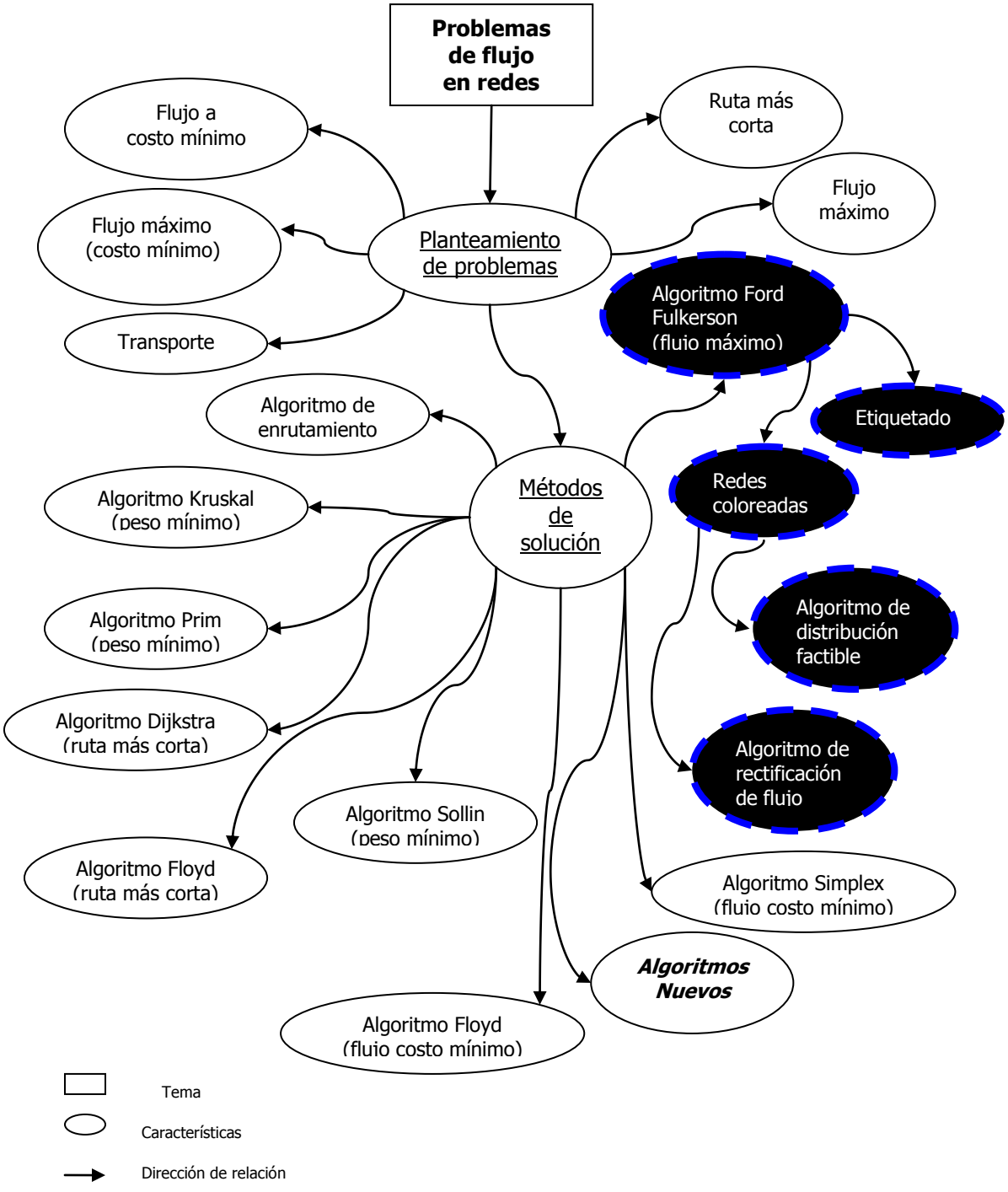


Figura 0.1 Problemas de flujo en redes

PROBLEMÁTICA:

Los algoritmos de Ford-Fulkerson de flujo máximo son susceptibles de mejorarse en su estructura, ya que los métodos tradicionales presentan anomalías las cuales afectan la eficiencia de cada algoritmo y con ello la complejidad computacional.

Algunas desventajas

- Corren en tiempo pseudopolinomial.
- Si contienen arcos racionales el algoritmo puede no converger, y si contienen arcos irracionales el algoritmo no converge.
- Presentan una anomalía principal: la cual se genera cuando existen arcos con cantidades grandes, enteras y finitas, pues el algoritmo tardar en converger.
- En el de etiquetado, si las capacidades son irracionales, el algoritmo podría no terminar. Para algunos casos del problema de flujo máximo que presentan anomalías, el algoritmo de etiquetado no converge, y aunque los sucesivos valores de flujo convergen, estos convergerán a un valor estrictamente menor que el valor del flujo máximo. Por lo tanto, si queremos garantizar la efectividad del algoritmo, debemos seleccionar los caminos incrementales cuidadosamente.
- El algoritmo de etiquetado tiene la desventaja de "olvidar". En cada iteración, el algoritmo genera etiquetas de los nodos que contienen información acerca de los caminos incrementales desde la fuente a otros nodos. Borra las etiquetas cada vez que se pasa de una iteración a la siguiente, si bien mucha de esta información podría ser válida en la siguiente iteración. Eliminando las etiquetas, por lo tanto, se destruye información potencial. Idealmente, retener las etiquetas puede ser provechoso en futuros cálculos.

APORTACIÓN:

Se propone una mejora al algoritmo de flujo máximo (Ford y Fulkerson) en aquellas partes que son susceptibles de mejorar, como son el flujo inicial y la metodología para obtener el flujo máximo, y con ello elevar la eficiencia computacional del algoritmo actual. La importancia de este trabajo es el mejorar la complejidad computacional del algoritmo original pues como se sabe, la complejidad del algoritmo de Ford- Fulkerson es $O(nmU)$, donde n representa el número de nodos iniciales, m el número de iteraciones y U representa que las capacidades en los arcos, son enteras, grandes y finitas.

CAPÍTULO I

ESTADO DEL ARTE

1.1 ANTECEDENTES

La maximización de flujos es un problema típico de la investigación de operaciones, y como se sabe, la investigación de operaciones tuvo sus orígenes durante la segunda guerra mundial y esto con fines bélicos. Dos de los factores fundamentales que impulsaron este desarrollo son los siguientes: el primero fue el progreso sustancial en la modelación matemática y la consolidación del método *simplex* para resolver problemas de programación lineal, desarrollada por el matemático *G. Dantzig.* El segundo factor fue la irrupción del computador digital que agilizó la toma de decisiones en la mayoría de los problemas, ambos avances se lograron antes de finalizar los años 50.

La maximización de flujos tiene muchas aplicaciones, por ejemplo el flujo vial en una ciudad, el flujo de aguas negras, el flujo de informática, etc. Si se sobrecarga una calle, una tubería o un canal que obviamente tiene un limite de capacidad, se generara un problema, posiblemente un flujo mas lento o una tubería con demasiada presión. Los modelos de redes ayudan a tomar una decisión acertada que podría ser mejorar o dar mayor aprovechamiento a los flujos a vías donde tengan más capacidad, creando nuevas vías o eliminando algunas antiguas. También ayudan a maximizar este flujo de manera eficiente de forma tal que se aprovechen al máximo los recursos.

El problema de flujo máximo, fue introducido por Fulkerson y Dantzig en 1955 y resuelto por vez primera por Ford y Fulkerson con su conocido algoritmo de *caminos incrementales*.

Lester Randolph Ford es uno de los pioneros en el campo de la programación de flujos en redes. Su trabajo con Delbert Ray Fulkerson (1924 - 1976) ha puesto la base de casi toda la investigación de flujos en grafos. El artículo de Ford y de Fulkerson (1956) con el problema de flujo máximo estableció el famoso teorema del flujo máximo - mínimo corte.

Mientras trabajó en RAND CORPORATION, Ford Jr, publicó numerosos artículos que no sólo establecieron la base de los flujos de red sino también la futura investigación en este campo. En 1962, Princeton University Press publicó su libro Flow in Networks con el Dr. Fulkerson como co-autor. Este libro contiene todo su trabajo sobre redes.

La mayoría del trabajo de Ford lo hizo en colaboración con Fulkerson. Sin embargo, en 1956 presentó varios artículos firmados por él solo. Ha sido el autor de diversos algoritmos que se han refinado con los años y que todavía se utilizan para solucionar la mayoría de problemas de redes.

1.2 ALGORITMO DE FORD-FULKERSON (FLUJO MÁXIMO)

El algoritmo de Ford-Fulkerson, que data de 1956, es el primer algoritmo propuesto para resolver el problema de flujo máximo. Se basa en el concepto de grafo residual.

En 1962 Ford y Fulkerson establecieron el teorema de flujo máximo-corte mínimo. Resolvieron el problema de flujo máximo a través de algoritmos de trayectorias aumentantes.

El algoritmo de etiquetado es posiblemente el algoritmo más simple para la resolución del problema de flujo máximo. En la práctica, el algoritmo trabaja razonablemente bien. Sin embargo, la cota del peor caso sobre el número de iteraciones no es enteramente satisfactoria para valores de U grandes, donde U representa que las capacidades en los arcos, son enteras y finitas.

Desafortunadamente, el algoritmo de Ford y Fulkerson corre en tiempo pseudopolinomial; más aún, para redes con capacidades irracionales el algoritmo desarrolla una sucesión infinita de trayectorias aumentantes que pueden converger a un valor diferente del valor de máximo flujo. Existen varias versiones mejoradas que superan esta limitación.

En la primera versión del teorema, todos los intervalos de capacidad se toman de la forma $0, c_{-}$, y aparecen nodos sencillos en lugar de conjuntos de nodos $N^{+} y\, N^{-}$. Se pueden usar trucos para reformular el problema; sin embargo, el uso de intervalos arbitrarios y nodos múltiples fuente y sumidero hacen más fácil su aplicación.

Este algoritmo depende de dos conceptos principales:

- Un camino de aumento: es una trayectoria desde el nodo fuente s al nodo sumidero t que puede conducir más flujo.

- La capacidad residual: es la capacidad adicional de flujo que un arco puede llevar

Teorema de Ford-Fulkerson (1962): en cualquier red, el flujo máximo que fluye de la fuente al destino es igual a la capacidad del corte mínimo que separa a la fuente del destino [0].

El teorema demuestra que el algoritmo termina (si no existe ningún camino de aumento), el flujo devuelto es máximo. El costo del algoritmo depende del costo de cada iteración (buscar un camino de aumento y recorrerlo para actualizar el grafo residual) y del número de iteraciones.

Si las capacidades son todas valores enteros, el algoritmo terminará y además, por la propiedad de integridad, el flujo máximo será un valor entero. Sin más información sobre los caminos utilizados, el número de pasos puede ser tan alto como la capacidad máxima de las aristas, con lo que el costo es grande.

Es importante observar que el algoritmo converge sólo si las capacidades para el flujo en los arcos son enteras; sin embargo, en casos donde las capacidades sean racionales, pueden transformarse éstas en enteras multiplicándolas por la potencia adecuada. De este modo, el algoritmo puede usarse también en estos casos.

El primer algoritmo que generaron Ford y Fulkerson para resolver el problema de flujo máximo, es el de etiquetado.

Este algoritmo se puede usar para resolver problemas de:

- Transporte de mercancías (logística de aprovisionamiento y distribución).
- Flujo de gases y líquidos por tuberías.
- Componentes o piezas en líneas de montaje.
- Corriente en redes eléctricas.
- Paquetes de información en redes de comunicaciones.
- Tráfico ferroviario.
- Sistemas de riego, etc.

Es por ello que tiene una gran la importancia el desarrollo de esté algoritmo de flujo máximo el cual resuelve este tipo de problemas, los cuales tienen un gran impacto en la sociedad, la siguiente figura 1.1 muestra a detalle el algoritmo de etiquetado.

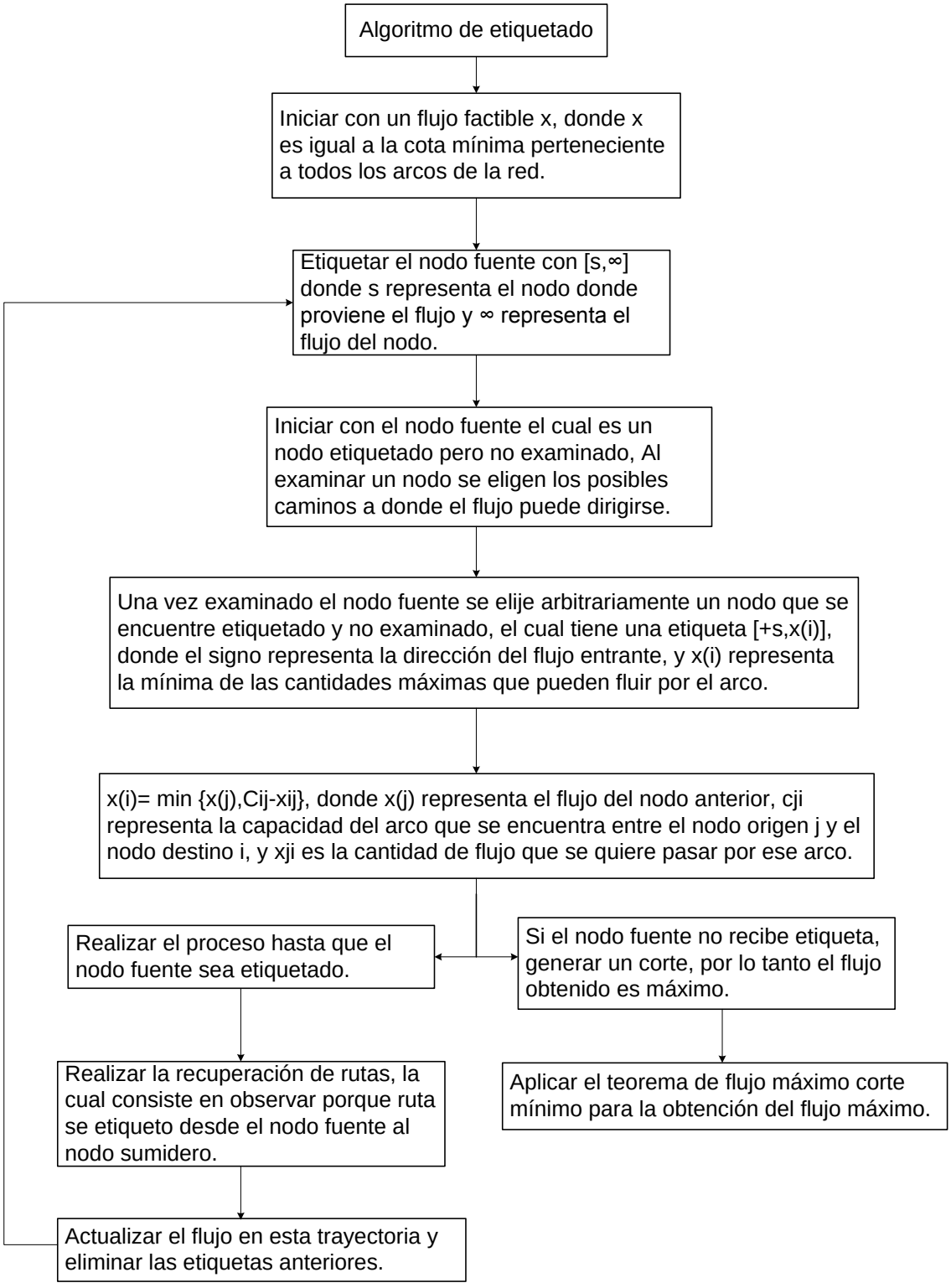


Figura 1.1 Algoritmo de etiquetado

Algoritmo: Ford- Fulkerson de etiquetado

Objetivo: determinar el flujo máximo entre el origen y destino en una red G.

Paso 1 Iniciar con cualquier flujo factible x
Paso 2 Etiquetar el origen con $[\,,\infty]$
Paso 3 Elegir un nodo etiquetado y no examinado; sea j dicho nodo y sean $[\pm j, x(ji)]$ Sus etiquetas.

3.1 Para todo $i \in P^+(j)$ que no este etiquetado y tal que $x_{ji} < c^+_{ji}$ asignar la etiqueta $[+ j, x(i)]$, donde $x(i) = \min\{x(j), c^+_{ji} - x_{ji}\}$

3.2 Para todo $i \in P^-(j)$ que no este etiquetado y tal que $c^+_{ji} > 0$ asignar la etiqueta $[- j, x(i)]$, donde $x(i) = \min\{x(j), x_{ji}\}$

Se puede decir ahora que el nodo j ha sido examinado

Paso 4 Repetir el paso 3 hasta que suceda:

4.1 Si el nodo destino t no tiene etiqueta y todos los nodos etiquetados han sido examinados. Terminar, ya que el flujo factible x es máximo.
4.2 Si el nodo t recibe etiqueta. Ir al paso 5

Paso 5 Sea y = t

5.1 Si la etiqueta de y es de la forma $[+ z, x(y)]$, hacer $x_{zy} = x_{zy} + x(t)$
5.2 Si la etiqueta de y es de la forma $[- z, x(y)]$, hacer $x_{yz} = x_{yz} - x(t)$

Paso 6 Si $z = s$, borrar todas las etiquetas y regresar al paso 2 actualizando el flujo de la trayectoria .
Si $z \neq s$, hacer $y = z$ y regresar al paso 5

Justificación del algoritmo

La justificación del algoritmo queda dada por el teorema de flujo máximo-cortadura mínima.

Cortadura mínima: al terminar de aplicar el algoritmo t no tiene etiqueta; luego, la cortadura de capacidad mínima está dada por el conjunto de arcos:

$\left(N, \bar{N}\right)$, donde $N = \{x \in X \mid x \ \ tiene \ \ etiqueta\}$.

■

Más adelante, cuando surge la coloración de las redes donde se pueden utilizar cuatro colores diferentes verde, blanco, negro y rojo para colorear cualquier red, Ford y Fulkerson no tardaron en generar un nuevo algoritmo en el cual tomaron en cuenta la coloración.

El algoritmo es el siguiente:

Tiene como objetivo determinar el flujo máximo de $N^+ a\, N^-$ en una red G en la cual no existen trayectorias de capacidad ilimitada.

Paso 1 Determinar x, un flujo que satisfaga todas las restricciones del problema.

Paso 2 Colorear los arcos de G de acuerdo a:

Verde	si	$c^-(j) < x(j) < c^+(j)$
Blanco	si	$c^-(j) = x(j) < c^+(j)$
Negro	si	$c^-(j) < x(j) = c^+(j)$
Rojo	si	$c^-(j) = x(j) = c^+(j)$

Paso 3 Utilizar el algoritmo de enrutamiento para determinar una trayectoria compatible con la coloración (es decir, una trayectoria aumentante para x).

3.1 Si se determina la trayectoria $P: N^+ \rightarrow N^-$ compatible con la coloración, calcular :

$$\alpha = \min\left\{\begin{array}{ll} c^+(j) - x(j) & j \in P^+ \\ x(j) - c^-(j) & j \in P^+ \end{array}\right\}$$

Hacer $x = x + \alpha e_p$ y regresar a 2

3.2 Si se determina un corte $Q: N^+ \downarrow N^-$ compatible con la coloración terminar con el flujo máximo x y el corte mínimo Q ya que este ultimo satisface

$$c^+(j) = x(j) \qquad j \in Q^+$$
$$c^-(j) = x(j) \qquad j \in Q^-$$

Por lo tanto el flujo x a través de $Q = \sum_{j \in Q^+} x(j) - \sum_{j \in Q^-} x(j) = C^+(Q)$

El valor en la igualdad puede ser $+\infty$ cuando el corte tiene capacidad ilimitada.

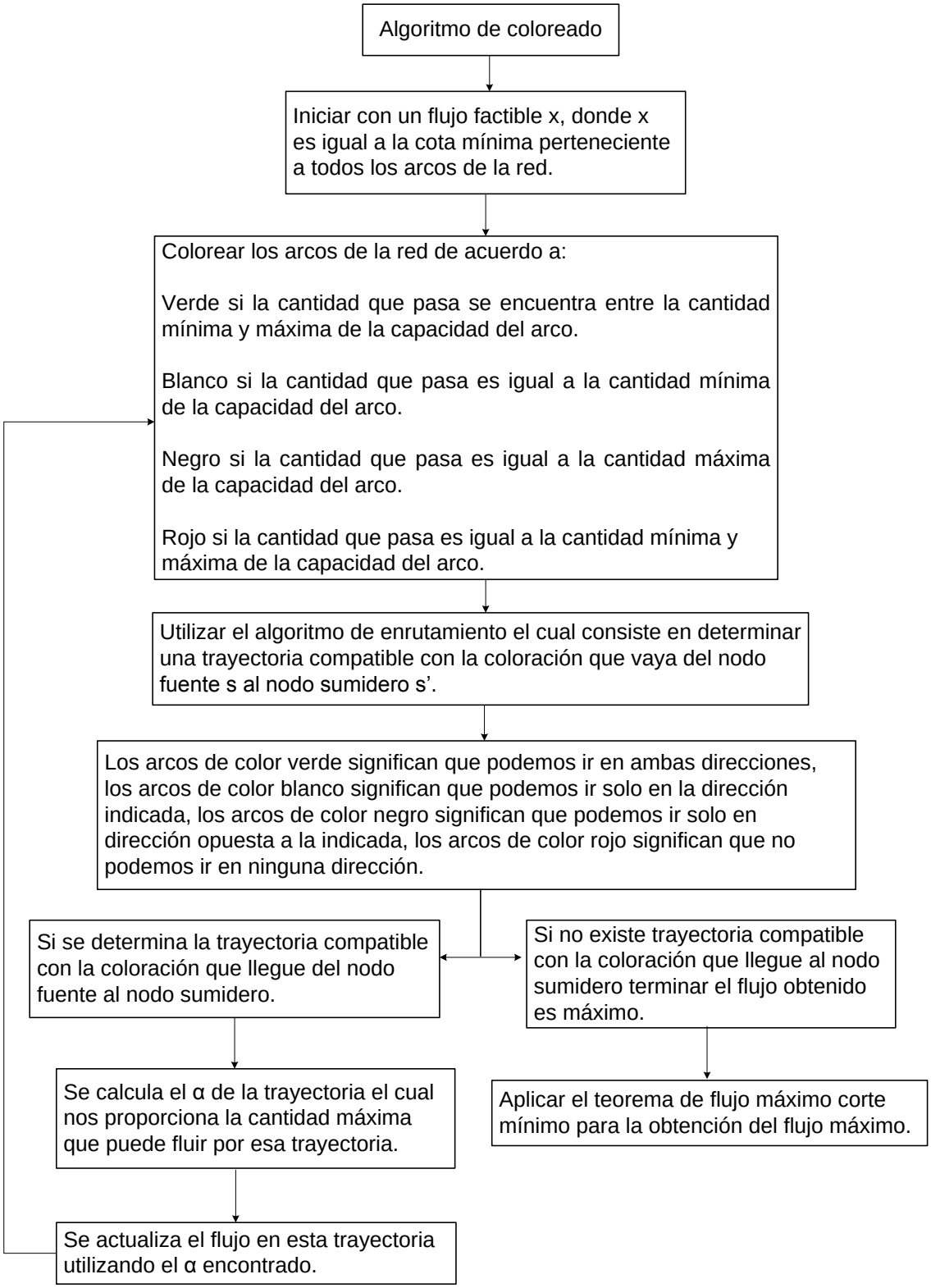


Figura 1.2 Algoritmo de coloreado

Justificación de convergencia del algoritmo

Suponga que las capacidades de los arcos $c^+(j)$, $c^-(j)$ y los valores de flujo inicial $x(j)$ son conmensurables, es decir, son múltiplos de cierta cantidad q. En particular capacidades y flujos enteros son conmensurables con $q=1$. De esta condición de conmensurabilidad se concluye que los números α y $x(j)$, calculados durante el algoritmo, son también múltiplos de la cantidad q. De aquí que, en cada iteración, el flujo de $N^+ a\, N^-$ se incrementa al menos en la cantidad positiva q y por lo tanto se realiza un número finito de iteraciones si no existen trayectorias de capacidad ilimitada. Sin esta condición el algoritmo puede no converger o puede que converja dando una solución errada.

■

La condición de conmensurabilidad puede eliminarse si el algoritmo de Ford y Fulkerson se utiliza con una pequeña modificación, llamada criterio de discriminación de arcos. Este cambio consiste en utilizar, siempre que sea posible, arcos verdes, el flujo mejorado alcanzará la cota superior o la inferior para al menos un arco, el correspondiente al valor de α. De este modo, para la siguiente iteración, al menos un arco se vuelve blanco o negro mientras que los que tenían estos colores no se alteran. Después de un número finito de iteraciones no existen trayectorias aumentantes con todos los arcos verdes, por lo que para continuar el proceso de aumento de flujo (si esto es posible) deberá recurrirse a los arcos blancos o negros. En tal momento hay un conjunto S correspondiente a un corte Q que no contiene arcos verdes; es decir, todo arco j de Q satisface:

$x(j)=c^+(j)$ o $x(j)=c^-(j)$ o ambas. De aquí que:

$$\sum_{j\in Q^+} x(j) - \sum_{j\in q^-} x(j) = \text{[flujo a través del corte } Q\text{]= [flujo } x \text{ de } N^+ \;\; a \;\; N^-\text{]}$$

Por lo tanto, tiene todos sus arcos con términos iguales a $\pm c^+(j)$ o a $\pm c^-(j)$. Puesto que hay un número finito de arcos, existe un número finito de sumas de esta forma y por tanto existe un número finito de valores posibles para el flujo de $N^+ a \;\; N^-$. Ninguno de estos valores se repite durante la aplicación del algoritmo, ya que el flujo se incrementa en cada iteración.

Puede concluirse entonces, que el algoritmo converge en cualquier caso si se utiliza el criterio de discriminación de arcos.

■

La primer modificación al algoritmo de Ford y Fulkerson que en la actualidad se está manejando es la modificación que realizaron Edmonds y Karp: en el algoritmo de Edmonds-Karp (J. Edmonds; R.M. Karp - 1972), el 'camino de aumento' es elegido usando una búsqueda por niveles o en anchura (breadth-first search). El algoritmo de Edmonds-Karp requiere $O(VE^2)$ tiempo de computación, donde V es el número de nodos o vértices, y E el número de arcos de la red.

Edmons y Karp demostraron que si en el algoritmo de Ford-Fulkerson se utiliza el camino de aumento más corto en cada iteración, el algoritmo consigue mejores cotas asintóticas: se vuelve polinomial con costos $O(|V||E|^2)$. El algoritmo de Edmonds-Karp es tan simple como utilizar búsqueda primero en anchura (BFS) para buscar el camino de aumento en el algoritmo de Ford-Fulkerson.

Es por ello que en años recientes, J. Edmonds y R.M. Karp (1972), demostraron que si se usa una búsqueda de primer amplitud en la subrutina de pintado de la red, el algoritmo terminará en tiempo polinomial. El efecto de la búsqueda de primer amplitud consiste en seleccionar en cada iteración una trayectoria aumentante con la menor cantidad de arcos posible. Lo mismo se puede cumplir usando una versión de trayectorias múltiples de la subrutina de pintado de la red.

1.3 CRONOLOGÍA DE LOS ALGORITMOS DE FLUJO MÁXIMO

La siguiente tabla 1.1 muestra la cronología de los algoritmos de flujo máximo, las ideas principales en las que se basa dicho algoritmo. Ford y Fulkerson fueron los primeros en desarrollar el algoritmo que resuelven este tipo de problemas, hasta hoy en día se han desarrollado 10 algoritmos que atacan el problema de flujo máximo, donde muchos de ellos retoman algoritmos anteriores y realizan modificaciones para volverlos de manera más eficiente.

Autores	Ideas
Ford-Fulkerson (1956)	Caminos Incrementales (DFS)
Edmonds-Karp (1972)	Caminos Incrementales Minimos (BFS)
Dinic (1970)	Red Estractificada
Malhotra-Kumar-Maheshwari (1978)	Throughput
Goldberg (1985)	Preflujo;regla FIFO
Version del algoritmo de Goldberg	Preflujo combinacion;regla LIFO y FIFO
Version del algoritmo de Goldberg	Preflujo;regla LIFO
Goldberg-Targan (1986)	
Cheriyan-Maheshwari (1989)	Prefujo;regla Mayor etiqueta distancia
Ahuja-Orlin (1989)	Escalado del exceso

Tabla 1.1 Algoritmos de flujo máximo

Es necesario dar una breve descripción de cada uno de los algoritmos de flujo máximo incluidos en este estudio, desde el primer algoritmo propuesto para resolver problema de flujo máximo hasta el algoritmo más actual.

Algoritmo de Ford y Fulkerson (1956)

Recordando que este algoritmo es conocido como el algoritmo de caminos incrementales. El algoritmo procede identificando caminos incrementales y enviando flujo a través de dichos caminos hasta que la red residual no contiene tales caminos. Utiliza un recorrido en profundidad (DFS) para identificar un camino incremental en la red en cada iteración.

La complejidad del algoritmo es $O(nmU)$.donde n representa el número de nodos, m el número de iteraciones y U representa que las capacidades en los arcos son enteras y finitas. Se denota a este algoritmo por **F&F**.

Algoritmo de Dinic (1970)

Dinic introduce el concepto de red estratificada L. La red estratificada $L = (V, A^*)$ es la red de caminos mínimos, donde A^* es el conjunto de arcos $(i, j) \in A$ que satisfacen la condición $d(i) = d(i) + 1$, y todo camino del nodo fuente al nodo sumidero en L es un camino mínimo de R. El algoritmo de Dinic identifica varios caminos y envía flujo a través de todos ellos a la vez. Utiliza el concepto de flujo bloqueante: un flujo es un flujo bloqueante si no hay caminos incrementales en L . Un flujo máximo es bloqueante pero no al revés. La idea consiste en determinar un flujo bloqueante en la red estratificada y, entonces, actualizar el flujo parcial.

El algoritmo construye, a lo sumo, n redes estratificadas y encuentra un flujo bloqueante en un tiempo $O(nm)$. Consecuentemente, la complejidad del algoritmo es $O(n^2m)$. Se denota a este algoritmo por **DINIC**.

Algoritmo de Edmonds y Karp (1972)

Es conocido como el algoritmo de caminos incrementales mínimos. El método envía flujo a lo largo de un camino mínimo del nodo fuente al nodo sumidero en la red residual. La longitud del camino coincide con el número de arcos contenidos en él. El algoritmo utiliza una búsqueda en amplitud (BFS) para identificar el camino mínimo. Edmonds y Karp muestran que para encontrar el flujo máximo en este caso se requieren, a lo sumo, $12mn$ envíos de flujo. Esto implica una complejidad $O(nm^2)$. Se denota a este algoritmo por **E&K**.

Algoritmo de Malhotra, Kumar y Maheshwari (1978)

Es una generalización al algoritmo de Dinic. Este método presenta una vía alternativa para obtener el flujo bloqueante. Introduce el concepto de *throughput* ("filtrado") de un nodo, que se define como la máxima cantidad de flujo que puede atravesar dicho nodo.

Formalmente: $thr(i) = \min\left\{ \sum_{j \in succ\, i} r_{ij}, \sum_{h \in pred\, i} r_{hi} \right\}$.

El algoritmo selecciona en cada iteración, el nodo i con el menor *throughput* ($thr(i)$). Entonces se sabe que es posible enviar $thr(i)$ unidades de flujo desde i a t y la misma cantidad desde s a i. El proceso se repite hasta que el throughput del nodo fuente o del nodo sumidero se hace igual a cero. En este caso el flujo es determinado. La complejidad del algoritmo es $O(n^3)$, debido a que el tiempo necesario para computar el flujo bloqueante es a lo sumo n^2. Se denota a este algoritmo por **MKM**.

Algoritmo de Goldberg (1985)

Este método mantiene un preflujo. La operación básica consiste en seleccionar un nodo activo y enviar flujo a sus vecinos. Para estimar los nodos activos más cercanos al sumidero, el método usa las etiquetas distancias enviando flujo únicamente a través de arcos admisibles. Si el nodo activo seleccionado no tiene arcos admisibles, su etiqueta distancia se incrementa. A esta operación se le denomina reetiquetar. El algoritmo termina cuando no contiene nodos activos. La operación cuello de botella del algoritmo es el número de envíos no saturantes. Un envió es no saturante si $e(i) < r_{ij}$ para un nodo i (un envió es saturante si $e(i) \geq r_{ij}$). Por lo tanto, la complejidad resultante es $O(n^2 m)$.

En la implementación de este procedimiento, se computan las etiquetas de distancias exactas iniciales, mediante una búsqueda en amplitud (BSF). En este algoritmo, es posible especificar diferentes reglas para seleccionar el nodo activo mejorando el número de envíos no saturantes, es decir, la complejidad. Los criterios son:

- Selección primero-en-entrar primero-en-salir (regla FIFO). Este criterio almacena los nodos activos en una cola y los selecciona de ella. La complejidad de este algoritmo es $O(n^3)$. Se denota a este algoritmo por **G_Q**.

- Selección último-en-entrar primero-en-salir (regla LIFO). Este criterio almacena los nodos activos en una pila y los selecciona de ella. Se denota a este algoritmo por **G_S**.

- Combinación de las reglas FIFO y LIFO. Esta regla almacena los nodos activos en una cola doble; es decir, una cola con dos terminaciones. Los nodos son sacados del principio de la cola. Un nodo activo se añade al principio de la cola la primera vez que es activo; en otro caso, se añade al final de la cola, Se denota al algoritmo por **G_DQ**.

- Selección de la etiqueta mayor. Esta regla consiste en seleccionar un nodo activo con la etiqueta distancia mayor. Para ello, se mantienen las listas $L(h) = \{i : i \text{ es activo y } d(i) = h\}$ gestionadas como colas. Cheriyan y Maheswari muestran que esta regla implica una complejidad $O(n^2\sqrt{m})$. Se denota al algoritmo por **G_HL**.

Algoritmo de Ahuja y Orlin (1989)

Es una versión modificada del algoritmo anterior. Usa una técnica de escalado del exceso para reducir los envíos no saturantes. La idea básica consiste en enviar flujo desde nodos activos con gran exceso a nodos son exceso pequeño. En este algoritmo conocido como algoritmo de escala en el exceso, se define el exceso dominate, Δ, como el menor entero potencia de 2 que satisface $e(i) \le \Delta$, $\forall i \in V - \{s, t\}$. Una iteración comienza con Δ igual a $\Delta/2$. Después de $\log U$ iteraciones, todos los nodos tienen un exceso igual a cero y, por tanto, se ha obtenido el flujo máximo. Para seleccionar un nodo activo con exceso mayor que $\Delta/2$ y con etiqueta distancia mínima, el algoritmo mantiene las listas: $L(r) = \{i \in V : e(i) > \Delta/2 \text{ y } d(i) = r\}$.

La complejidad de este algoritmo es $O(nm + n^2 \log U)$, donde $n^2 \log U$ es el número de envíos no saturantes para todas las fases y nm es una cota superior de las operaciones tales como envíos saturantes, operaciones de actualización de la etiqueta distancia y de determinar arcos admisibles. Este algoritmo se denota por **A&O**.

La importancia de este capítulo es mostrar el estado del arte del problema de flujo máximo, todo lo que se ha desarrollado hasta el momento, así como las diferentes maneras, en que se ha atacado este tipo de problemas.

CAPÍTULO 2

COMPLEJIDAD COMPUTACIONAL

2.1 CONCEPTOS BÁSICOS

Antes de plantear los distintos problemas de flujo máximo en redes, se introducen los conceptos necesarios para analizar la eficiencia de un algoritmo. Todos los algoritmos que resuelven un determinado problema pueden no resultar iguales porque, por ejemplo, alguno de ellos es más rápido que otros. Por lo tanto, se está en la necesidad de encontrar patrones que indiquen cuando un algoritmo es mejor que otro y porque. Para ello introduciremos el concepto de complejidad computacional.

Para que una computadora lleve a cabo una tarea es preciso decirle qué operaciones debe realizar, es decir, se tiene que describir como debe realizar la tarea. Dicha descripción se llama algoritmo.

Un algoritmo describe el método mediante el cual se realiza una tarea. Un algoritmo consiste en una secuencia de instrucciones, las cuales, realizadas adecuadamente, dan lugar al resultado deseado.

Un aspecto importante es el **diseño de algoritmos.** El diseño de buenos algoritmos requiere creatividad e ingenio y no existen, en general, reglas para diseñar algoritmos. En otras palabras, no existe un algoritmo para diseñar algoritmos.

Existen varios algoritmos para resolver un mismo problema, la pregunta es cual es el “mejor”. El mejor es aquel que utilice los mínimos recursos informáticos necesarios para llevar a cabo su tarea determinada.

Generalmente, se está interesado en encontrar el algoritmo más eficiente para resolver un problema. Utilizando el tiempo empleado por el algoritmo como medida de eficiencia del mismo, aunque en general deben tenerse en cuenta los recursos que emplea para obtener la solución.

Se definine *instante* al caso particular de un problema con datos específicos para todos los parámetros del problema. teniendo en cuenta que un algoritmo podrá resolver rápidamente determinados casos particulares de un problema y tardar demasiado en la resolución de otros.

2.2 COMPLEJIDAD DE ALGORITMOS EN TIEMPO Y ESPACIO

El tiempo de ejecución requerido por un algoritmo para resolver un problema es uno de los parámetros importantes en la práctica para medir la bondad de un algoritmo ya que, entre otros factores, el tiempo de ejecución equivale al tiempo de utilización de la computadora y, en consecuencia, costo económico.

Las diferentes instrucciones típicas de un algoritmo son instrucciones de asignación, aritmética y lógicas. El número total de las mismas resulta de la suma de todas las instrucciones definidas anteriormente y determina el tiempo requerido en la ejecución del algoritmo.

Si el tiempo de ejecución es demasiado grande, puede suceder que el algoritmo sea en la práctica inútil, pues el tiempo necesario para su ejecución puede sobrepasar el tiempo disponible de la computadora.

Otro factor importante es que la cantidad de memoria de máquina requerida para almacenar los datos durante el proceso y a la cantidad de memoria utilizada por un algoritmo durante el proceso se suele llamar **espacio** requerido por el algoritmo.

La característica de los problemas de optimación combinatoria de tener un conjunto numerable de soluciones hace que la obtención de la mejor solución sea un problema que podría tener poco interés matemático. Sin embargo es frecuente que problemas de este tipo combinatorio cuyo tamaño es tengan n! soluciones factibles.

Si una computadora puede examinar 10^9 soluciones por segundo podría terminar su tarea para:

$n = 20$ en unos 800 años
$n = 21$ en cerca de 16,800 años

Asi que es muy importante saber la complejidad del problema, porque si se utiliza un algoritmo cuya complejidad computacional es grande no serviría de nada obtener una solución después de 800 años. Para medir la eficiencia de un algoritmo se intentara establecer una relación entre:

- La **duración** de ejecución de dicho algoritmo expresado en términos del número de operaciones elementales.

- El **tamaño**, expresado como el número de caracteres para codificar los datos. Sin embargo, por algoritmo mas “eficiente” se entiende como el mas rápido.

Un algoritmo se considera ***eficiente*** si y solo si es polinomial.

Definición: Un algoritmo es polinomial si el numero de operaciones elementales necesarias para resolver un ejemplo de tamaño n esta acotado por una función polinomial en n.

2.2.1 NOTACIONES PARA EXPRESAR LA COMPLEJIDAD EN TIEMPO

La tasa u orden de crecimiento del tiempo que toma un algoritmo $O(f(n))$ se considera que un algoritmo es mas eficiente que otro si el tiempo que toma su peor caso tiene menor orden de crecimiento.

La notación de $O(f(n))$ representa que está acotando a la función $f(n)$ por arriba, lo que significa que está acotando a $f(n)$ en el peor de los casos.

Se dice que una función $f(n)$ es de orden $O(g(n))$ si y solo si se cumple que existen unas constantes positivas c y n_0, ambas independientes de n, tales que:

$$f(n) \leq cg(n), \quad \forall \quad n \geq n_0$$

Decidir que $f(n)$ es $O(g(n))$ supone que $c\ g(n)$ es un cota superior del algoritmo

Para estimar la bondad de los distintos algoritmos, en este trabajo se utilizara el análisis del peor caso, un estudio experimental de los algoritmos suministra información importante para su utilización práctica, dependiendo del instante del problema que se ha de resolver.

El tiempo de ejecución de un algoritmo depende de la naturaleza y tamaño de la entrada. Una función temporal de la complejidad de un algoritmo es una función del tamaño del problema y especifica el tiempo máximo que se necesita para resolver un instante de un tamaño dado. En otras palabras, la función temporal de complejidad da una medida de la proporción en que se incrementa el tiempo de resolución con respecto a un incremento en el tamaño del problema. Se refiere a la función temporal de complejidad del peor caso de un algoritmo como su cota del peor caso (una cota superior del tiempo invertido).

La notación O grande tiene unas cuantas implicaciones. La complejidad de un algoritmo es una cota superior de tiempo de ejecución del algoritmo para valores de n lo suficientemente grandes. Mas aun, esta notación indica solo los términos más dominantes en el tiempo de ejecución, y representa el hecho de que para un n grande, los términos con un crecimiento menor tienden a ser insignificantes si se compara con el término de mayor crecimiento.

Por ejemplo, si el tiempo de ejecución de un algoritmo es $100n + n^2 + 0.0001n^3$, entonces para todo $n > 100$ el segundo término domina al primer término y para todo $n > 10000$ el tercer término domina al segundo término. Así, la complejidad del algoritmo es $O(n^3)$ .Otra importante implicación de ignorar las constantes en el análisis de la complejidad es que se asume que las operaciones elementales, tales como la suma, resta, multiplicación, división, asignación y operaciones lógicas, requieren una misma cantidad de tiempo.

Una idea que se ha ganado en aceptación en los últimos años es la de considerar que un algoritmo de redes es bueno si la complejidad del peor caso está acotada por una función polinomial de los parámetros del problema. Un algoritmo que cumpla esto es llamado algoritmo polinomial, y un algoritmo polinomial se dice que es fuertemente polinomial si su complejidad esta acotada por una función polinomial en n y m y no aparece $\log(C)$ o $\log(U)$. En caso contrario se dice que el algoritmo es débilmente polinomial.

La notación de $\Omega(f(n))$ representa que está acotando a la función $f(n)$ por abajo, lo que significa que está acotando a $f(n)$ en el mejor de los casos.

Se dice que una función $f(n)$ es de orden $\Omega(g(n))$ si y solo si existen unas constantes positivas c y n_0, tales que:

$$f(n) \geq cg(n), \quad \forall \quad n \geq n_0$$

Decidir que $f(n)$ es $\Omega(g(n))$ supone que $cg(n)$ es un cota inferior del algoritmo

La notación de $\theta(f(n))$ representa que esta acotando a la función $f(n)$ por los extremos, lo que significa que está acotando a $f(n)$ tomando en cuenta el mejor caso y el peor caso en ese rango.

Se dice que una función $f(n)$ es $\theta(g(n))$ si y solo si se cumple que existen unas constantes positivas c_1, c_2 y n_0, tales que:

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \quad \forall \quad n \geq n_0$$

La notación $\theta(\ )$ es más precisa que las notaciones $O(\ )$ y $\Omega(\ )$, ya que $f(n)$ es $\theta(g(n))$ si y solo si $g(n)$ es a la vez una cota inferior y superior de $f(n)$.

2.2.2 FUNCIONES DE COMPLEJIDAD EN TIEMPO MAS USUALES

Las funciones de complejidad algorítmica más usuales, ordenadas de mayor a menor eficiencia (tiempo de ejecución) son:

$O(1)$: Complejidad constante. Es la complejidad mas deseada.

$O(\log n)$: Complejidad logarítmica. Esta complejidad suele aparecer en determinados algoritmos con iteración o recursion no estructural (por ejemplo, busqueda binaria). Noteses que todos los algoritmos, sea cual sea su base, son del mismo orden, por eso se van a representar en base 10.

$O(n)$: Complejidad lineal. Es, en general, una complejidad buena y bastante usual. Suele aparecer en la evaluación de un bucle simple cuando La complejidad de las operaciones interiores es constante o en algoritmos con recursion estructural.

$O(n^2)$: Complejidad cuadrática. Aparece en bucles o recursiones doblemente anidados.

$O(n^3)$: Complejidad cúbica. Aparece en bucles o recursiones triples. Para un valor grande de n empieza a crecer en exceso.

$O(n^k)$: Complejidad polinómica $(k > 3)$. Si k crece, la complejidad del programa es bastante mala.

$O(2^n)$: Complejidad exponencial. Debe evitarse en la medida de lo posible. Puede aparecer en un subprograma recursivo que contenga dos o más llamadas internas. En problemas donde aparece esta complejidad suele hablarse de explosión combinatoria.

Algunos ejemplos de cotas polinomiales son $O(1), O(n), O(n^2), O(nm),$ y además de algunos débilmente polinomiales como $O(\log n)$, $O(m + n\log(C))$.

Se dice que un algoritmo es exponencial en tiempo, si su ejecución no puede ser acotada polinomialmente por la longitud de la entrada. Algunos ejemplos son $O(n^k)$, $O(2^n)$, $O(n!)$ y $O(n^{\log(n)})$. Se dice que un algoritmo es pseudopolinomial si su tiempo de ejecución es polinomicamente acotado en n, m, C y U. algunos ejemplos son $O(m + nC)$ y $O(mC)$.

2.3 TIPOS DE ALGORITMOS

La Teoría de la Complejidad estudia la manera de clasificar algoritmos como buenos o malos y de clasificar problemas de acuerdo a la dificultad inherente de resolverlos.

ALGORITMOS DETERMINÍSTICOS: Tiene la propiedad de que el resultado de cada operación, se define en forma única. En otras palabras nos garantizan en cada paso la solución.

ALGORITMOS NO-DETERMINÍSTICOS: Tiene la propiedad de que el resultado de una o varias operaciones, se determina dentro de un conjunto especificado de posibilidades. En otras palabras están adivinando en cada paso la solución pero no garantizan llegar a la solución optima a un en el mejor de los casos.

Para determinar si los algoritmos son buenos o malos se determinaron las siguientes convenciones:

Convención 1: Todos los algoritmos, desde constantes hasta polinomiales, son Polinomiales.

Convención 2: Todos los algoritmos exponenciales y factoriales, son Exponenciales.

Convención 3: Los algoritmos polinomiales son "buenos" algoritmos. Los Algoritmos exponenciales son "malos" algoritmos.

Por lo tanto los problemas los podemos clasificar como:

- ✓ Fáciles si su método de solución cuyo tiempo de ejecución en una computadora crece de forma “razonable” o moderada (o polinomial).

- ✓ Difíciles si no existe algoritmo.

Técnicamente hablando, determinar si un problema es fácil o difícil se denomina establecer la complejidad computacional del problema. Hay razones para preferir algoritmos polinomiales a algoritmos exponenciales. Las funciones de complejidad exponencial tienen un crecimiento explosivo y en general resuelven solo pequeños problemas.

2.4 COMPLEJIDAD DE LOS PROBLEMAS

Podemos clasificar los problemas en cuatro clases de acuerdo a su grado de dificultad:

1.- Problemas indecidibles. Son aquellos problemas para los cuales no se puede escribir un algoritmo. Por ejemplo, se ha probado que un programa que se detendrá en una Máquina de Turín (1937) es de esta clase. El problema de *Turing* al igual que el décimo problema de *Hilbert* y el de programación cuadrática en enteros son problemas indecidibles, es decir no sólo no existe un algoritmo polinomial para su solución sino que no existe algoritmo.

2.- Problemas intratables (problemas que se demuestran son difíciles). Son aquellos problemas para los cuales no se pueden desarrollar algoritmos polinomiales. En otras palabras, sólo se pueden resolver con algoritmos exponenciales.

3.- Problemas NP (donde NP se entiende por polinomial no determinístico). Esta clase incluye problemas que se pueden resolver en tiempo polinomial si podemos adivinar correctamente que ruta computacional se puede seguir. El concepto de adivinar es extraño ya que todos los programas computacionales son determinísticos. En general, esta clase incluye todos los problemas que tienen algoritmos exponenciales pero que no se ha probado que no se puedan resolver con algoritmos de tiempo polinomial.

4.- Problemas P (donde P se entiende por polinomial).Esta clase incluye todos los problemas que tienen algoritmos de tiempo polinomial. Mucha gente considera a esta clase como una subclase propia de la clase 3.

Un problema se dice polinomial si existe un algoritmo para el cual el tiempo requerido para su solución, está acotado por una función polinomial del tamaño del problema (donde entendemos el tamaño del problema como la longitud de un código, por ejemplo binario de los datos del problema). Se tiene así por ejemplo que en una gráfica $G = X, A$ con $N = X$ nodos y $M = A$ arcos, una ruta más corta se encuentra a lo más en un tiempo $O(MN)$, un flujo máximo en un tiempo $O(N^3)$, un árbol de peso mínimo en un tiempo $O(N^2)$. Sin embargo no todos los problemas combinatorios son polinomiales.

5.- NP-completos: Subconjunto de problemas NP que son los más difíciles de resolver. Se dice que un problema pertenece a la clase NP-completo si cada problema de la clase NP puede reducirse en tiempo polinomial a este. Además cualquier problema de decisión, si es o no elemento de NP. Al cual se puede reducir en tiempo polinomial un problema NP-completo, se llama NP-difícil, ya que es, en cierto sentido, al menos tan difícil como los NP-completos.

Problemas como el del agente viajero o el de la mochila se conocen como NP-completos, donde NP se entiende como polinomial no determinístico. El problema del agente viajero se puede resolver con un algoritmo determinístico como el de recursividad, donde se especifica en cada paso que arco se debe considerar desde un nodo dado.

Para la pregunta de que si existe un recorrido con una distancia total que sea menos que B, el algoritmo recursivo implícitamente prueba todos los recorridos posibles y da una respuesta afirmativa si tal recorrido existe y negativa en caso contrario.

Ya que los recorridos se prueban uno por uno, y el último puede ser el que cumple con la propiedad deseada, el algoritmo recursivo para resolver el problema del agente viajero se considera $O(c^n)$.

Un algoritmo no determinístico puede adivinar correctamente que arco se debe incluir en el recorrido. Si existe un recorrido con una distancia total menor que B, el algoritmo no determinístico requiere un tiempo de $O(n)$ para calcular la distancia total del recorrido y verificar si tal recorrido existe. Si el problema del agente viajero se puede resolver por un algoritmo no determinístico en tiempo polinomial se dice que el problema pertenece a la clase NP.

Existe un subconjunto de problemas NP que son los más difíciles, en el siguiente sentido (los problemas en el subconjunto son polinomialmente equivalentes):

Cualquier problema en NP se puede reducir a cualquier problema en éste subconjunto, podemos resolver todos los problemas en NP en tiempo polinomial. Los problemas en éste subconjunto se llaman NP-completos. Esquemáticamente se puede ver en la figura 1.1

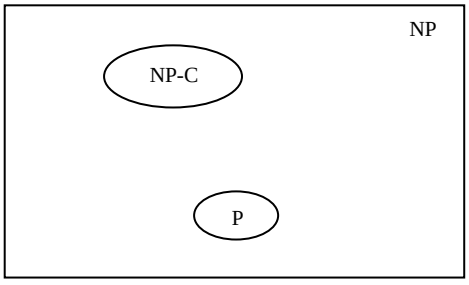


Figura 1.1

2.5 ALGORITMO DE FORD Y FULKERSON

Este primer algoritmo considera en cada iteración un camino incremental en R y envía a través de el una cantidad de flujo igual a la capacidad residual del camino. El algoritmo termina cuando no quedan caminos incrementales en R. El algoritmo genérico de caminos incrementales es de la manera siguiente:

Algoritmo de Ford-Fulkerson;
Begin
$x := 0;\ \ F := 0;$
While haya un camino P de $s\ \ a\ \ t$ en R **do**
Begin
$\Delta := \min\{r_{ij} : (i,j) \in P\}$; Δ es el cuello de botella del camino
Enviar Δ unidades de flujo a lo largo de P y actualizar R;
$F := F + \Delta$
end
end.

Para cada arco $(i,j) \in P$ se actualiza $r_{ij} = r_{ij} - \Delta\ \ y\ \ r_{ij} = r_{ij} + \Delta$. Suponiendo que se actualizan las capacidades residuales en algún punto del algoritmo y que se pregunta por el efecto sobre los flujos de los arcos. De la definición de la capacidad residual se tiene que un flujo adicional de Δ unidades sobre el arco (i,j) en la red residual R, corresponde a: (1) un incremento de x_{ij} en Δ unidades en la red original, o (2) un decremento de x_{ij} en Δ unidades en la red original, o (3) una combinación de ambas.

Finalmente, se supone que están dados los valores para las capacidades residuales. ¿Cómo se podrian determinar los flujos x_{ij}? Se observa que $r_{ij} = u_{ij} - x_{ij}\ \ y\ \ r_{ji} = x_{ij}$ y, por lo tanto, se puede obtener los valores de x_{ij} haciendo $x_{ij} = u_{ij} - r_{ij}\ \ o\ \ x_{ij} = r_{ji}$.

En la descripción del algoritmo precedente, no se discutieron algunos detalles importantes: (1) como identificar un camino incremental o mostrar que la red no contiene tales caminos, y (2) si el algoritmo termina en un número finito de iteraciones y si, cuando lo hace, obtiene un flujo máximo.

Se introduce a continuación una implementación específica del algoritmo genérico de caminos incrementales conocido como *algoritmo de etiquetado*. El algoritmo de etiquetado usa una técnica de búsqueda para identificar un camino en R de la fuente al sumidero.

El algoritmo parte del nodo fuente encontrando a todos los nodos que son alcanzables desde la fuente a lo largo de un camino en la red residual. En cualquier paso, el algoritmo divide el conjunto de nodos de la red en dos grupos: etiquetados y no etiquetados. Los nodos etiquetados son aquellos nodos que el algoritmo ha alcanzado en el proceso de la búsqueda; los nodos no etiquetados son aquellos nodos que el algoritmo todavía no ha alcanzado en el proceso. Cuando el nodo sumidero es etiquetado, el algoritmo envía la máxima cantidad de flujo posible sobre el camino desde $s \;\; a \;\; t$. En este punto, se borran las etiquetas y se repite el proceso. El algoritmo termina cuando, examinados todos los nodos etiquetados, el nodo sumidero permanece sin etiquetar, implicando que el nodo fuente no esta conectado al nodo sumidero en la red residual.

El siguiente esquema especifica los pasos de etiquetado. Se necesita un índice $\mathrm{Pr}ed(i),$ para cada nodo etiquetado i, que indica el nodo que motivo la asignación de etiqueta a i. Este índice será utilizado posteriormente para indicar el camino de $s \;\; a \;\; t$, sin mas que empezar con $i = t$ y, con $\mathrm{Pr}ed(i),$ llegar hasta $s.$

Procedure Aumentar;
Begin
Utilizar los índices de los predecesores para identificar un camino incremental P de la fuente al sumidero;
$\delta := \min \{ r_{ij} : (i, j) \in P \}$;
$F := F + \delta$
Envía δ unidades de flujo a lo largo de P y actualiza las capacidades residuales
end;

El algoritmo de etiquetado que se muestra enseguida termina obteniendo un flujo máximo, debido a que su criterio de parada es la no existencia de caminos incrementales. Ahora veremos en cuantos pasos determina el flujo máximo basándose en el criterio del peor caso. En cada iteración del algoritmo, se etiqueta algún nodo i al menos una vez, inspeccionando cada arco de $A(i)$. Por lo tanto, en el proceso de etiquetado, se examina cada arco al menos una vez. Esto requiere $O(m)$ computaciones. Si todas las capacidades de los arcos son enteras y acotadas por un número finito $U,$ entonces la capacidad del $s-t$ corte dado por $(s, V - s)$ es a lo sumo nU. Este nos da una cota superior del valor del flujo máximo. Como el algoritmo de etiquetado incrementa el valor del flujo en al menos una unidad en cada iteración, necesita entonces nU iteraciones para obtener dicho valor de flujo. Esto implica que el algoritmo tiene una complejidad $O(nmU)$.

Algoritmo de Etiquetado;
Begin
$x := 0$; $F := 0$;
repeat
Etiquetado [i] := **false** y $\mathrm{Pr}ed(i)$:=0 para todos los nodos;
Etiquetado [s] := **true**; $L := \{s\}$;
While $(L \neq \vartheta)$ **and not** (etiquetado [t]) do
Begin
Selecciona un nodo $i \in L$;
for $(i, j) \in A(i)$ do
if j no esta etiquetado y $r_{ij} > 0$ then
Begin
$\mathrm{Pr}ed(j) := i$;
Etiquetado [j] := true;
$L := L + \{j\}$
end;
end;
if etiquetado [t] then Aumentar
until not (etiquetado [t]);
end.

El algoritmo de etiquetado es posiblemente el algoritmo más simple para la resolución del problema de flujo máximo. En la práctica, el algoritmo trabaja razonablemente bien. Sin embargo, la cota del peor caso sobre el número de iteraciones no es enteramente satisfactoria para valores de U grandes.

Edmonds y Karp demuestran que si en cada iteración del algoritmo de Ford y Fulkerson se selecciona el camino incremental mínimo, se necesitan como máximo $mn/2$ iteraciones, obteniendo así un algoritmo de complejidad fuertemente polinomial. Para la obtención de los caminos mínimos se utiliza un recorrido en amplitud en el grafo que requiere un esfuerzo de $O(m)$. Así se obtiene un algoritmo de complejidad $O(m^2 n)$.

A continuación se muestra en la tabla 2.1 la complejidad computacional de los algoritmos que resuelven el problema de flujo máximo desde el primero hasta el más actual vistos en el capitulo1.

Codigo	Autores	Complejidad
F&F	Ford-Fulkerson (1956)	$O(nmU)$
E&K	Edmonds-Karp (1972)	$O(m^2n)$
DINIC	Dinic (1970)	$O(mn^2)$
MKM	Malhotra-Kumar-Maheshwari (1978)	$O(mn^2)$
G_Q	Goldberg (1985)	$O(n^3)$
G_DQ	Version del algoritmo de Goldberg	$O(n^3)$
G_S	Version del algoritmo de Goldberg	$O(mn^2)$
G_HL	Goldberg-Targan (1986)	$O(n^2\sqrt{m})$
	Cheriyan-Maheshwari (1989)	
A&O	Ahuja-Orlin (1989)	$O(mn+n^2\log U)$

Tabla 2.1 Complejidad de los Algoritmos de flujo máximo

2.6 NOTAS IMPORTANTES

La **teoría de la complejidad computacional** es la rama de la teoría de la computación que estudia, de manera teórica, los recursos requeridos durante el cálculo para resolver un problema.

Los recursos comúnmente estudiados son **el tiempo** (mediante una aproximación al número y tipo de pasos de ejecución de un algoritmo para resolver un problema) **y el espacio** (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema).

Se pueden estudiar igualmente otros parámetros, tales como el número de procesadores necesarios para resolver el problema en paralelo. Los problemas que tienen una solución con orden de complejidad lineal son los problemas que se resuelven en un tiempo que se relaciona linealmente con su tamaño.

Hoy en día, las máquinas resuelven problemas mediante algoritmos que tienen como máximo una complejidad o coste computacional polinómico, es decir, la relación entre el tamaño del problema y su tiempo de ejecución es polinómica. Éstos son problemas agrupados en el conjunto P. Los problemas con costo factorial o combinatorio están agrupados en NP. Estos problemas no tienen una solución práctica, es decir, una máquina no puede resolverlos en un tiempo razonable. Para ver por qué las soluciones de tiempo exponencial no son útiles en la práctica, se puede considerar un problema que requiera 2^n operaciones para su resolución (n es el tamaño de la fuente de información). Para una fuente de información relativamente pequeña, $n=100$, y asumiendo que una computadora puede llevar acabo 10^9 operaciones por segundo, una solución llevaría cerca de $4x10^{12}$ años para completarse, mucho más tiempo que la actual edad del universo.

Es fundamental saber como se genera el tiempo de ejecución de cierto algoritmo y no confundirlo con la complejidad computacional del algoritmo para esto se vera el siguiente ejemplo:

El algoritmo ordena una lista L de n números naturales de menor a mayor.

Entrada: Una lista $L = a_1, a_2, ..., a_n$

Paso 1. Asignar $j = n,\ \ i = 1$

Paso 2. Si $i < j$ entonces si $a_i > a_{i+1}$ intercambiar a_i con a_{i+1} . En caso contrario (es decir $i \geq j$) ir al Paso 4

Paso 3. Asignar $i = i + 1$ y volver al paso 2

Paso 4. Si $j > 2$ asignar $j = j - 1$ e $i = 1$ y volver al Paso 2

Paso 5. FIN (la lista L esta ordenada)

Aplicar este algoritmo a una lista de números para deducir cual es el tiempo de ejecución del mismo.

Suponga que esta es la solución para una lista dada $L = x_1, x_2, ..., x_8$ entonces se tiene:

	Veces	No Pasos
Paso 1.	1	1
Paso 2.	3	n
Paso 3.	1	$n-1$
Paso 4.	2	1
Paso 5.	1	1

Tiempo de ejecución $= 2 + (n-2)(3n + n - 1 + 2)$

$= 2 + (n-2)(4n+1)$

$= 2 + 4n^2 - 8n + n - 2$

$= 4n^2 - 7n$

Complejidad $= \lim_{n\to\infty} (4n^2 - 7n)$

$= \lim_{n\to\infty} (4 - 7/n\ n^2)$

$= \lim_{n\to\infty} (n^2)$

$= (\infty^2)$

Por lo tanto $O(n^2)$ ■

En este capítulo se sientan las bases para poder determinar si un algoritmo es eficiente computacionalmente, y se dice eficiente si su tiempo de ejecución se puede acotar por medio de un polinomio, tomando en cuenta el peor de los casos.

CAPÍTULO 3

ALGORITMO DE FLUJO MÁXIMO MEJORADO

3.1 PROPUESTA DE MEJORA

Es habitual que por las redes circulen flujos. Una red suele ser el soporte, físico o abstracto, por el que circula uno o varios bienes que, de forma general, son ofertados y demandados por algunos puntos localizados en la red. Las conexiones en la misma, bajo factibilidad del problema, aseguran la satisfacción de los requerimientos establecidos.

Como ya se habló sobre la importancia que representa el flujo máximo en las redes así como la infinidad de aplicaciones que tiene éste, ahora se hablara sobre la propuesta de mejora al algoritmo de Ford-Fulkerson en redes coloreadas. Primero hay que especificar que esta mejoría no se realizó antes porque la mayoría de las modificaciones, se trataron de hacer primero para un caso particular y después generalizar. En este caso, la modificación se realiza tanto para los casos particulares del algoritmo de Ford-Fulkerson en redes coloreadas como para el caso general en que no se conoce el flujo inicial en el nodo fuente.

Antes que nada, debe quedar claro que el método para los casos particulares de sólo funciona cuando se conoce la cantidad en nuestro nodo fuente y se quiere saber cuál es el flujo máximo que puede llegar al nodo sumidero. Además, dicha modificación se realiza utilizando los casos particulares del algoritmo de Ford-Fulkerson en redes coloreadas. Puesto que el algoritmo de Ford-Fulkerson de etiquetado es un algoritmo que ha sido muy estudiado y que realmente ya fue modificado y a pesar que su modificación es buena, se proponen soluciones para eliminar sus anomalías como son la pérdida de memoria, cuando tenemos nodos con capacidades irracionales el algoritmo no converge y se propone también una modificación al algoritmo para mejorar su eficiencia. Además, con el tiempo se han generado varios algoritmos de etiquetado que cada vez son más eficientes en cuanto a la complejidad computacional.

La modificación propuesta sirve para refinar los casos particulares del algoritmo de flujo máximo de Ford-Fulkerson en redes coloredas, el caso general y el algoritmo de etiquetado. Con la propuesta de flujo x estos métodos se vuelven más eficientes puesto que requieren de menos iteraciones para llegar a un flujo óptimo.

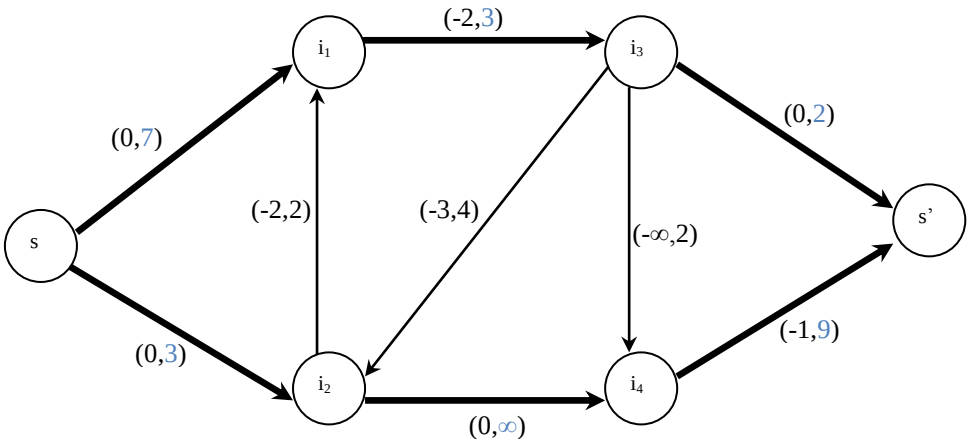
Lo propuesta surgió por la deficiencia que existe en los algoritmos cuando inician, pues lo hacen, tomando la cota mínima de los arcos existentes en la red, lo que es

conveniente en el caso general donde se desconoce la cantidad inicial en el nodo fuente, pero en los casos particulares, como se conoce la cantidad en el nodo fuente, se puede evitar utilizar el criterio de la cota mínima y asi mejorar la eficiencia.

Para mejorar el algoritmo original de flujo máximo de Ford-Fulkerson en el cual no se conoce la cantidad inicial en el nodo fuente, fue necesario entender bien cómo funciona este algoritmo, la propuesta de mejora se puede extender al método de flujo máximo de etiquetado, la cual consiste en iniciar no con la cota mínima contenida en todos los arcos de la red, que es lo que utiliza el algoritmo original, la modificación al algoritmo propone separar el problema por capas y comenzar con un flujo inicial, sobre cada capa de la red a estudiar.

Las capas son aquellas trayectorias que salen del nodo fuente y llegan al nodo sumidero cubriendo a la red, la primer capa que se analiza se encuentra en la frontera superior de la red, una vez que una capa es elegida se busca la cota máxima que se encuentra contenida en todos los arcos que pertenecen a esa capa en la red y se elimina esa capa al asignarle un flujo inicial igual a la cota máxima que se encontró, nuevamente se toma otra capa y asi sucesivamente hasta llegar a la última capa que se encuentra en la frontera inferior de la red, se busca la cota máxima que se encuentra contenida en todos los arcos que pertenecen a esa capa en la red y se elimina esa capa al asignarle un flujo inicial igual a la cota máxima que se encontró, y todos aquellos arcos que no pertenecen a una capa adquieren un flujo inicial igual a cero, que es equivalente a la cota mínima perteneciente a todos los arcos de la red, en caso de que la cota mínima sea distinta de cero se tendría que tomar en cuenta la divergencia entre los nodos.

El siguiente ejemplo muestra con más detalle en que consiste esta propuesta.

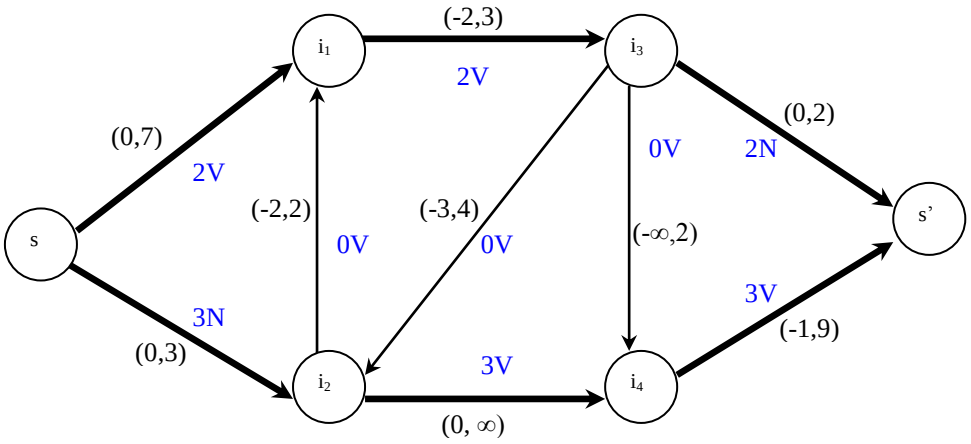


Capa 1: $s \to i_1 \to i_3 \to s'$ = $\max \in c_1$: {(0,7),(-2,3),(0,2)} = 2

Capa 2: $s \to i_2 \to i_4 \to s'$ = $\max \in c_2$: {(0,3),(0, ∞),(-1,9)} = 3

Como se puede ver las cotas máximas de cada arco que pertenecen a la primera capa (frontera superior) de la red son: {7, 3, 2} en este caso la cota máxima que se encuentra contenida en todos los intervalos de capacidad en los arcos es 2.

Es importante entender el concepto de cota máxima, puesto que no se está hablando del mínimo de las cotas máximas. El flujo inicial de los arcos que no son parte de una capa son igual a cero, puesto que la cota mínima perteneciente a todos los arcos de la red es 0 y por lo tanto no hay necesidad de revisar la divergencia entre los nodos, el flujo inicial seria el siguiente.



Una nueva forma de obtener el flujo máximo a partir de esta modificación es la siguiente:

$$Flujo \quad Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j)$$

Donde:

α_i : representa las cantidades de actualización de flujo de cada iteración.

$x(j)$: representa la cota máxima perteneciente a todos los arcos contenidos en cada capa la red.

n : representa el número de iteraciones.

m : representa la cantidad de capas de la red.

Es importante destacar que esta formulación no está en la bibliografía especializada consultada. Esta forma de obtener el flujo máximo de una red, sólo es aplicable para el algoritmo de Ford-Fulkerson mejorado que es una aportación de este trabajo.

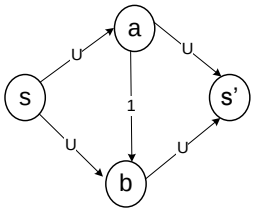
La importancia de este trabajo es el mejorar la complejidad computacional del algoritmo original pues como se sabe, la complejidad del algoritmo de Ford-Fulkerson es $O(nmU)$, donde n representa el número de nodos iniciales, m el número de iteraciones y U representa que las capacidades en los arcos, son enteras, grandes y finitas. La propuesta de mejora, es muy significativa para valores muy grandes de U puesto que su complejidad se reduce a $O(nmu)$, donde $u = U - x(j)$ y representa que las capacidades en los arcos, son enteras, no tan grandes y finitas, puesto que se les está restando la cota máxima perteneciente a todos los arcos contenidos en cada capa de la red, y así evita la anomalía que presenta el algoritmo original: que para valores muy grandes de U, el algoritmo requiere de demasiadas iteraciones y utiliza mucho tiempo máquina para llegar a la solución, mientras que el algoritmo modificado requiere una cantidad pequeña de iteraciones.

Algoritmos de Ford-Fulkerson de flujo máximo

1. Algoritmo de etiquetado

Desventajas:

1.- Corre en tiempo pseudopolinomial
2.- Si contiene arcos racionales el algoritmo puede no converger.
3.- Perdida de memoria, debido al reetiquetado de cada iteración.
4.- Si contiene arcos irracionales el algoritmo puede no converger.
5.- Presenta la anomalía principal.



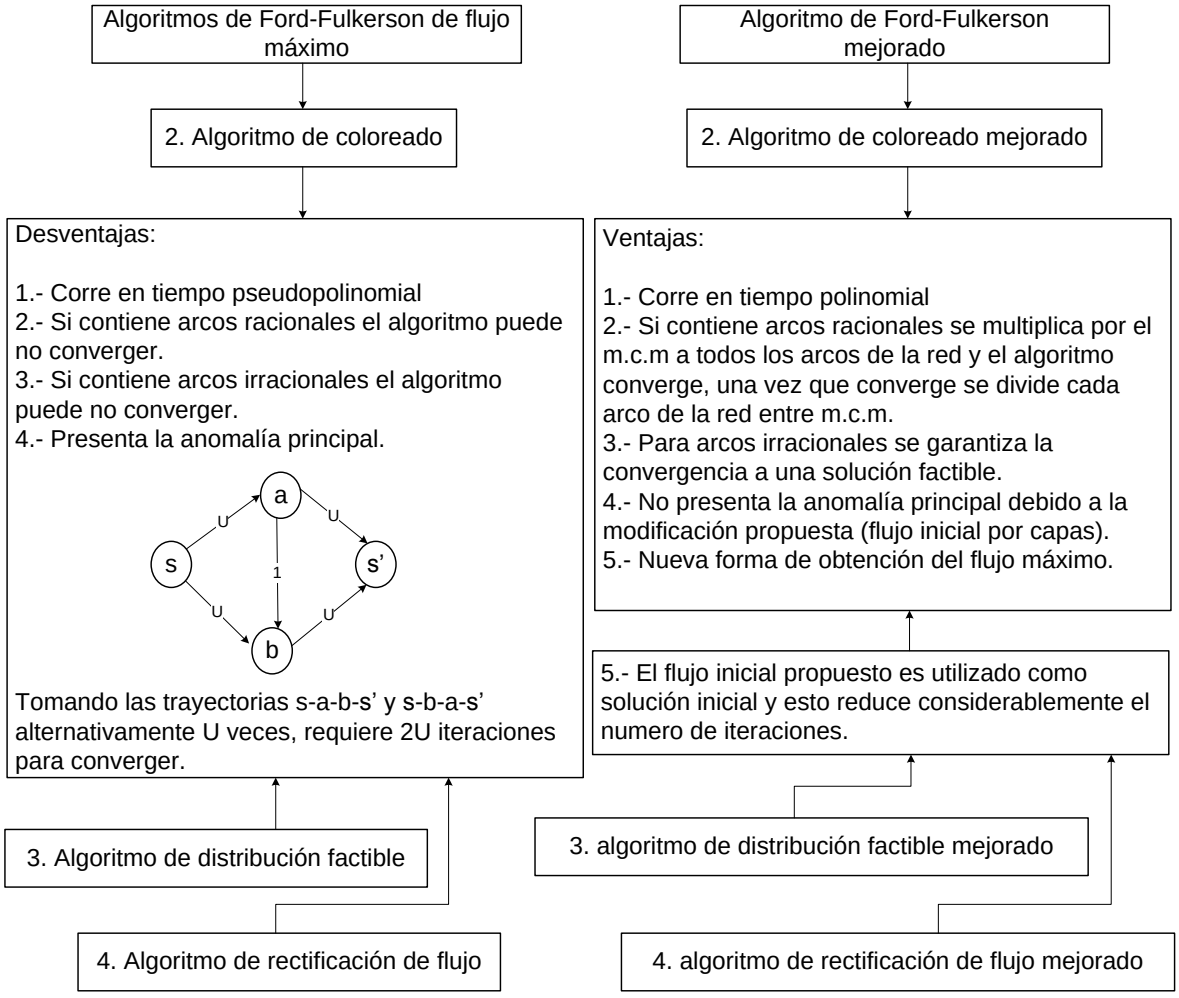
Tomando las trayectorias s-a-b-s' y s-b-a-s' alternativamente U veces, requiere 2U iteraciones para converger.

Algoritmo de Ford-Fulkerson mejorado

1. Algoritmo de etiquetado mejorado

Ventajas:

1.- Corre en tiempo polinomial
2.- Si contiene arcos racionales se multiplica por el m.c.m a todos los arcos de la red y el algoritmo converge, una vez que converge se divide cada arco de la red entre m.c.m.
3.- No tiene perdida de memoria porque las etiquetas se almacenas como superíndices.
4.- Para arcos irracionales se garantiza la convergencia a una solución factible.
5.- No presenta la anomalía principal debido a la modificación propuesta (flujo inicial por capas).
6.- Nueva forma de obtención del flujo máximo.



3.2 ALGORITMO FORD-FULKERSON MEJORADO

El algoritmo de Ford-Fulkerson mejorado se inicia tomando en cuenta las capas que cubren a la red y proponiendo un flujo inicial que es igual a la cota máxima que se encuentra contenida en todos los intervalos de capacidad pertenecientes a los arcos de la capa que se esté analizando en red, este flujo inicial saldrá de el nodo fuente por cada arco perteneciente a una capa en la red, las capas son trayectorias que cubren a la red iniciando de un nodo fuente a un nodo sumidero y debe de recordarse que una vez que una capa adquiere su flujo inicial, ésta ya no es tomada en cuenta, por lo tanto es como si se eliminara. Este proceso se realiza hasta que ya no exista alguna capa sin flujo inicial, los arcos que no pertenecen a una capa tendrán un flujo igual a cero y en caso de que la cota mínima perteneciente a todos los arcos de la red sea diferente de cero tendría que tomarse cuenta la divergencia entre los nodos, sin tomar en cuenta el nodo fuente y el nodo sumidero, si se quisieran tomar en cuenta se tendría que generar un arco artificial que vaya del nodo sumidero al nodo fuente.

Primer paso determinar un flujo inicial para cada capa de la red que debe considerar el criterio de la cota máxima perteneciente a todos los arcos en cada capa a analizar, este flujo saldrá de el nodo fuente por cada arco perteneciente a una capa y los arcos que no pertenezcan a una capa, iniciarán con un flujo igual a cero, que es equivalente a utilizar el criterio de la cota mínima perteneciente a todos los arcos de la red, si ésta no es cero, se tiene que cumplir la divergencia entre nodos. Esto con el fin de que se generen inicialmente trayectorias de color negro y el algoritmo termine en un menor número de iteraciones.

Segundo paso colorear los arcos de G de acuerdo a:

Verde	si	$c^-(j) < x(j) < c^+(j)$
Blanco	si	$c^-(j) = x(j) < c^+(j)$
Negro	si	$c^-(j) < x(j) = c^+(j)$
Rojo	si	$c^-(j) = x(j) = c^+(j)$

Tercer paso utilizar el algoritmo de enrutamiento el cual consiste en determinar una trayectoria compatible con la coloración (es decir, una trayectoria aumentante para x).

Si se determina la trayectoria $P: N^+ \to N^-$ compatible con la coloración, calcular:

$$\alpha = \min\begin{Bmatrix} c^+(j) - x(j) & j \in P^+ \\ x(j) - c^-(j) & j \in P^+ \end{Bmatrix}$$

Hacer $x = x + \alpha e_p$ y regresar a 2

Si se determina un corte $Q: N^+ \downarrow N^-$ compatible con la coloración terminar con el flujo máximo x y el corte mínimo Q ya que este ultimo satisface

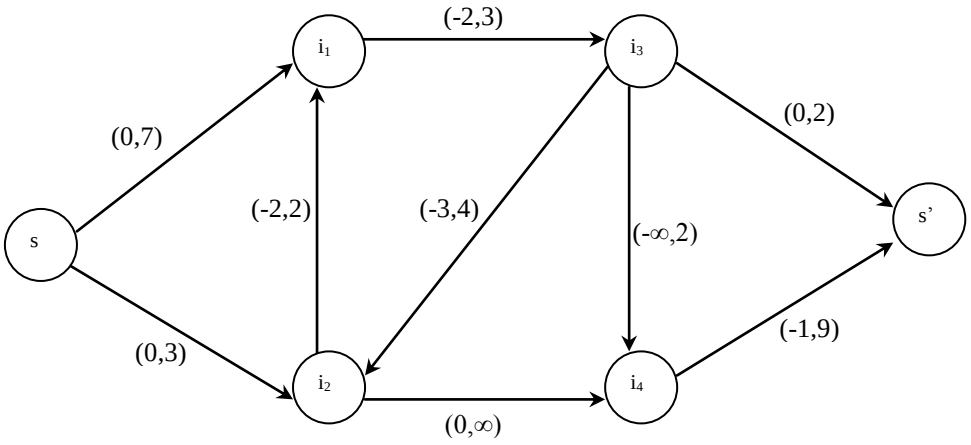
$$c^+(j) = x(j) \qquad j \in Q^+$$
$$c^-(j) = x(j) \qquad j \in Q^-$$

Por lo tanto el flujo x a través de $Q = \sum_{j \in Q^+} x(j) - \sum_{j \in Q^-} x(j) = C^+(Q)$

El valor en la igualdad puede ser $+\infty$ cuando el corte tiene capacidad ilimitada.

Ejemplo 3.1

Primero utilizar el algoritmo original. En la siguiente red se busca el flujo máximo, comenzando con un flujo igual a cero $x(j)=0$ para toda $j \in R$ esto es posible porque todos los intervalos de capacidad contienen al cero.



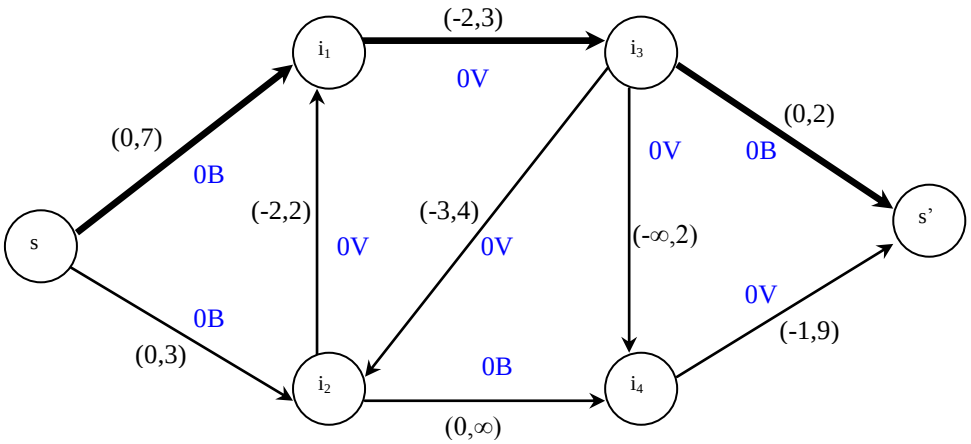
Solución:

ITERACIÓN 1

Paso 1: Comenzar con un flujo igual a 0 $x(j)=0 \ \forall \ j \in R$ esto es posible porque todos los intervalos de capacidad contienen al 0.

Paso 2: Colorear

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

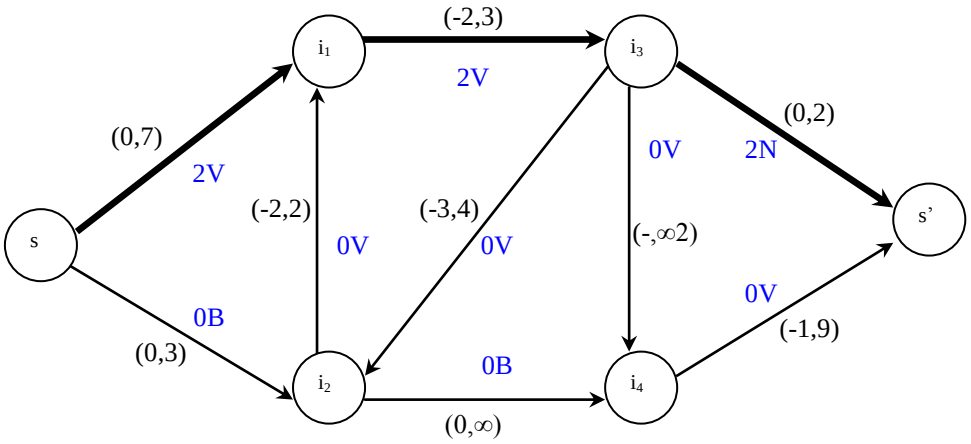


Paso 3.1: Calcular α

$P_1 : s \to i_1 \to i_3 \to s'$

$\alpha_1 = \min \{7\text{-}0,\ 3\text{-}0,\ 2\text{-}0\}$
$= \min\{7,\ 3,\ 2\} = 2$

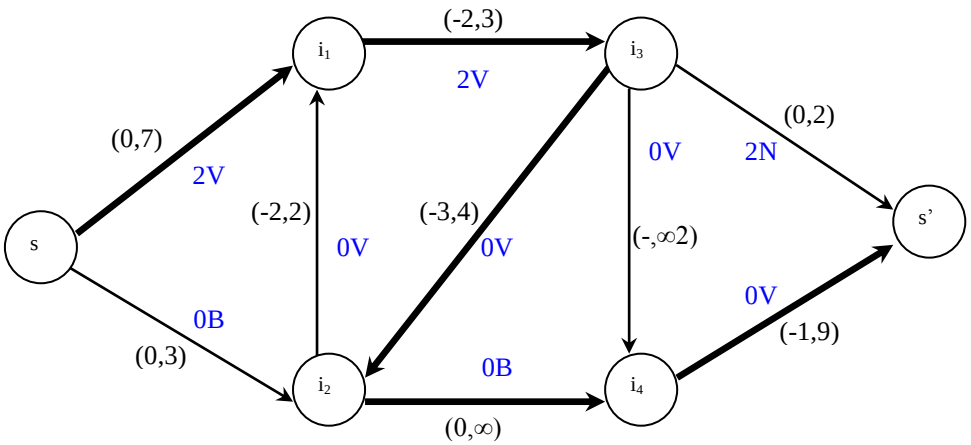
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 2

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

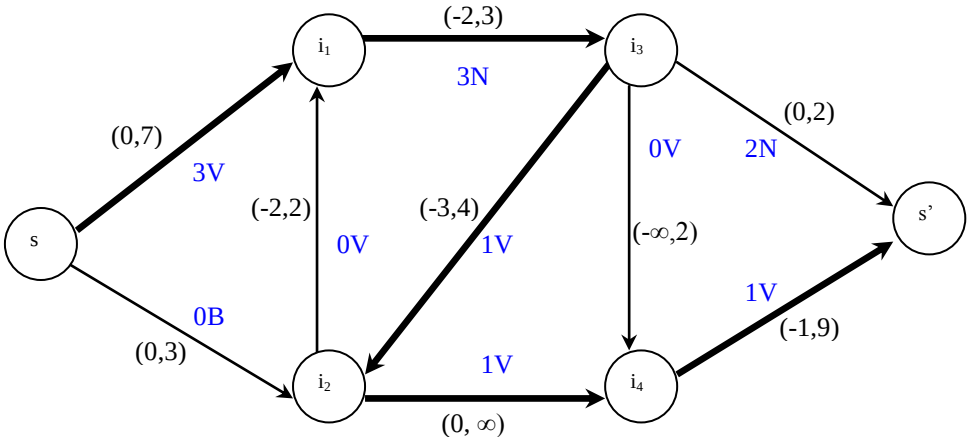


Paso 3.1: Calcular α

$P_2 : s \to i_1 \to i_3 \to i_2 \to i_4 \to s'$

α_2 = min {7-2, 3-2, 4-0, ∞-0, 9-0}
= min {5, 1, 4, ∞, 9}
= 1

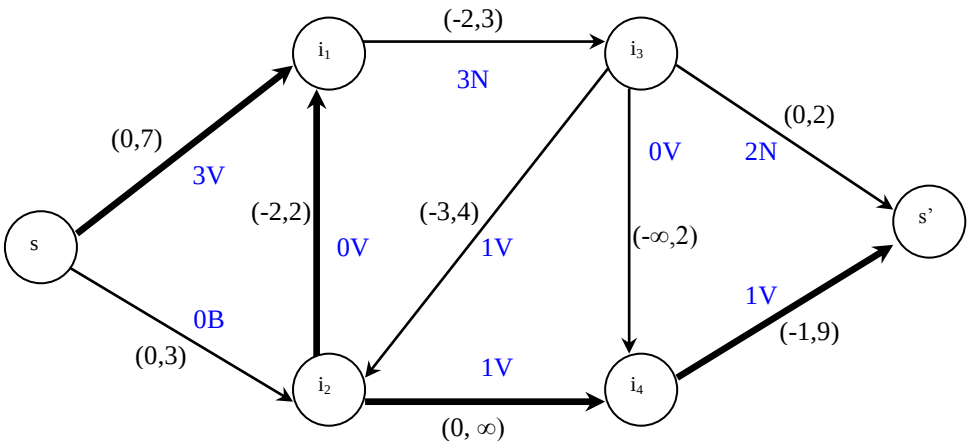
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 3

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

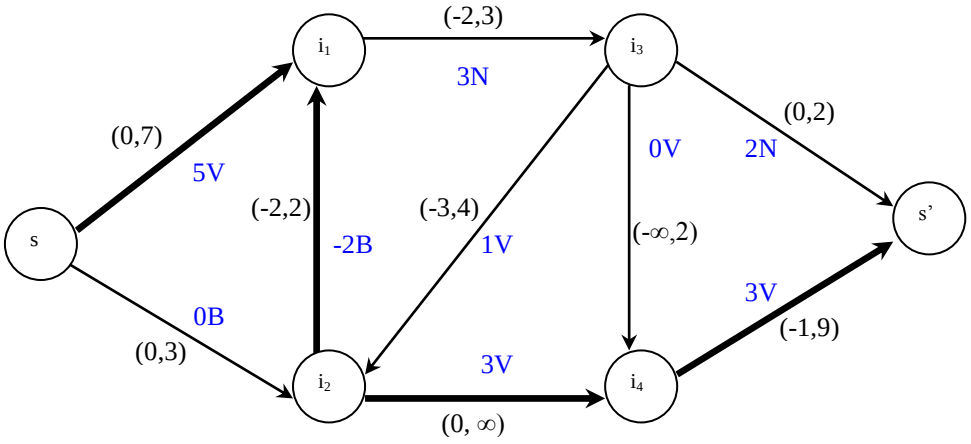


Paso 3.1: Calcular α

$P_3 : s \to i_1 \leftarrow i_2 \to i_4 \to s'$

$\alpha_3 = \min \{7\text{-}3,\ 0\text{-}(-2),\ \infty\text{-}1,\ 9\text{-}1\}$
$= \min \{4,\ 2,\ \infty,\ 8\}$
$= 2$

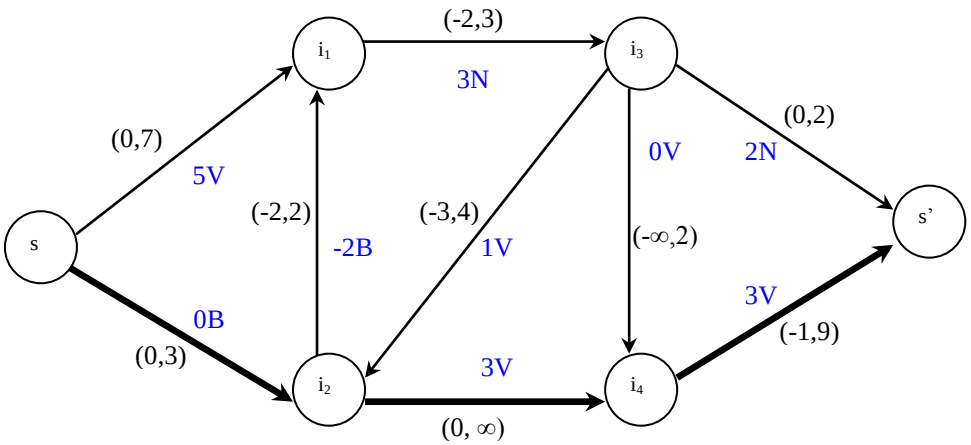
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 4

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

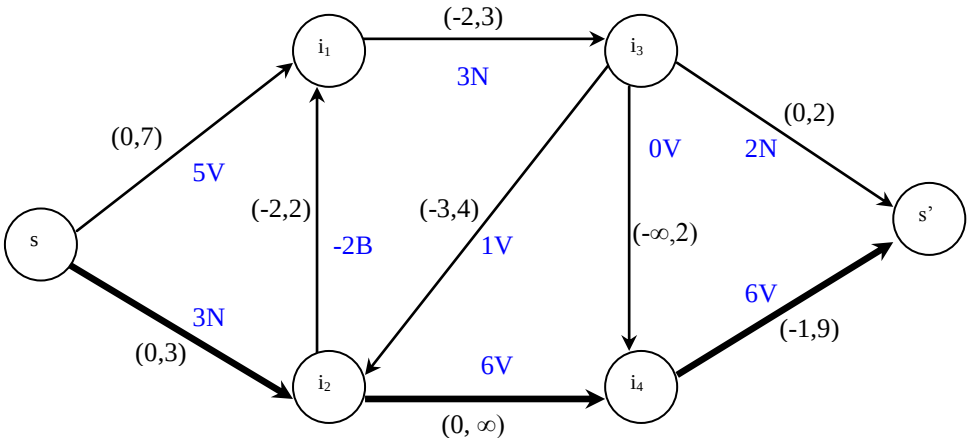


Paso 3.1: Calcular α

$P_4 : s \to i_2 \to i_4 \to s'$

$\alpha_4 = \min \{3\text{-}0, \infty\text{-}3, 9\text{-}3\}$
$= \min \{3, \infty, 6\}$
$= 3$

Paso 3.2: Actualizar el flujo con α en la trayectoria.

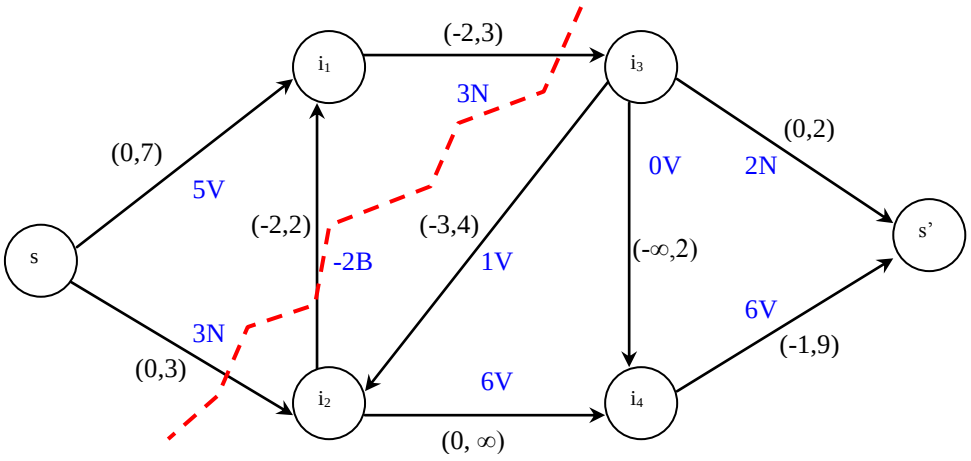


ITERACIÓN 5

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

Paso 3.2: No existe trayectoria compatible con la coloración, se determino un corte Q compatible con la coloración terminar.



Por lo tanto el flujo x a través de $Q = \sum_{j \in Q^+} x(j) - \sum_{j \in Q^-} x(j) = C^+(Q)$

$S = \{s, i_1\}$ $\qquad Q^+ = \{(s,i_3),(s,i_2)\} = 3+3 = 6$

$Q^- = \{(s,i_1)\} = -2$

$y(s) = 5+3 = 8$

$y(i_1) = 3-5-(-2) = 0$ $\qquad Q = Q^+ - Q^- = 6 - (-2) = \mathbf{8}$

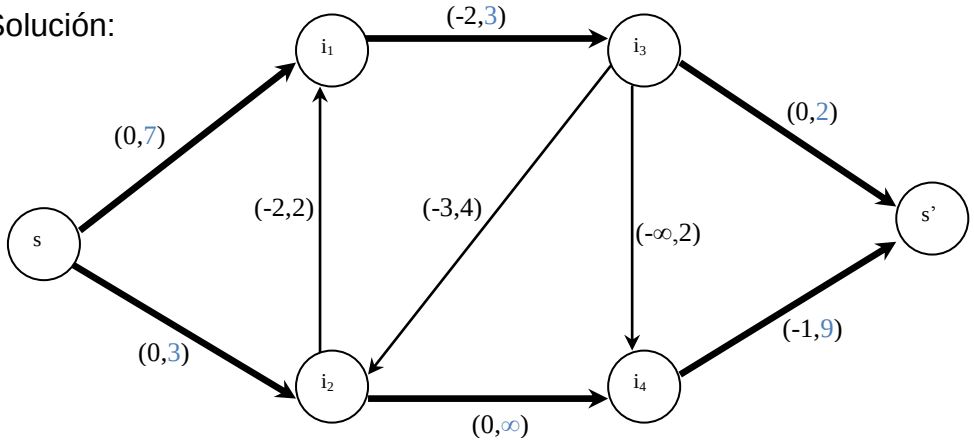
$\therefore\ y(s) = \mathbf{8}$

El flujo máximo se obtiene mediante el teorema de flujo máximo corte mínimo, el cual indica que el flujo máximo es igual a la sumatoria de todos los arcos que salen del corte menos la sumatoria de todos los arcos que entran al corte. Otra forma de obtener el flujo máximo es mediante la suma de las divergencias de los nodos que se encuentran dentro del corte, la cual es igual al flujo máximo.

Ejemplo 3.2

Ahora utilizar la modificación propuesta para resolver el ejemplo 3.1, se inicia al determinar un flujo inicial que es igual a la cota máxima contenida en los arcos pertenecientes a cada capa de la red $\forall\ j \in C_R$. Comenzar con un flujo inicial de $x(j) = 2$ para la capa 1 $\forall\ j \in C_1$ y un flujo de $x(j) = 3$ para la capa 2 $\forall\ j \in C_2$ y esto es posible porque todos los intervalos de capacidad contienen al dos y al tres respectivamente, los arcos que no pertenecen a una capa comienzan con un flujo igual a la cota mínima perteneciente a todos los arcos de la red con $j \in R$ en este y la mayoría de los casos es 0.

Solución:



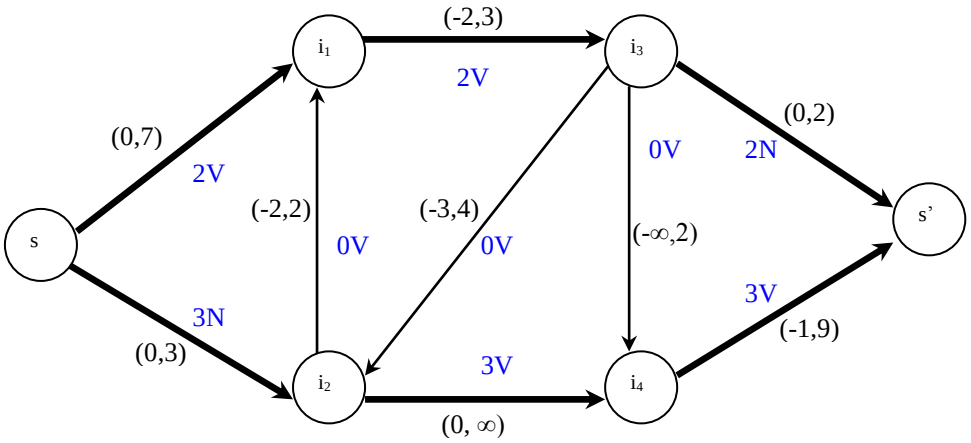
Capa 1: $s \to i_1 \to i_3 \to s'$ = $\max\ j \in c_1$: {(0,7),(-2,3),(0,2)} = 2

Capa 2: $s \to i_2 \to i_4 \to s'$ = $\max\ j \in c_2$: {(0,3),(0, ∞),(-1,9)} = 3

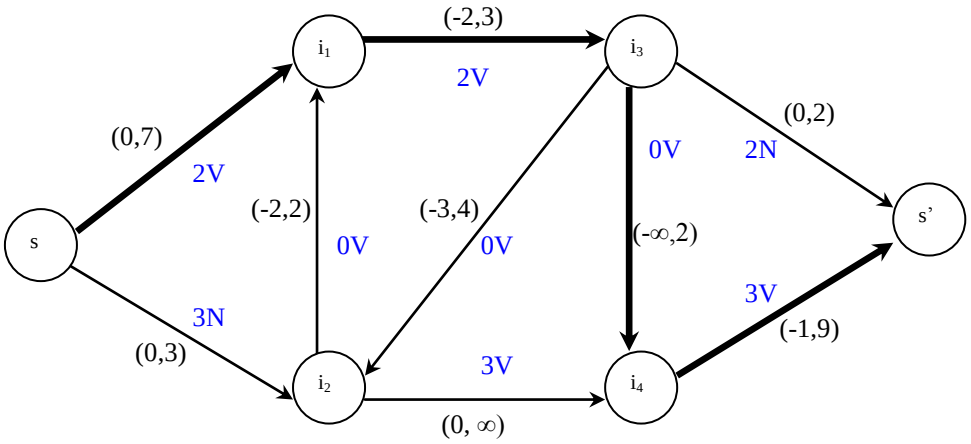
ITERACIÓN 1

Paso 1: Comenzar con un flujo de 2 $x(j)=2 \ \forall \ j \in C_1$ y un flujo de 3 $x(j)=3 \ \forall \ j \in C_2$ esto es posible porque todos los intervalos de capacidad de cada capa contienen al 2 y al 3 respectivamente, los arcos que no pertenecen a una capa, inician con un flujo de $0 \ \forall \ j \in R$.

Paso 2: Colorear



Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

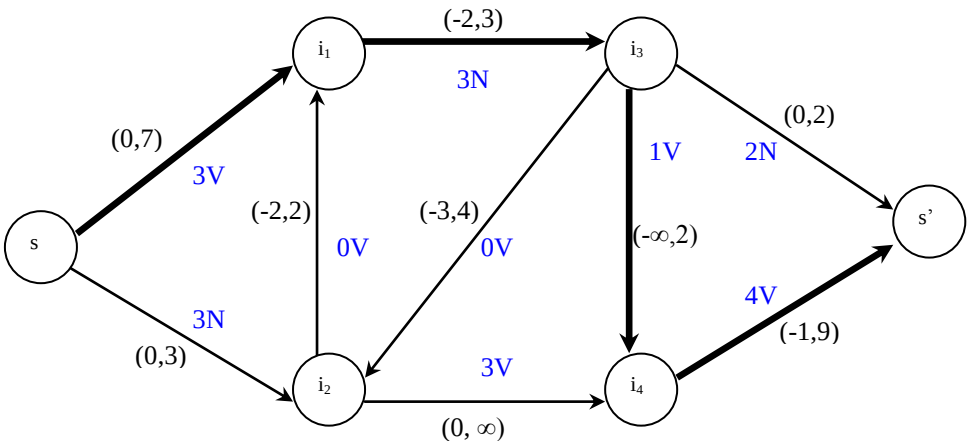


Paso 3.1: Calcular α

$P_1 : s \to i_1 \to i_3 \to i_4 \to s'$

α_1 = min {7-2,3-2,2-0,9-3}
= min {5,1,2,6}
= 1

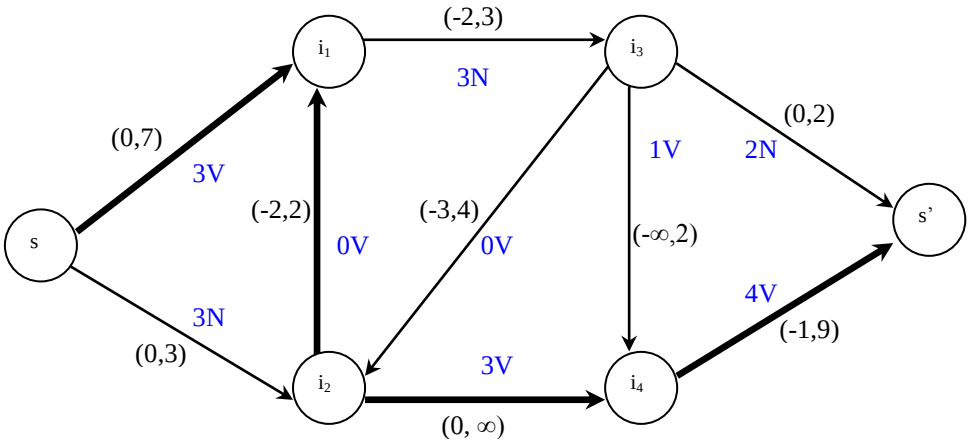
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 2

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

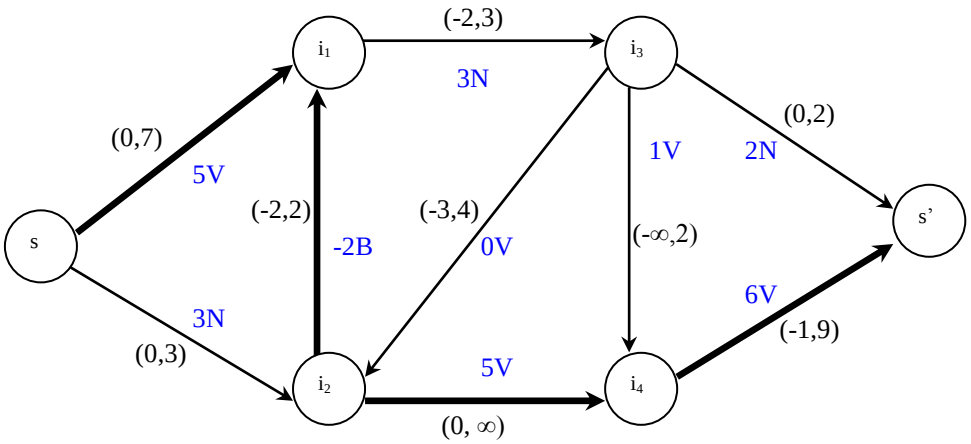


Paso 3.1: Calcular α

$P_2 : s \to i_1 \leftarrow i_2 \to i_4 \to s'$

α_2 = min {7-3,0-(-2),∞-3,9-4}
= min {4,2,∞,5}
= 2

Paso 3.2: Actualizar el flujo con α en la trayectoria.

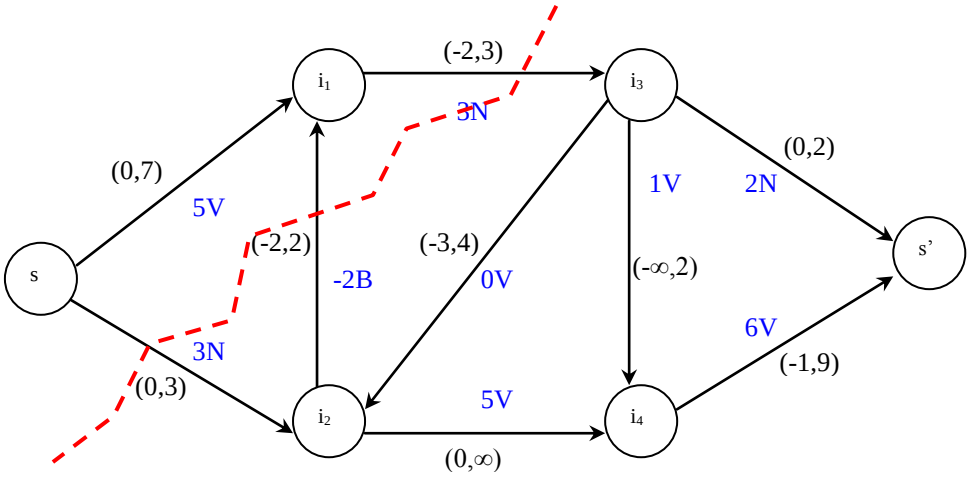


ITERACIÓN 3

Paso 2: Recolorear

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

Paso 3.2: No existe trayectoria compatible con la coloración, se determino un corte Q compatible con la coloración terminar.



Puesto que el flujo propuesto no proporciona otra trayectoria de flujo aumentante, sólo el corte $Q = [S, N/S]$ correspondiente a $S = \{s, i_1\}$, Así, el flujo x en esta etapa resuelve el problema de flujo máximo mientras Q resuelve el problema de corte mínimo. El flujo de s a s' es 8, y este es el mismo valor de $C^+(Q)$.

$S = \{s, i_1\}$ $\qquad Q^+ = \{(s, i_3), (s, i_2)\} = 3+3 = 6$

$Q^- = \{(i_2, i_1)\} = -2$

$y(s) = 5+3 = 8$

$y(i_1) = 3-5-(-2) = 0$ $\qquad Q = Q^+ - Q^- = 6 - (-2) = \mathbf{8}$

$\therefore \; y(s) = \mathbf{8}$

Una nueva forma de obtener el flujo máximo a partir de esta modificación es la siguiente:

$$Flujo \quad Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j)$$

Donde:

α_i : representa las cantidades de actualización de flujo de cada iteración.

$x(j)$: representa la cota máxima perteneciente a todos los arcos contenidos en cada capa la red.

n : representa el número de iteraciones.

m : representa la cantidad de capas de la red.

Para el ejemplo 3.2 se tendría:

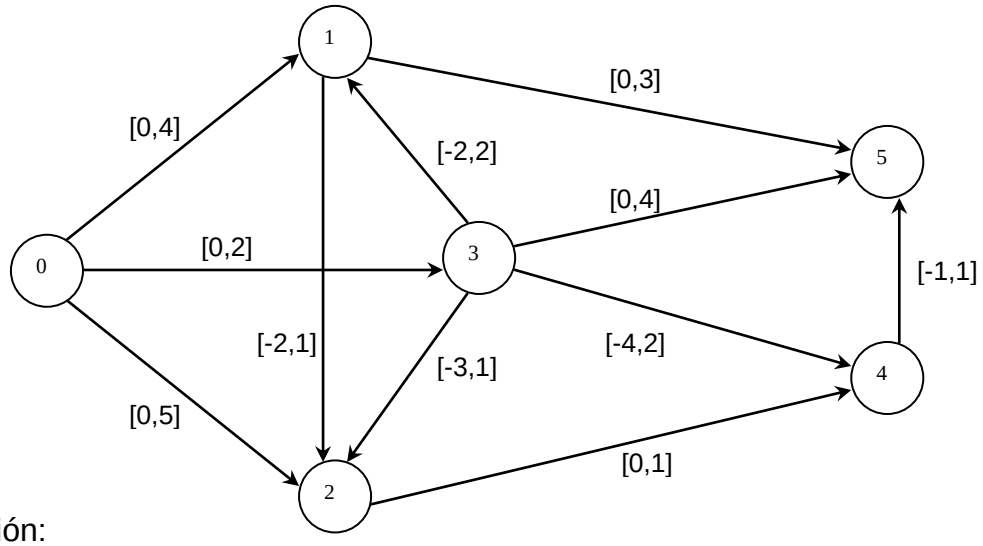
$$Flujo \quad Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j) = (1+2)+(2+3) = 3+5 = 8$$

Por lo tanto el algoritmo de Ford-Fulkerson mejorado resulta más eficiente pues reduce considerablemente el número de iteraciones del algoritmo original. Como se observó en el ejemplo 3.1 resuelto con el algoritmo original este requirió de **5 iteraciones** para llegar al óptimo mientras que el algoritmo modificado requirió de tan sólo **3 iteraciones**.

Cuando la red a estudiar crece, se vuelve más evidente la ventaja de utilizar el algoritmo de Ford-Fulkerson mejorado. Así como también cuando tenemos cantidades U grandes, donde U representaría que las capacidades en los arcos son enteras y finitas, el número de iteraciones se reduce considerablemente y el nuevo método no sufre de las anomalías que sufre el algoritmo original. Ver ahora un problema más grande.

Ejemplo 3.3

Primero utilizar el algoritmo original. En la siguiente red se busca el flujo máximo, comenzar con un flujo igual a cero $x(j)=0$ para toda $j \in R$ esto es posible porque todos los intervalos de capacidad contienen al cero.



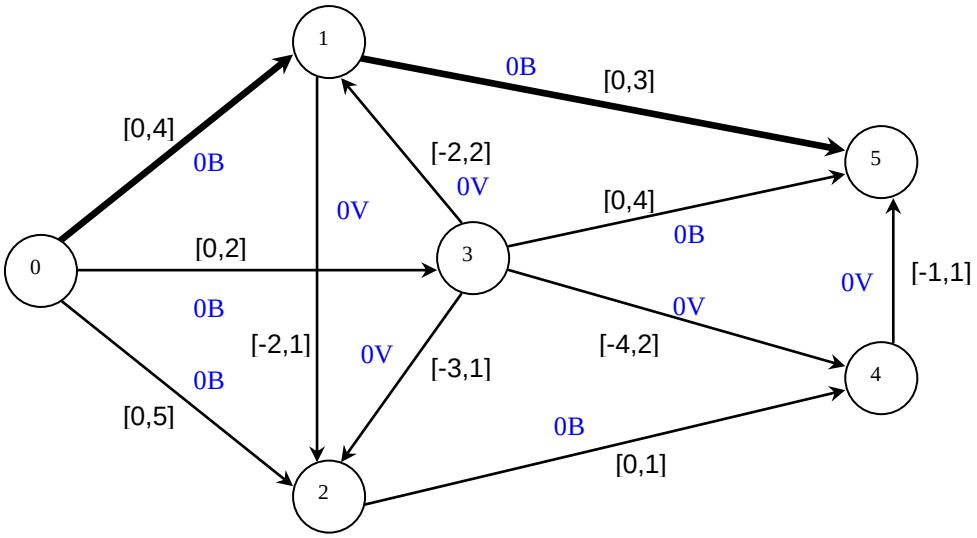
Solución:

ITERACIÓN 1

Paso 1: Comenzar con un flujo igual a 0 $x(j)=0 \ \forall \ j \in R$ esto es posible porque todos los intervalos de capacidad contienen al 0.

Paso 2: Colorear

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

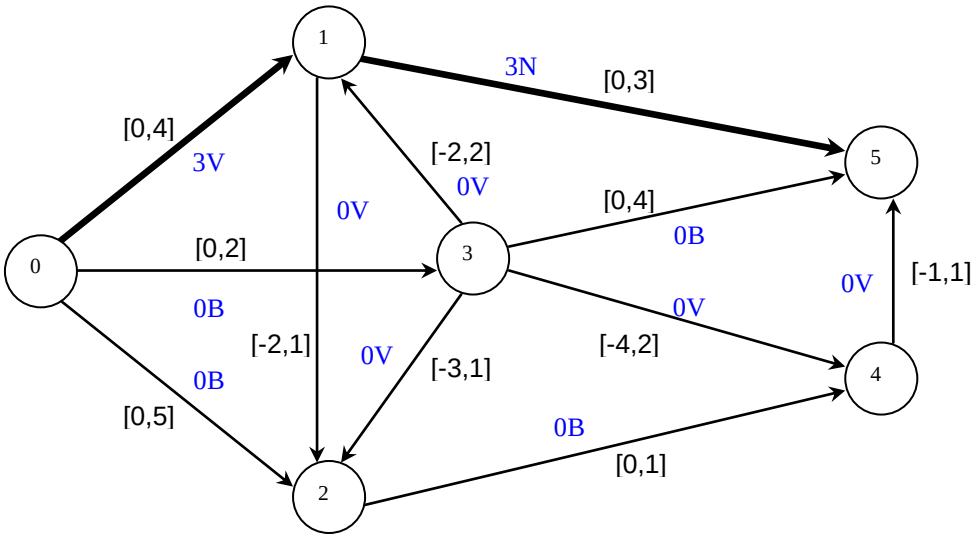


Paso 3.1: Calcular α

$P_1 : 0 \rightarrow 1 \rightarrow 5$

$\alpha_1 = \min \{4\text{-}0, 3\text{-}0\}$
$= \min\{4, 3\}$
$= 3$

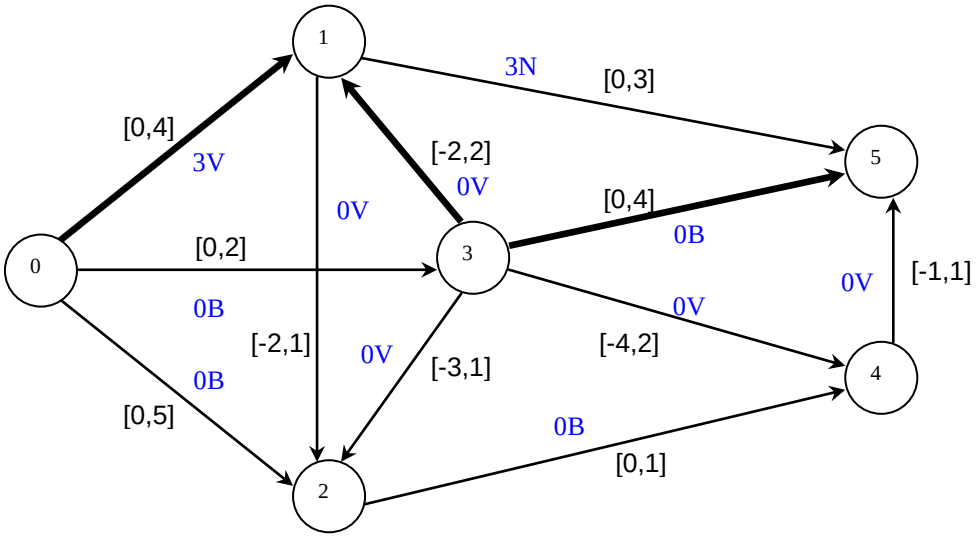
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 2

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

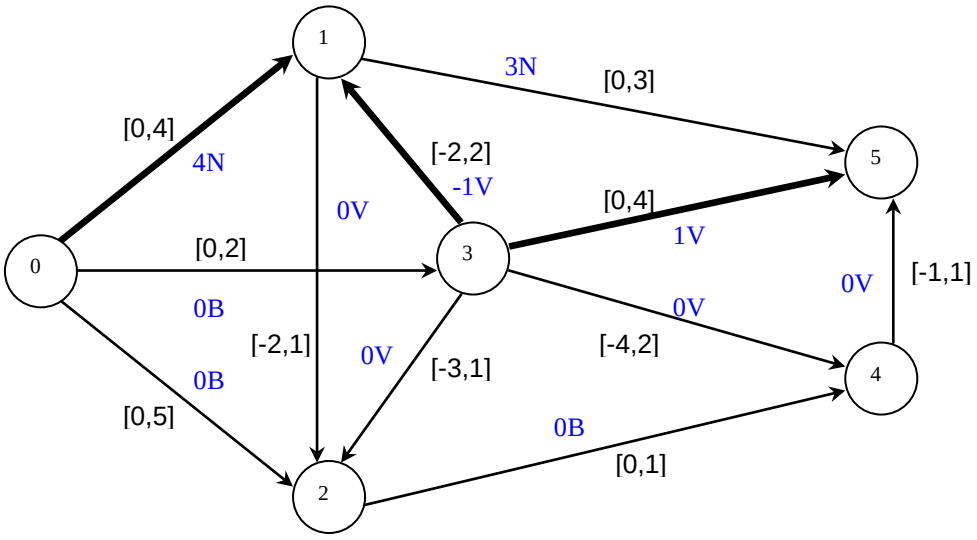


Paso 3.1: Calcular α

$P_2 : 0 \rightarrow 1 \leftarrow 3 \rightarrow 5$

$\alpha_2 = \min \{4\text{-}3,\ 0\text{-}(\text{-}2),\ 4\text{-}0\}$
$= \min \{1,\ 2,\ 4\}$
$= 1$

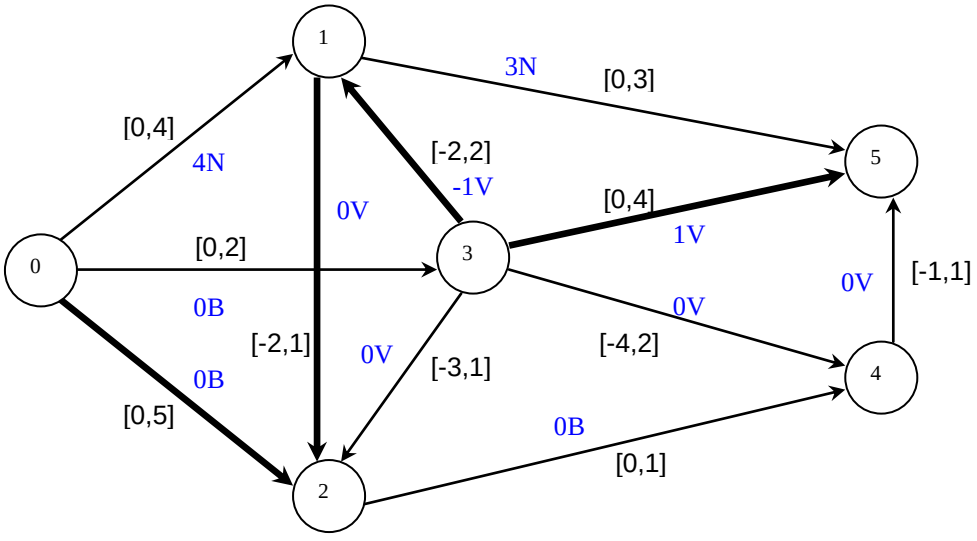
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 3

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

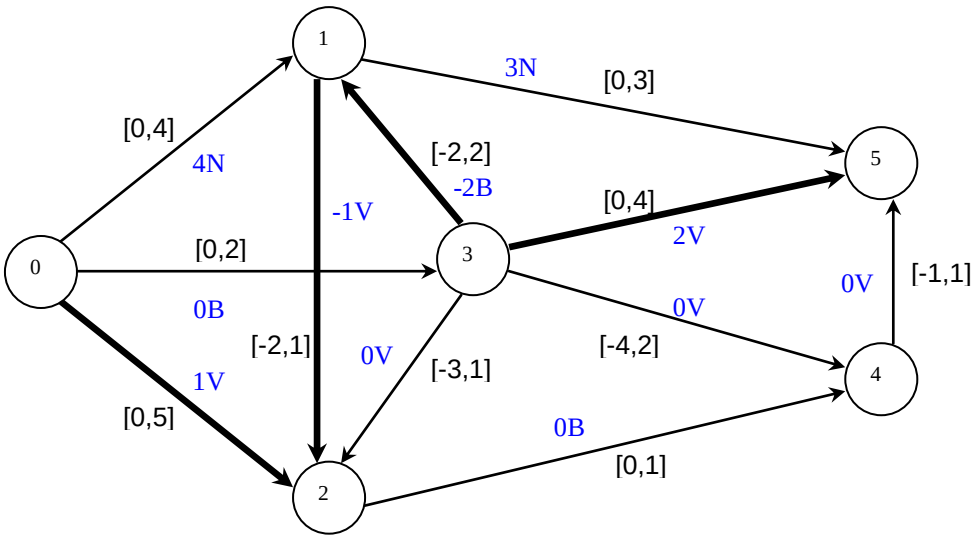


Paso 3.1: Calcular α

$P_3 : 0 \rightarrow 2 \leftarrow 1 \leftarrow 3 \rightarrow 5$

α_3 = min {5-0, 0-(-2), -1-(-2),4-1}
= min {5, 2, 1,3}
= 1

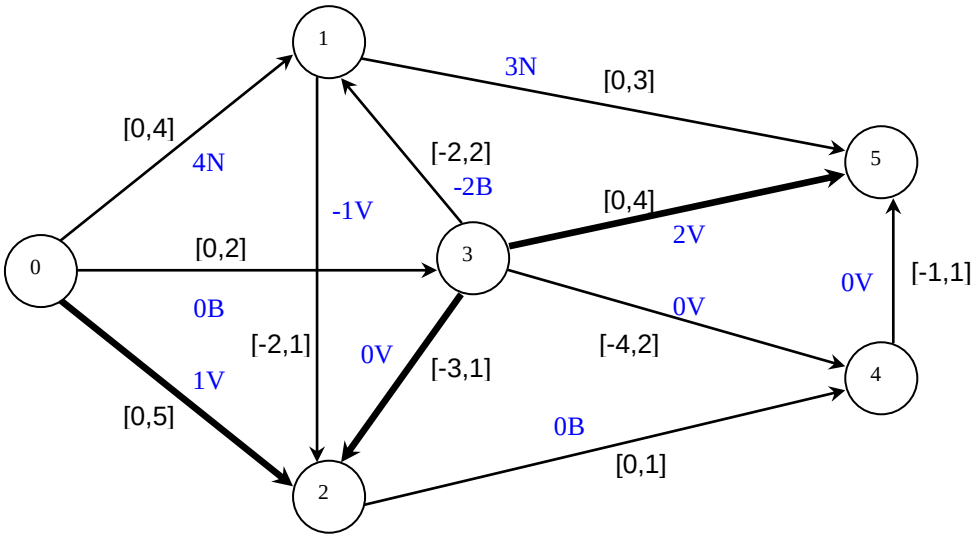
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 4

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

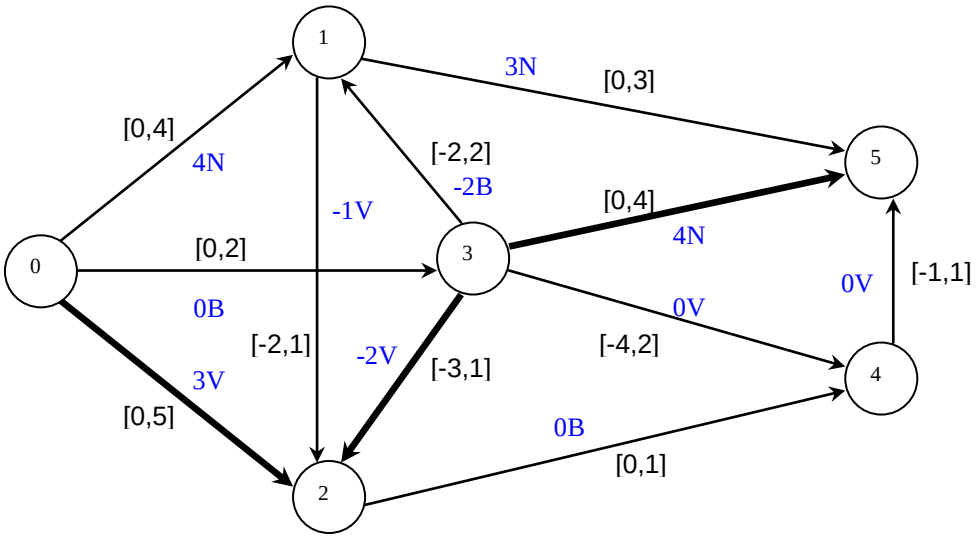


Paso 3.1: Calcular α

$P_4 : 0 \rightarrow 2 \leftarrow 3 \rightarrow 5$

$\alpha_4 = \min \{5\text{-}1, 0\text{-}(-3), 4\text{-}2\}$
$= \min \{4, 3, 2\}$
$= 2$

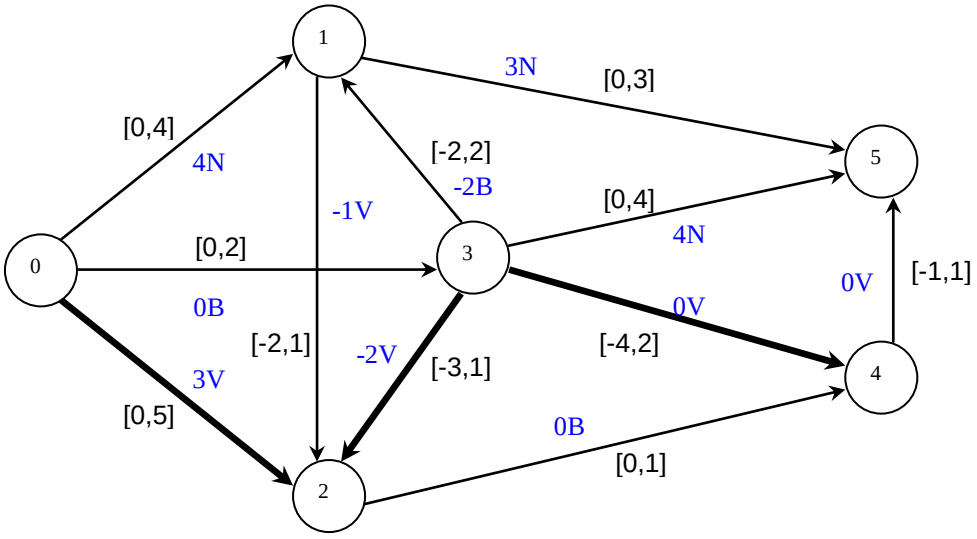
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 5

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

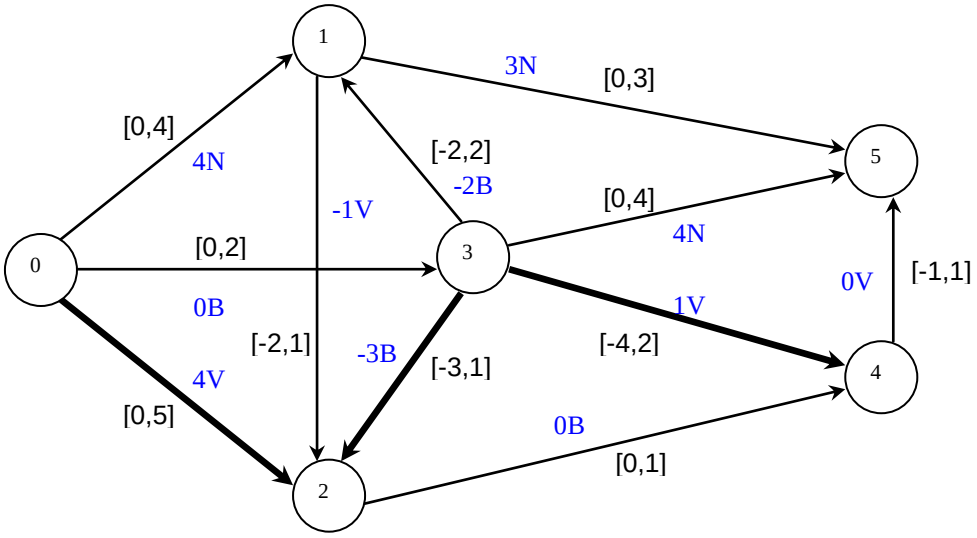


Paso 3.1: Calcular α

$P_5 : 0 \rightarrow 2 \leftarrow 3 \rightarrow 4$

$\alpha_5 = \min \{5\text{-}3, -2\text{-}(-3), 2\text{-}0\}$
$= \min \{2, 1, 2\}$
$= 1$

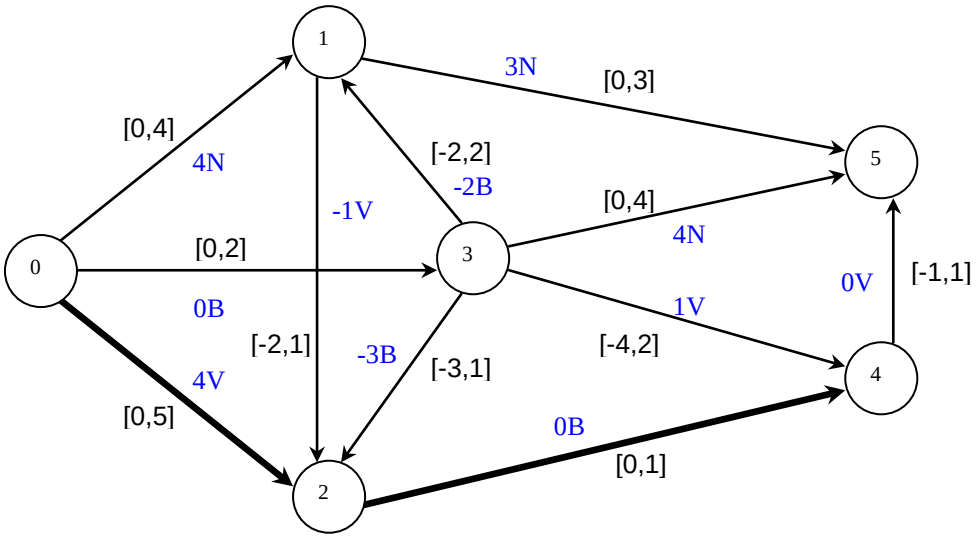
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 6

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

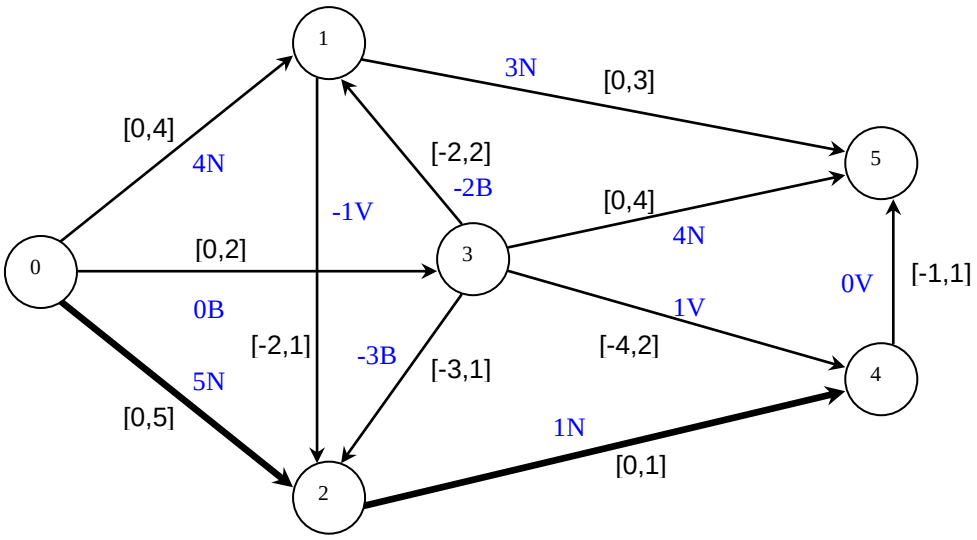


Paso 3.1: Calcular α

$P_6 : 0 \rightarrow 2 \rightarrow 4$

$\alpha_6 = \min \{5\text{-}4,\ 1\text{-}0\}$
$= \min \{1,1\}$
$= 1$

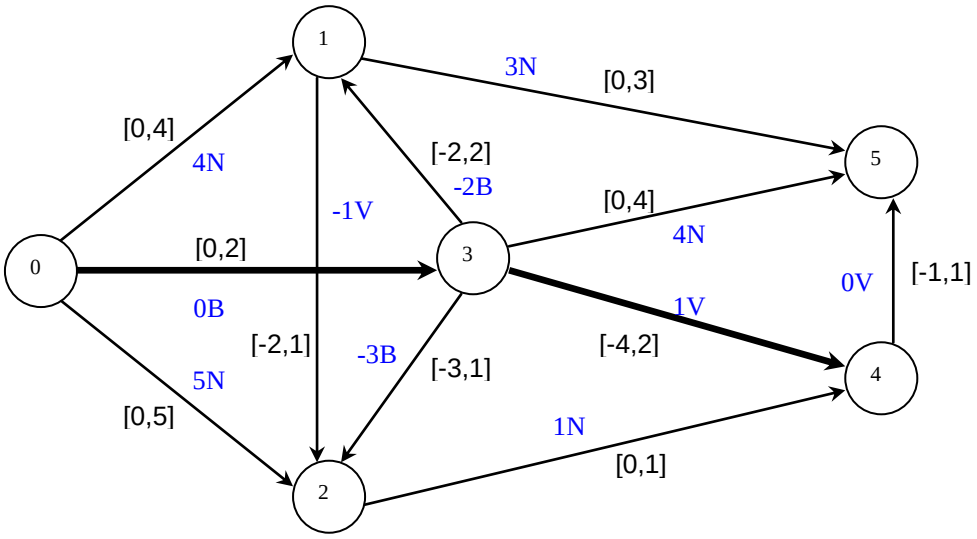
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 7

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

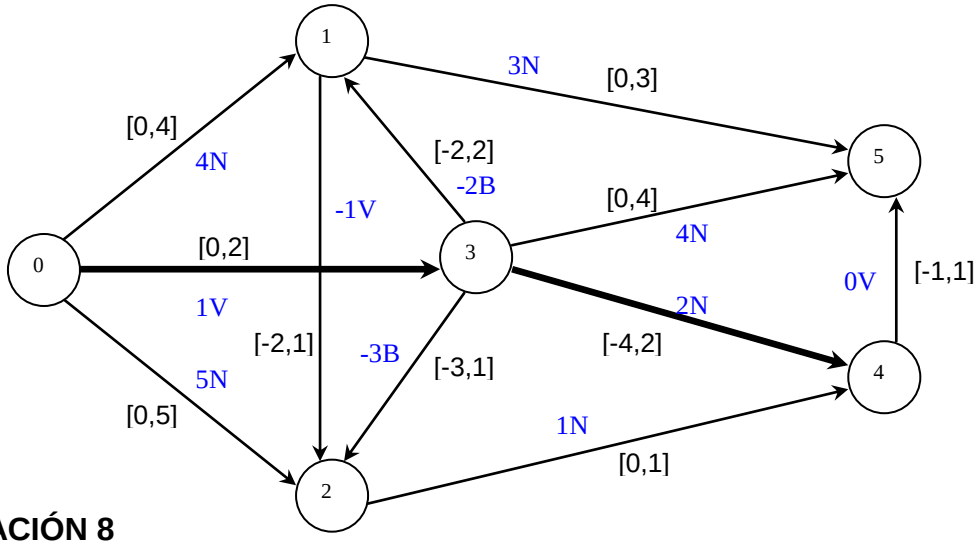


Paso 3.1: Calcular α

$P_7 : 0 \rightarrow 3 \rightarrow 4$

$\alpha_7 = \min \{2\text{-}0,\ 1\text{-}2\}$
$= \min \{2,1\}$
$= 1$

Paso 3.2: Actualizar el flujo con α en la trayectoria.

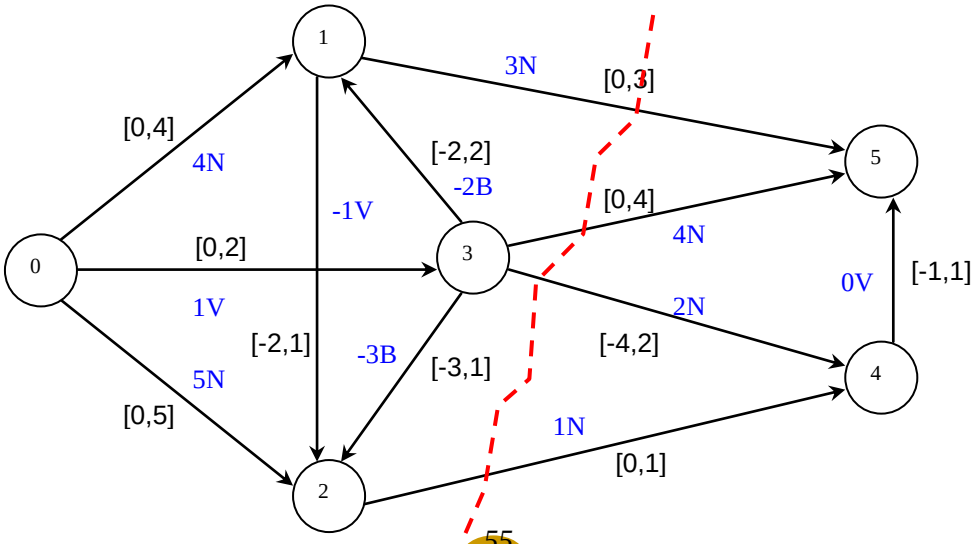


ITERACIÓN 8

Paso 2: Recolorear

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

Paso 3.2: No existe trayectoria compatible con la coloración, se determinó un corte Q compatible con la coloración por lo tanto el flujo encontrado es máximo.



Puesto que el flujo propuesto no proporciona otra trayectoria de flujo aumentante, sólo el corte $Q = \overline{S, N/S}$ correspondiente a $S = \{1,2,3\}$. Así, el flujo x en esta etapa resuelve el problema de flujo máximo mientras Q resuelve el problema de corte mínimo. El flujo de s a s' es 10 y este es el mismo valor de $C^+(Q)$.

$S = \{1,2,3\}$ $\qquad Q^+ = \{(1,5), (3,5), (3,4), (2,4)\} = 3+4+2+1=10$

$y(1) = 3\text{-}1\text{-}(-4\text{-}2) = 9$ $\qquad Q^- = \{\} = 0$

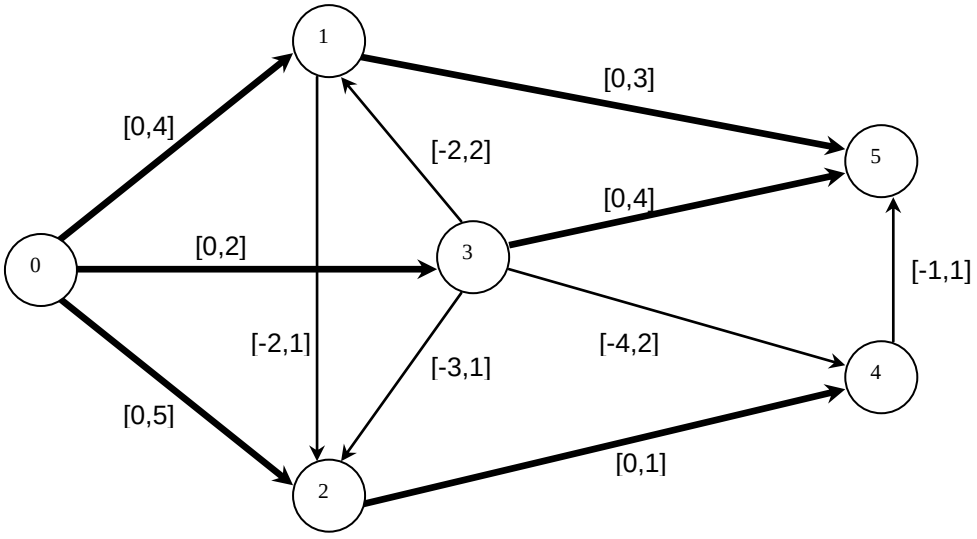
$y(2) = 1\text{-}(5\text{-}1\text{-}3) = 0$ $\qquad Q = Q^+ - Q^- = 10 - 0 = \mathbf{10}$

$y(3) = 4+2\text{-}2\text{-}3 = 1$

$\therefore\ y(s) = \mathbf{10}$

Ejemplo 3.4

Ahora se utilizara la modificación propuesta para resolver el ejemplo 3.3, se inicia al determinar un flujo inicial que es igual a la cota máxima contenida en los arcos pertenecientes a cada capa de la red $\forall\ j \in C_R$. Comenzando con un flujo inicial de $x(j) = 3$ para la capa $1\ \forall\ j \in C_1$, un flujo de $x(j) = 2$ para la capa $2\ \forall\ j \in C_2$ y un flujo de $x(j) = 1$ para la capa $3\ \forall\ j \in C_3$, esto es posible porque todos los intervalos de capacidad contienen al tres, dos y al uno respectivamente, los arcos que no pertenecen a una capa comienzan con un flujo igual a la cota mínima perteneciente a todos los arcos de la red $\forall\ j \in R$ en este y la mayoría de los casos es 0. Solución:



Capa 1: $0 \to 1 \to 5$ = $\max\ j \in c_1$: {(0,4),(0,3)} = 3

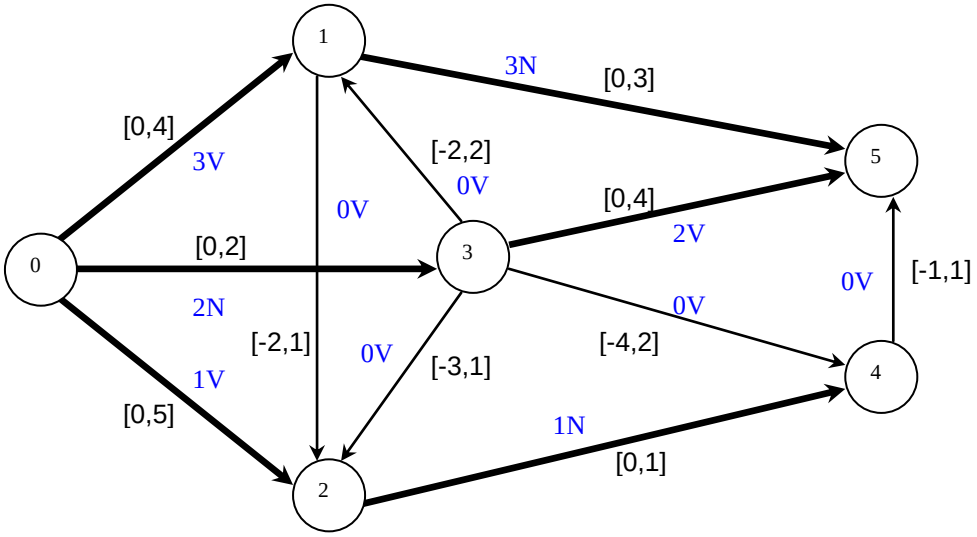
Capa 2: $0 \to 3 \to 5$ = $\max\ j \in c_2$: {(0,2),(0, 4)} = 2

Capa 3: $0 \to 2 \to 4$ = $\max\ j \in c_3$: {(0,5),(0,1)} = 1

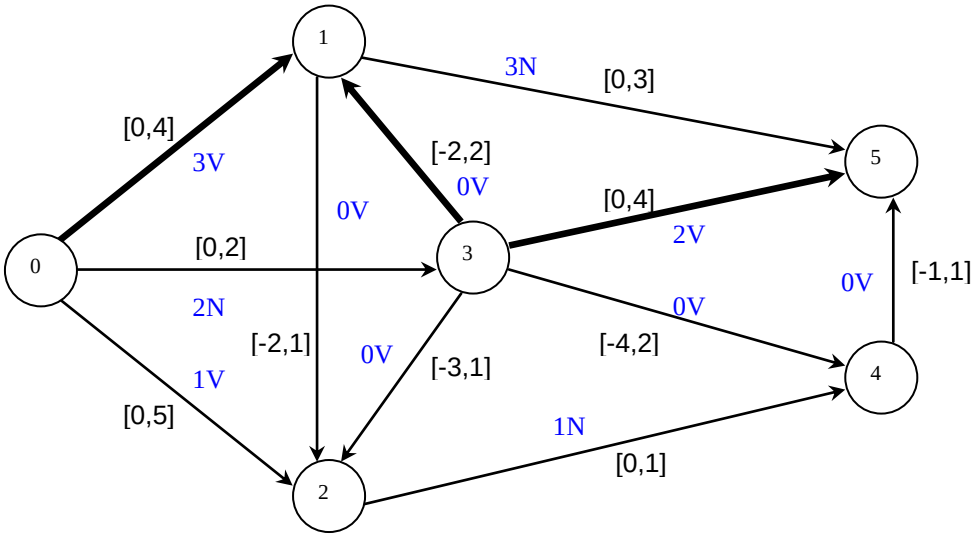
ITERACIÓN 1

Paso 1: Comenzar con un flujo de 3 $x(j)=3\ \forall\ j\in C_1$, un flujo de 2 $x(j)=2\ \forall\ j\in C_2$ y un flujo de 1 $x(j)=1\ \forall\ j\in C_3$ esto es posible porque todos los intervalos de capacidad de cada capa contienen al $3,2$ y 1 respectivamente, los arcos que no pertenecen a una capa, inician con un flujo de $0\ \forall\ j\in R$.

Paso 2: Colorear



Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

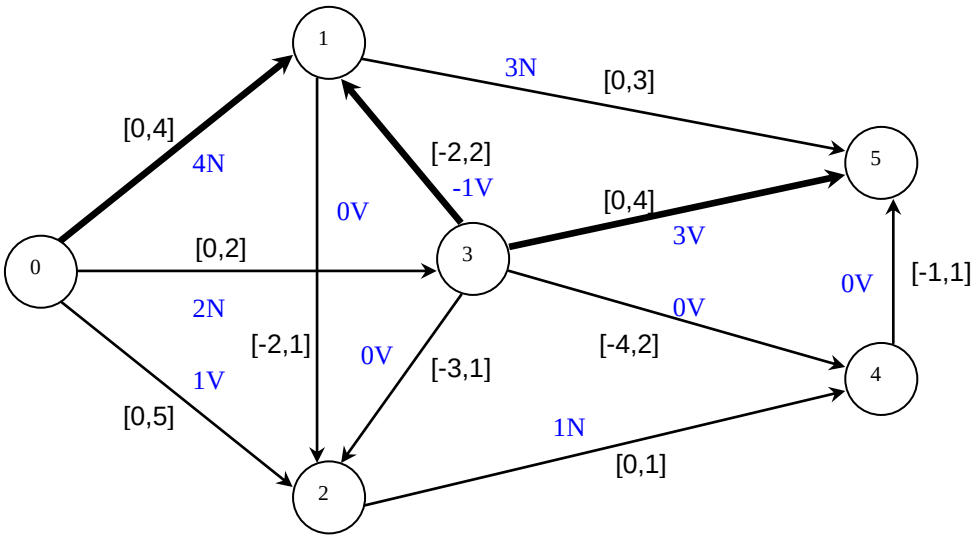


Paso 3.1: Calcular α

$P_1 : 0 \rightarrow 1 \leftarrow 3 \rightarrow 5$

α_1 = min {4-3,0-(-2),4-2}
= min {1,2,2}
= 1

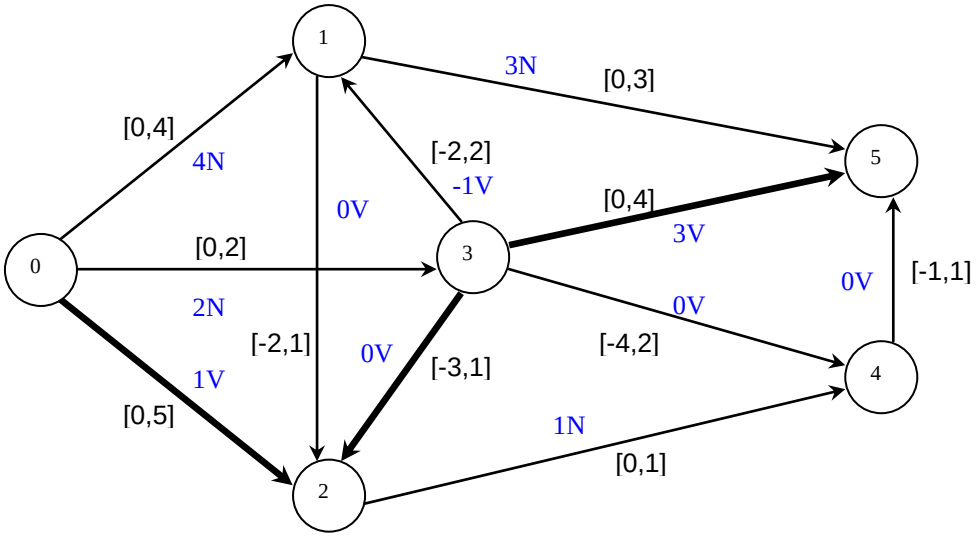
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 2

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

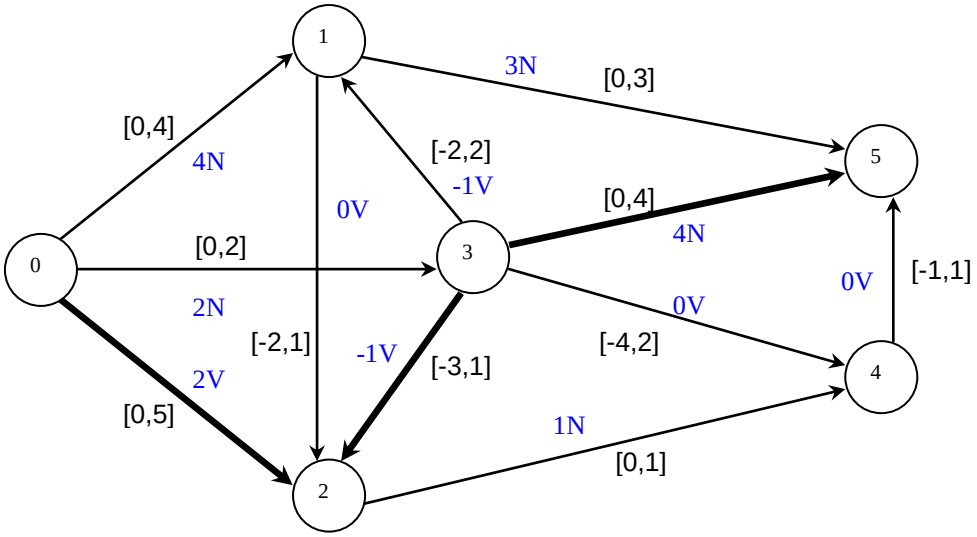


Paso 3.1: Calcular α

$P_2 : 0 \rightarrow 2 \leftarrow 3 \rightarrow 5$

$\alpha_2 = \min \{5\text{-}1, 0\text{-}(-3), 4\text{-}3\}$
$= \min \{4,3,1\}$
$= 1$

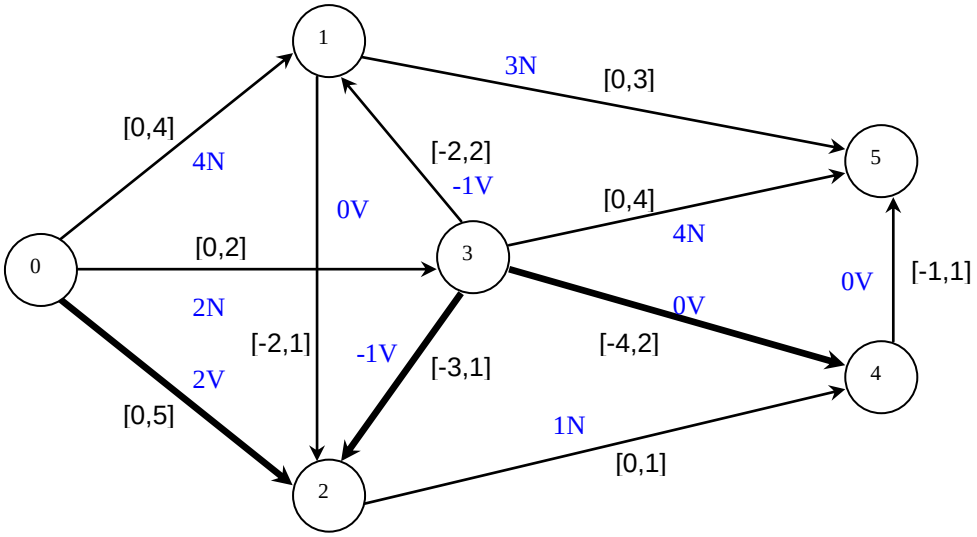
Paso 3.2: Actualizar el flujo con α en la trayectoria.



ITERACIÓN 3

Paso 2: Recolorear la trayectoria.

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

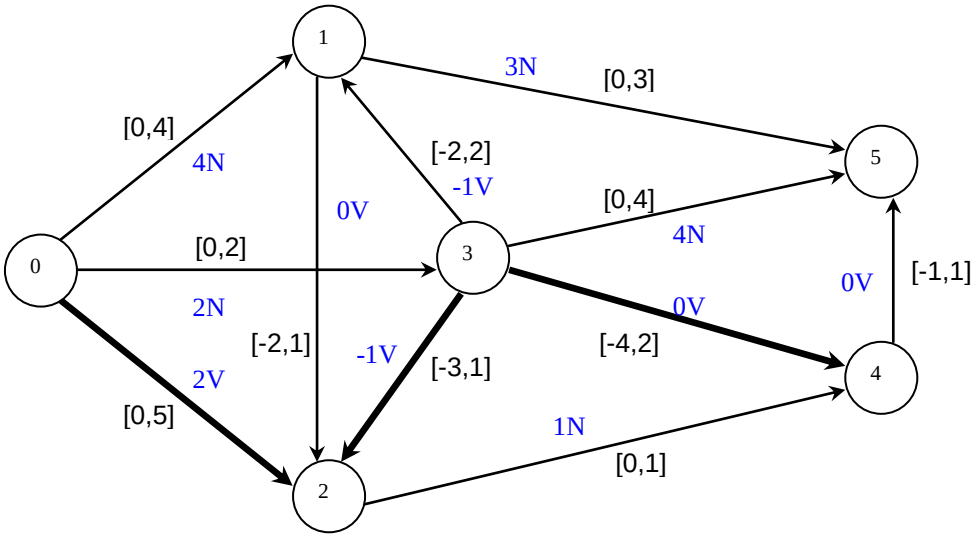


Paso 3.1: Calcular α

$P_3 : 0 \rightarrow 2 \leftarrow 3 \rightarrow 4$

α_3 = min {5-2,-1-(-3),2-0}
= min {3,2,2}
= 2

Paso 3.2: Actualizar el flujo con α en la trayectoria.

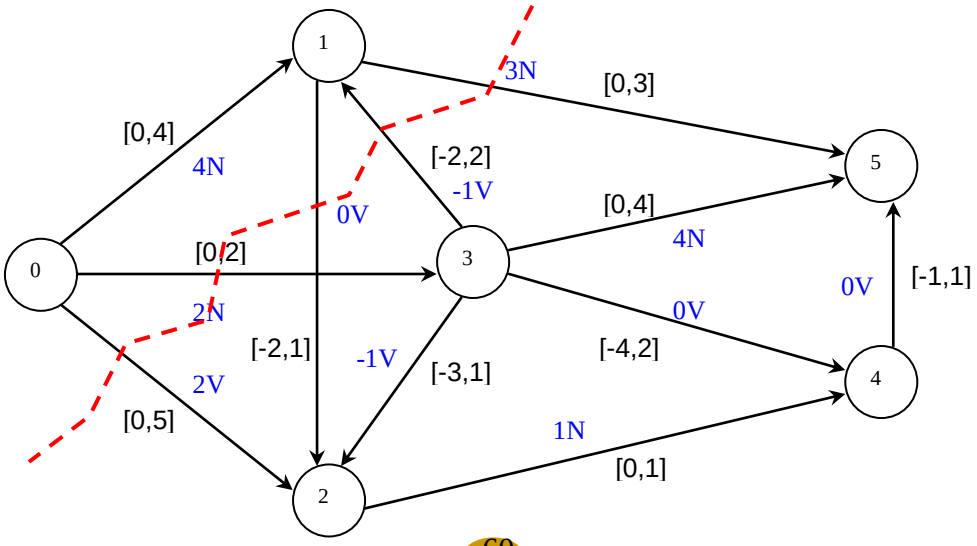


ITERACIÓN 4

Paso 2: Recolorear

Paso 3: Buscar una trayectoria compatible con la coloración que vaya de s a s'.

Paso 3.2: No existe trayectoria compatible con la coloración, se determinó un corte Q compatible con la coloración por lo tanto el flujo encontrado es máximo.



$$Flujo \quad Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j) = (1+1+2)+(3+2+1) = 4+6 = 10$$

Aplicando el Teorema de flujo máximo corte mínimo se tiene:

$S = 1,2,3$

$y(1) = 3\text{-}1\text{-}(-4\text{-}2) = 9$

$y(2) = 1\text{-}(5\text{-}1\text{-}3) = 0$

$y(3) = 4+2\text{-}2\text{-}3 = 1$

$\therefore \; y(s) = \mathbf{10}$

$Q^+ = \{(1,5), (3,5), (3,4), (2,4)\} = 3+4+2+1 = 10$

$Q^- = \{\} = 0$

$Q = Q^+ - Q^- = 10 - 0 = \mathbf{10}$

Como se puede observa el número de iteraciones, en este ejemplo con un mayor número de nodos se redujo considerablemente de **8 iteraciones** que requiere el algoritmo original, a sólo **4 iteraciones** que requiere el algoritmo mejorado, el número de iteraciones se redujo a la mitad, y para problemas más grandes se nota la gran diferencia que existe al utilizar el método tradicional y el mejorado.

Como se puede observar, la propuesta de proponer un flujo inicial es muy eficiente en cuanto al número de iteraciones que se requiere para llegar a una solución óptima o aproximada. Pero esta modificación se extendió para los casos particulares del algoritmo Ford-Fulkerson, puesto que habrá ocasiones que el flujo que propongamos no podrá cumplir con las restricciones de los arcos o de las divergencias en los nodos y es cuando haremos uso de estas modificaciones para obtener una solución al problema de distribución de flujo factible. Enseguida se muestran dichas modificaciones con un flujo inicial propuesto el cual las vuelve más eficientes y prácticas, además se ilustrara cada algoritmo con ejemplos para que las ideas sean claras.

Distribución factible mejorado es un caso particular del algoritmo de Ford y Fulkerson que se realiza suponiendo que el flujo en la red es factible en los arcos, además suponiendo que tenemos un flujo determinado en nuestro nodo fuente. El algoritmo consiste en determinar inicialmente un flujo factible con respecto a las restricciones de las capacidades de los arcos y posiblemente no factible con respecto a las restricciones de oferta en los nodos, tomando en cuenta que se deben utilizar el menor número de arcos posibles cuando se pasa de una etapa a otra. Esto con el fin de que se generen inicialmente trayectorias de color negro y el algoritmo termine en un menor número de iteraciones.

Primer paso se determina un flujo factible x tomando en cuenta que se debe sacar como flujo inicial la cantidad que se tiene en nuestro nodo fuente y llevarla al nodo sumidero y esto sin violar las restricciones de capacidad de los arcos y tratar de violar lo menos que se pueda las restricciones de oferta de los nodos.

Segundo paso se determina dos conjuntos $N^+ \text{y } N^-$ compuestos por los nodos cuya divergencia < demanda y cuya divergencia > demanda respectivamente $N^+ = \{i \in N \mid y(i) < b(i)\}$, $N^- = \{i \in N \mid y(i) > b(i)\}$ Si el conjunto de $N^+ \text{y } N^-$ son igual a vació, entonces el flujo x propuesto sería la solución deseada, en caso contrario, se colorean los arcos de igual forma que en el algoritmo original.

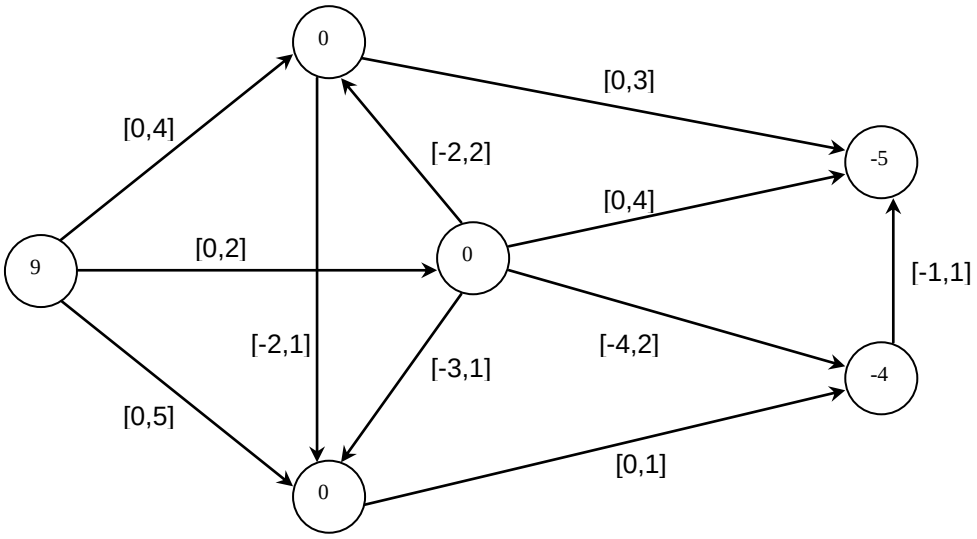
Tercer paso se aplica el algoritmo de enrutamiento a la red como el algoritmo original, esto con la finalidad de determinar una trayectoria compatible con la coloración que va $P : N^+ \to N^-$.

$$\alpha = \min \begin{Bmatrix} c^+(j) - x(j) & j \in P^+ \\ x(j) - c^-(j) & j \in P^- \\ b(i) - y(i) & i \text{ nodo inicial de } P \text{ en } N^+ \\ y(i) - b(i) & i \text{ nodo terminal de } P \text{ en } N^- \end{Bmatrix}$$

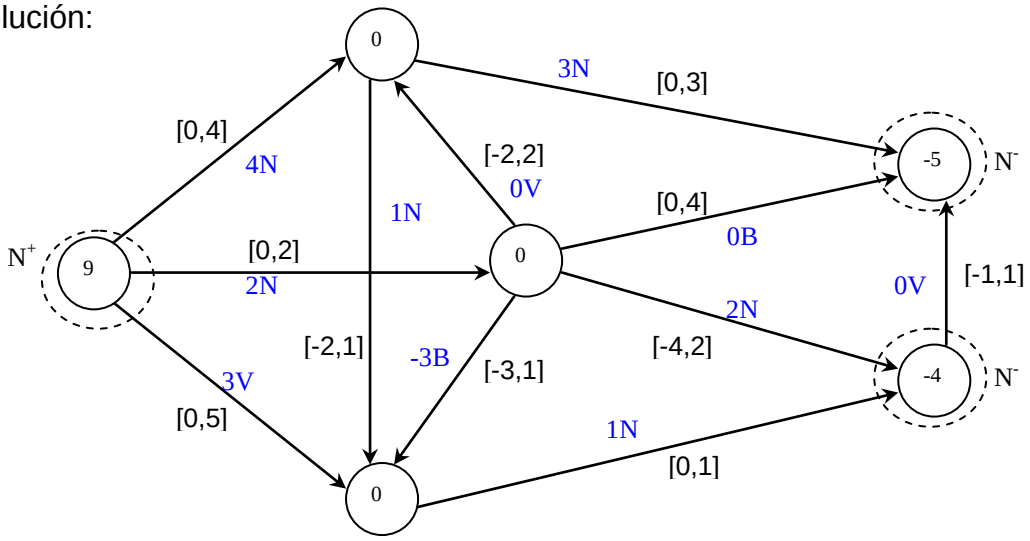
Hacemos $x' = x + \alpha e_p$ y volvemos a determinar nuestros conjuntos $N^+ \text{y } N^-$.Si se determina un corte $Q = (S, N/S)$ compatible con la coloración terminar, no existe solución al problema. Y terminamos cuando los conjuntos de $N^+ \text{y } N^-$ sean vacíos, puesto que se cumple que $y(i) = b(i)$ para toda i y el flujo que representa es máximo.

Ejemplo 3.5

Ahora emplear la modificación propuesta para resolver el problema de la siguiente red donde se busca la distribución del flujo factible, comenzar con un flujo propuesto.



Solución:

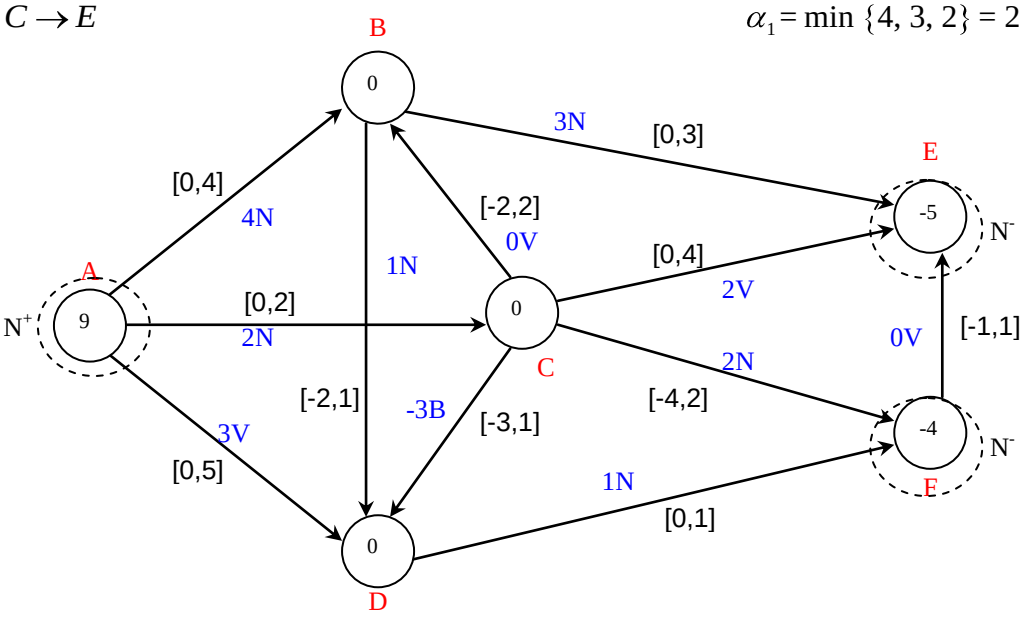


Usando el método de distribución factible mejorado, iniciar dando un flujo factible en los arcos, tratamos de violar lo menos que se pueda las divergencias de cada nodo. Una vez hecho esto, se comparan con sus demandas respectivas. Si se cumple la igualdad paramos, si no, continuamos con el algoritmo.

i	A	B	C	D	E	F
y(i)	9	0	-3	0	-3	-3
b(i)	9	0	0	0	-5	-4

$N^+ = \{A, \}\ N^- = \{E, F$

$P: C \to E$ $\qquad \alpha_1 = \min\{4, 3, 2\} = 2$

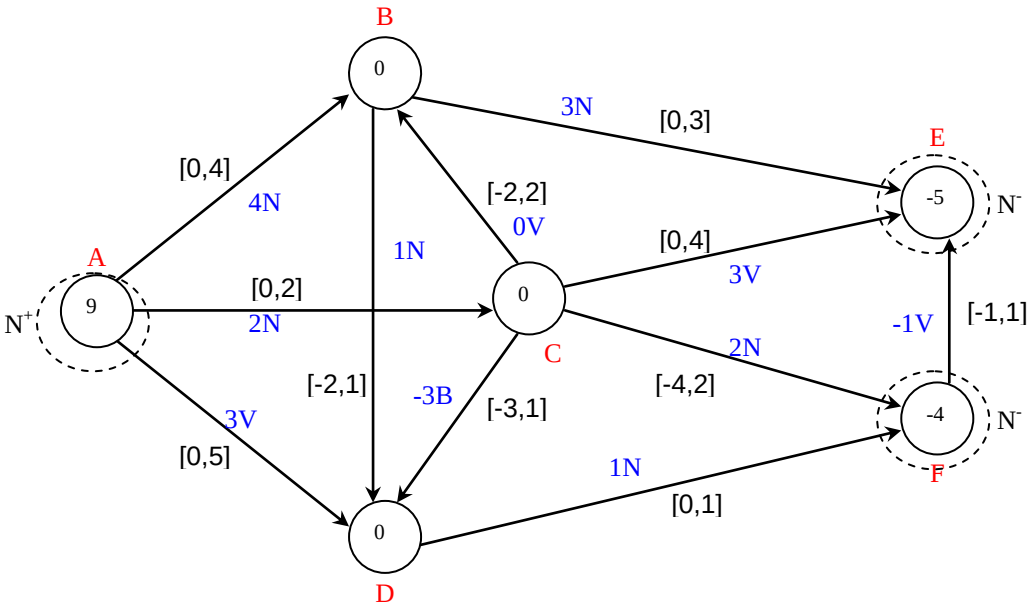


Después, se obtienen las divergencias de cada nodo y se comparan con sus demandas respectivas. Si se cumple la igualdad se para, si no se continua con el algoritmo.

i	A	B	C	D	E	F
y(i)	9	0	-1	0	-5	-3
b(i)	9	0	0	0	-5	-4

$N^+ = \{\}, N^- = \{\}$

$P: C \rightarrow E \rightarrow F$ $\qquad \alpha_2 = \min\{2, 1, 1, 1\} = 1$



i	A	B	C	D	E	F
y(i)	9	0	0	0	-5	-4
b(i)	9	0	0	0	-5	-4

Después, se obtiene las divergencias de cada nodo y se comparan con sus demandas respectivas. A aquí se cumple la igualdad así que se para.

$$N^+ = \phi \quad N^- = \phi \qquad \therefore \; y(i) = b(i) \quad \forall \; i \in A$$

En este ejemplo no se maneja el método distribución factible tal y como se utiliza, porque el número de iteraciones es realmente grande, puesto que el algoritmo original hubiera iniciado con un flujo inicial de 1 en todos los arcos por tal motivo sólo se utilizó, el método distribución factible mejorado para asi reducir las iteraciones.

El flujo que se propone, cumple con las restricciones de los arcos y de las divergencias a la vez, tal vez sea más tardado proponer el flujo pero ahorraría más iteraciones, y por lo tanto, en este caso el flujo propuesto es óptimo.

$$N^+ = \phi \quad N^- = \phi \qquad \therefore \; y(i) = b(i) \quad \forall \; i \in A$$

Rectificación de flujo mejorado Este es otro caso particular del algoritmo de Ford y Fulkerson, que inicia suponiendo que el flujo en la red es factible en nodos, además, suponiendo que se tiene un flujo determinado en nuestro nodo fuente. El algoritmo consiste en determinar inicialmente un flujo factible con respecto a las restricciones de oferta en los nodos y posiblemente no factible con respecto a las restricciones de capacidades sobre los arcos, el cual toma en cuenta que se debe utilizar el menor número de arcos posibles cuando se pasa de una etapa a otra. Esto con el fin de que se generen inicialmente trayectorias de color negro y el algoritmo termine en un menor número de iteraciones.

Primer paso se determina un flujo factible x con respecto a las restricciones de divergencia, es decir, donde la divergencia de $x = b$, y tomando el menor número de arcos posibles.

Segundo paso se definen dos conjuntos de arcos A^+ y A^- en los cuales se pondrían los arcos que sobrepasan las capacidad del arco por arriba y por abajo respectivamente $A^+ = \{j \in A, x(j) > c^+(j)\}$, $A^- = \{j \in A, x(j) < c^-(j)\}$. Si el conjunto de A^+ y A^- es igual a vació, entonces el flujo x propuesto sería la solución deseada. En caso contrario, si existe $j \in A^+$ o $j \in A^-$, se colorean los arcos de la red de igual forma que en el algoritmo original.

Tercer paso se aplica el lema de Minty el cual consiste en determinar un circuito elemental P compatible con la coloración, que contenga a j, entonces se determina:

$$\alpha = \min \begin{Bmatrix} c^+(j) - x(j) & j \in P^+ \\ x(j) - c^-(j) & j \in P^- \\ \bar{\alpha} & \end{Bmatrix}$$

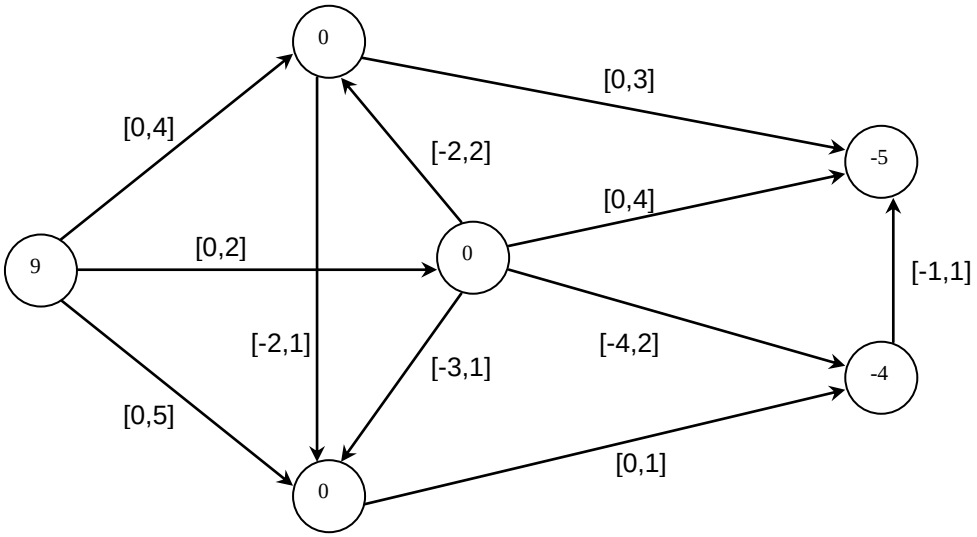
Donde

$$\bar{\alpha} = \min \begin{Bmatrix} c^+(j) - x(j) & j \in A^- \\ x(j) - c^-(j) & j \in A^+ \end{Bmatrix}$$

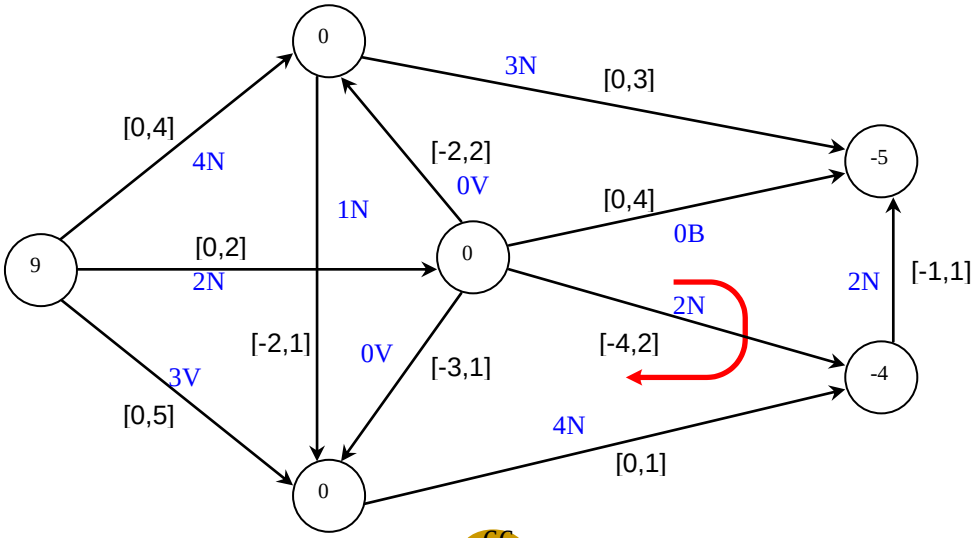
Se hace $x' = x + \alpha e_p$ y se vuelve a determinar los conjuntos de arcos A^+ y A^-. Si se determina un corte $Q = [S, N/S]$ compatible con la coloración que contenga a j terminar, ya que no existe solución al problema.
Y se termina cuando los conjunto de arcos A^+ y A^- sean vacíos, puesto se cumple que el flujo es factible en arcos y dado que cumple con las restricciones de divergencias el flujo es el óptimo y es máximo.

Ejemplo 3.6

Utilizar el mismo ejemplo anterior pero ahora emplear el método de rectificación de flujo mejorado para resolver el problema de la siguiente red, donde se busca el flujo factible máximo, comenzar con un flujo propuesto.



Solución: se inicia dando un flujo factible con respecto a las cantidades de divergencias. Y tratar de respetar las restricciones en arcos.



Una vez hecho esto, las comparamos con sus demandas respectivas. Si se cumple la igualdad parar, si no, continuar con el algoritmo. En este caso se genera un circuito compatible con la coloración que contiene los arcos pertenecientes a A^+ y A^-.

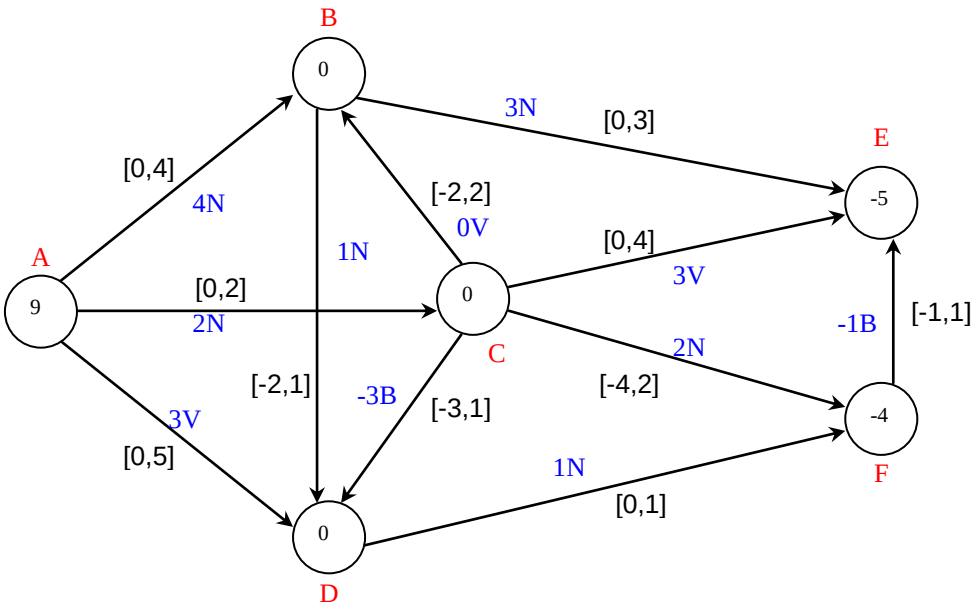
$A^+ = \{(i_d, i_f), (i_f, i_e)\}$

$P_1 : i_d \leftarrow i_c \rightarrow i_e \leftarrow i_f \leftarrow i_d$

$\alpha_1 = \min \{0\text{-}(-3), 4, \alpha_1, \alpha_2\} = 3$

$\alpha_1 = \min \{3, 4, 2\text{-}(-1), 4\} = 3$

$\alpha_1 = \min \{3, 4, 3, 4\} = 3$



$\therefore A^+ = \phi$ Por lo que el flujo definido es factible.

Como se puede ver, si se inicia con un flujo factible predeterminado, el algoritmo se vuelve más eficiente.

El algoritmo de Ford y Fulkerson que en la actualidad se está manejando, es el algoritmo de Edmonds-Karp, la cual es tan simple como utilizar búsqueda primero en anchura (BFS) para buscar el camino de aumento en el algoritmo de Ford-Fulkerson. Puesto que el algoritmo de Ford-Fulkerson de etiquetado es un algoritmo que ha sido muy estudiado y que realmente ya fue modificado, pero dado que aún presenta las anomalías del algoritmo original, se proporcionará la metodología para evitarlas.

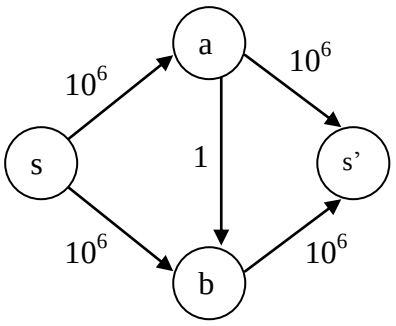
Además, con el tiempo se han generado varios algoritmos de etiquetado que cada vez son más eficientes en cuanto a la complejidad computacional.

Algoritmo de etiquetado mejorado para mejorar la complejidad respecto a que el algoritmo corra en tiempo pseudopolinomial. Se determina un flujo inicial para cada capa de la red que debe considerar el criterio de la cota máxima perteneciente a todos los arcos en cada capa a analizar, este flujo saldrá de nuestro nodo fuente por cada arco perteneciente a una capa y los arcos que no pertenezcan a una capa, iniciarán con un flujo igual a cero, que es equivalente a utilizar el criterio de la cota mínima perteneciente a todos los arcos de la red, si ésta no es cero, se tiene que cumplir la divergencia entre nodos.

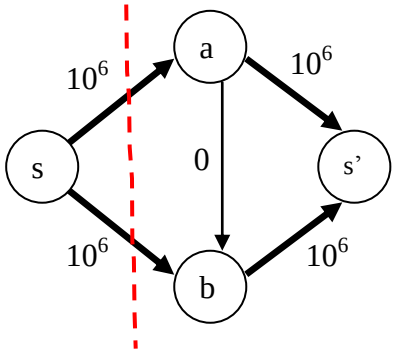
A continuación se describe un ejemplo de la anomalía principal del algoritmo original:

El algoritmo de etiquetado es posiblemente el algoritmo más simple para la resolución del problema de flujo máximo. En la práctica, el algoritmo trabaja razonablemente bien. Sin embargo, la cota del peor de los casos sobre el número de iteraciones no es enteramente satisfactoria para valores de U grandes. Por ejemplo, si $U = 2^n$, la cota seria exponencial en el número de nodos. Más aún, el algoritmo puede, de hecho, realizar muchas iteraciones como en el siguiente ejemplo.

En esta red, el algoritmo puede seleccionar los caminos incrementales s-a-b-s' y s-b-a-s' alternativamente 10^6 Veces, enviando cada vez una unidad de flujo a lo largo del correspondiente camino incremental.



Esto se podría evitar si no se utiliza el criterio de la mínima cota y se inicia con un flujo x, utilizando el criterio de la cota máxima perteneciente a todos los arcos contenidos en cada capa de la red y el criterio de la mínima cota para todos los arcos que no pertenecen a una capa en la red, para iniciar con un flujo de 10^6, en cada capa y un flujo de 0 en los arcos que no pertenecen a una capa. El problema se resolvería sin utilizar ninguna iteración puesto que el flujo propuesto es óptimo.



Capa 1: $s \to a \to s'$ = $\max\ j \in c_1 : \{(0,10^6),(0,10^6)\} = 10^6$
Capa 2: $s \to b \to s'$ = $\max\ j \in c_2 : \{(0,10^6),(0,10^6)\} = 10^6$

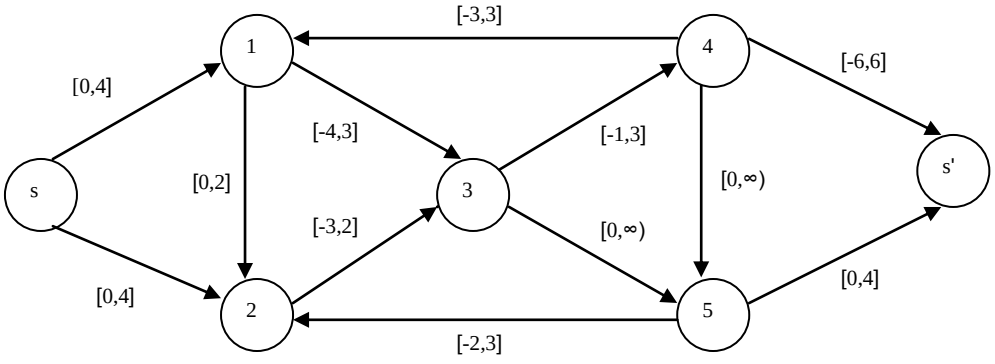
$$Flujo\quad Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j) = (0) + (10^6 + 10^6) = 2x10^6$$

Otra anomalía que presenta el algoritmo de etiquetado es cuando se presentan capacidades irracionales en los arcos pues cuando sucede esto, el algoritmo no converge. La metodología para eliminarla es tomar una aproximación de la cantidad irracional por ejemplo: Raíz (2) = 1.4141213562 ≈ 1.4 después convertirla en una cantidad racional después multiplicar a todos los arcos de la red por el m.c.m, una vez hecha esta transformación se resuelve el problema, una vez que converge a una solución se dividen todos los arcos de la red por el m.c.m y se habrá llegado a una solución, pero se debe estar conciente que la solución que se obtiene no es la óptima simplemente es factible y qué tan cerca del óptimo estará, eso dependerá de la aproximación que se utilice.

Para evitar la pérdida de memoria, al reetiquetar, se recomienda usar la siguiente metodología. Almacenar la información como superíndices, por ejemplo, si en la etapa inicial se tiene el intervalo $\left\| s,2 \right\|$ y en la siguiente iteración este aumenta en una unidad, se tiene $\left\| s,3^2 \right\|$ y así sucesivamente en cada reetiquetación $\left\| s,4^{32} \right\| \ldots \left\| s,4^{n(n-1)\ldots 2} \right\|$, donde esta información será de utilidad más adelante.

Ejemplo 3.7

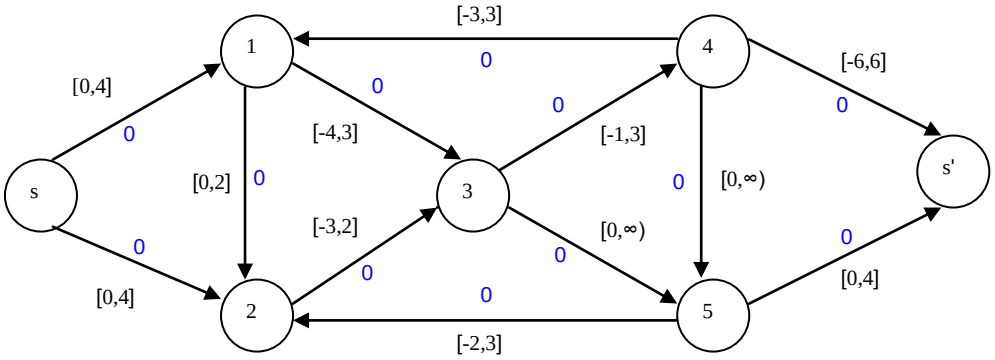
Utilizar el algoritmo original de Ford y Fulkerson de etiquetado, para resolver un problema de flujo máximo de la siguiente red.



Solución:

ITERACIÓN 1

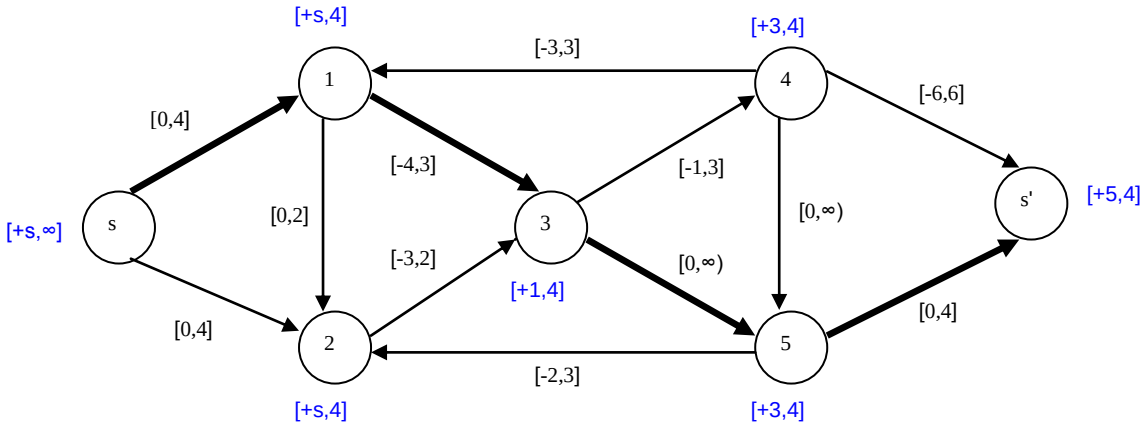
Paso 1: Iniciar con cualquier flujo factible x.



Paso 2: Etiquetar el origen con $[-,\infty]$.

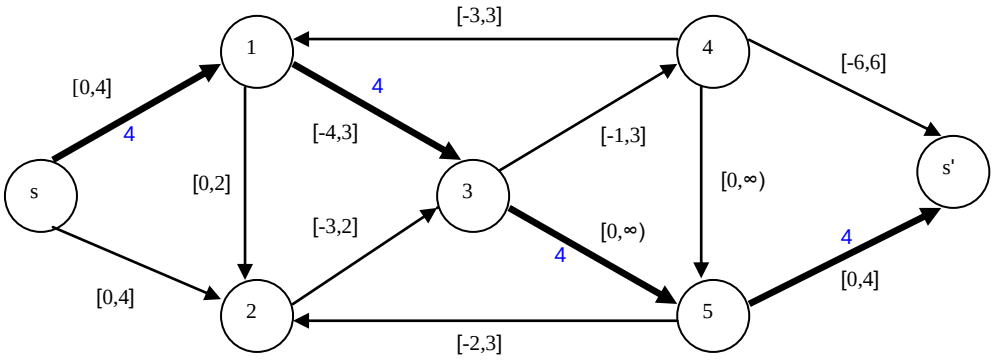
Paso 3: Elegir un nodo etiquetado y no examinado.

Paso 4: Dado que el nodo destino s' recibe etiqueta, eso implica que existe una trayectoria aumentante que va de s a s', ir al paso 5.



Paso 5: Borrar todas las etiquetas y regresar al paso 2.

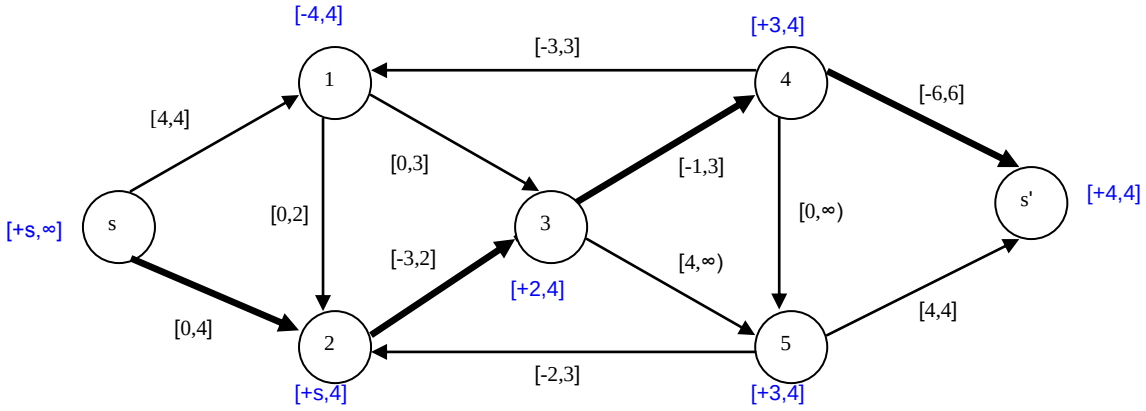
ITERACIÓN 2



Paso 2: Etiquetar el origen con $[-, \infty]$.

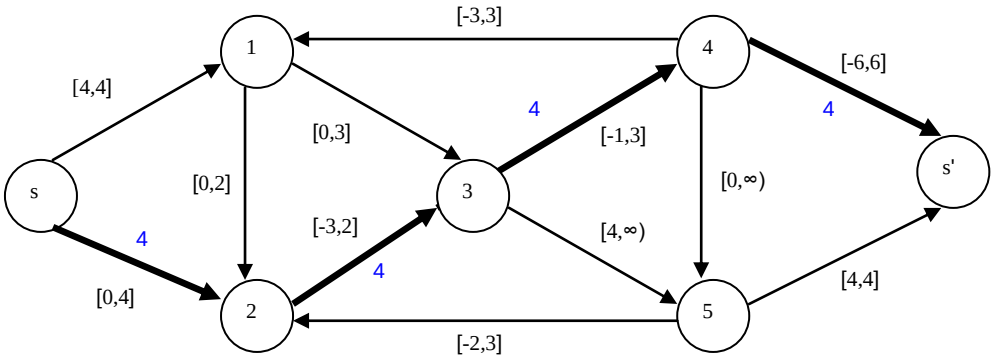
Paso 3: Elegir un nodo etiquetado y no examinado.

Paso 4: Dado que el nodo destino s' recibe etiqueta, eso implica que existe una trayectoria aumentante que va de s a s', ir al paso 5.



Paso 5: Borrar todas las etiquetas y regresar al paso 2.

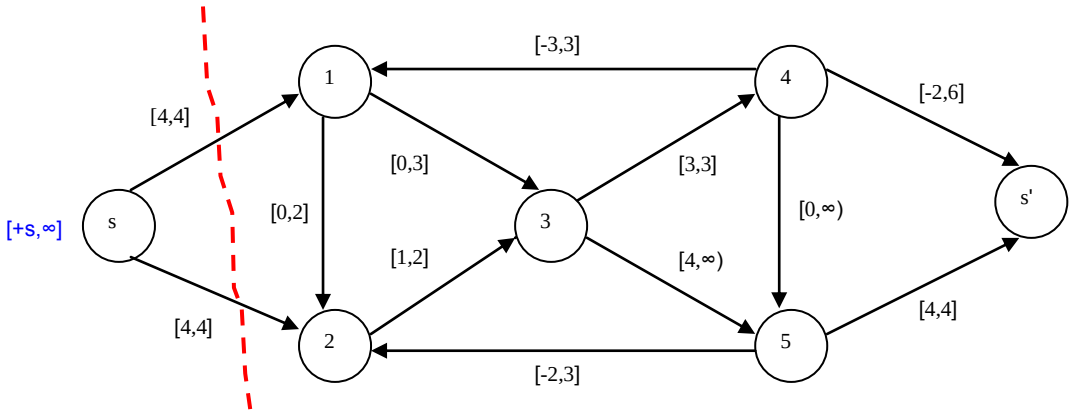
ITERACIÓN 3



Paso 2: Etiquetar el origen con $[-,\infty]$.

Paso 3: Elegir un nodo etiquetado y no examinado.

Paso 4: Dado que el nodo destino s' no recibe etiqueta y todos los nodos etiquetados han sido examinados terminar, ya que el flujo factible es máximo.

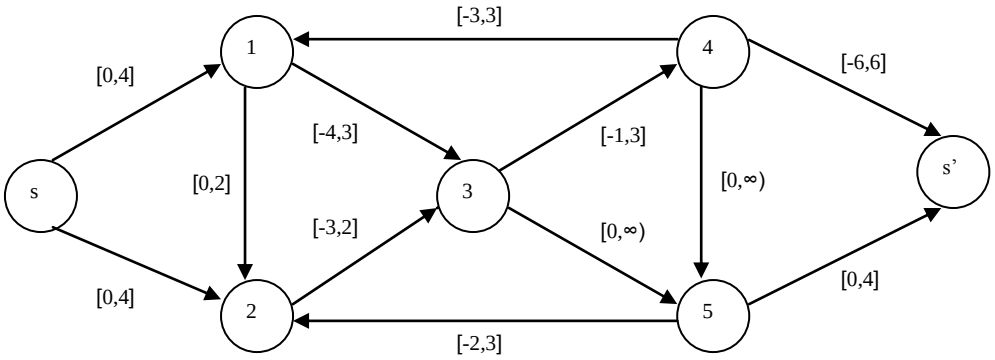


Como se puede observar, todos los nodos etiquetados han sido examinados y s' no recibió etiqueta por lo tanto, el último flujo es máximo. Por otro lado, el conjunto de nodos etiquetados s es $N = \{s\}$; por lo tanto, el corte mínimo es $(N, N') = \{(s,1),(s,2)\}$. Observe que su capacidad es $Q(N,N') = 4+4 = 8$ que es igual al valor del flujo máximo.

Como se puede observar, el algoritmo ya no presenta pérdida de memoria, pues conserva esta memoria en las mismas etiquetas. Y esta información puede ser utilizada más adelante con el fin de volver el método más eficiente. Además, si utilizamos en cada etapa, como recomienda el algoritmo elegir el arco de mayor anchura el algoritmo converge mas rápidamente. En este ejemplo no se puede mostrar ese detalle, pues existieron arcos de la misma anchura y la elección entre los arcos fue arbitraria.

Ejemplo 3.8

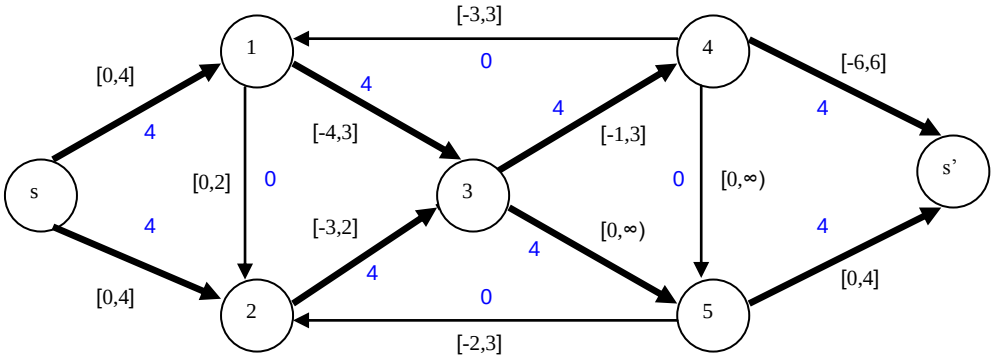
Utilizar ahora el algoritmo Ford-Fulkerson de etiquetado mejorado, para resolver el problema de flujo máximo del ejemplo anterior.



Solución:

ITERACIÓN 1

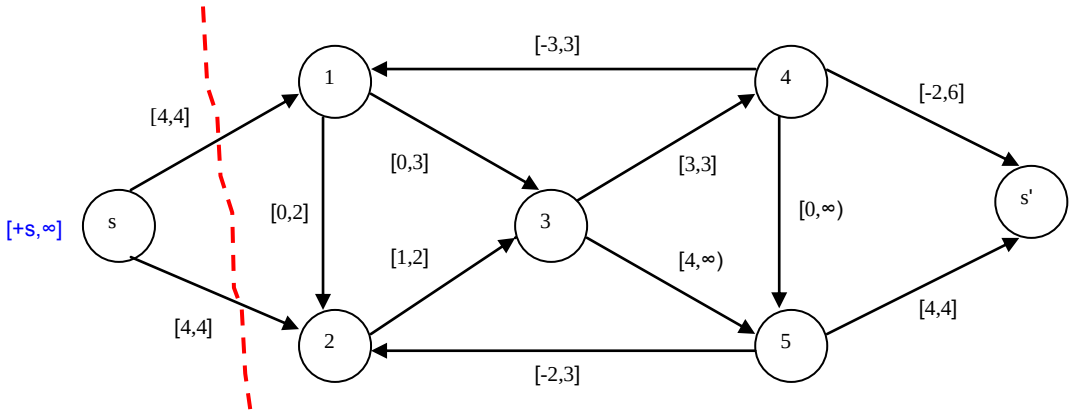
Paso1: Comenzar con un flujo de 4 $x(j)=4\ \forall\ j\in C_1$ y un flujo de 4 $x(j)=4\ \forall\ j\in C_2$ esto es posible porque todos los intervalos de capacidad de cada capa contienen al 4, los arcos que no pertenecen a una capa, inician con un flujo de 0 $\forall\ j\in R.$



Paso 2: Etiquetar el origen con $[+s,\infty]$.

Paso 3: Elegir un nodo etiquetado y no examinado.

Paso 4: Dado que el nodo destino s' no recibe etiqueta y todos los nodos etiquetados han sido examinados terminar, ya que el flujo factible es máximo.



Como se puede observar al utilizar la nueva manera de obtener el flujo máximo, por medio de la formulación obtenida por capas.

Capa 1: $s \to a \to s'$ = $\max\ j \in c_1$: {(0,4),(-4,3),(-1,3),(-6,6)} = 4

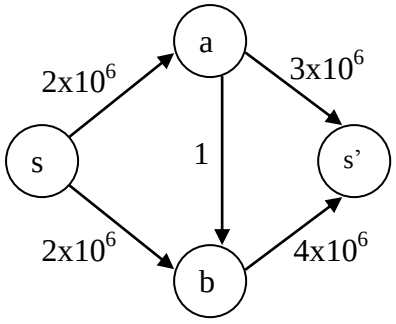
Capa 2: $s \to b \to s'$ = $\max\ j \in c_2$: {(0,4),(-3,2),(0,∞),(0,4)} = 4

$$Flujo\quad Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j) = (0) + (4+4) = 8$$

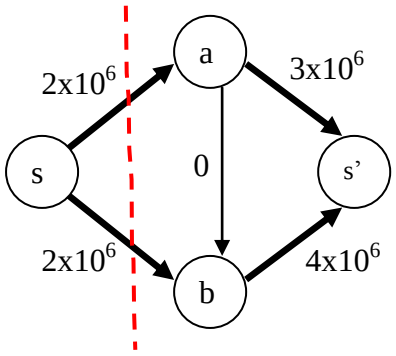
Ahora se puede validar el resultado obtenido utilizando el teorema de flujo máximo corte mínimo. Como se puede observar, todos los nodos etiquetados han sido examinados y s' no recibió etiqueta por lo tanto, el último flujo es máximo. Por otro lado, el conjunto de nodos etiquetados s es $N = \{s\}$; por lo tanto el corte mínimo es $(N, N') = \{(s,1),(s,2)\}$. Observe que su capacidad es $Q(N,N') = 4+4 = 8$ que es igual al valor de flujo máximo. Como se puede observar, la aportación al algoritmo de flujo máximo de etiquetado reduce también el número de iteración para llegar al óptimo, comparado con el método tradicional.

Como se puede observar, el número de iteraciones en este ejemplo se redujo considerablemente de **3 iteraciones** que requiere el algoritmo original de etiquetado, a sólo **1 iteración** que requiere el algoritmo de etiquetado mejorado. Para problemas más grandes se nota la gran diferencia que existe al utilizar el método tradicional y el mejorado.

La importancia de este capítulo es el mejorar la complejidad computacional del algoritmo original pues como se sabe, la complejidad del algoritmo de Ford-Fulkerson tiene una complejidad de $O(nmU)$, donde n representa el número de nodos iniciales, m el número de iteraciones y U representa que las capacidades en los arcos, son enteras, grandes y finitas. Con la propuesta de mejora, ésta complejidad se reduce a $O(nmu)$, donde $u = U - x(j)$ y representa que las capacidades en los arcos, son enteras, no tan grandes y finitas. Puesto que se les está restando la cota máxima perteneciente a todos los arcos de cada capa contenidos en la red.



El algoritmo de Ford-Fulkerson mejorado, evita este tipo de anomalías que presenta el algoritmo original, el cual requiere $4x10^6$ iteraciones y con la propuesta de mejora requiere de tan solo proponer el flujo inicial, el cual es óptimo y por lo tanto genera un corte obteniendo un flujo el cual es máximo, con el algoritmo mejorado se evita toda clase de anomalías y se reduce el número de iteraciones mejorando así la eficiencia.



Capa 1: $s \to a \to s'$ = $\max\ j \in c_1$: {(0,2x10^6),(0,2x10^6)} = 2x10^6

Capa 2: $s \to b \to s'$ = $\max\ j \in c_2$: {(0,2x10^6),(0,2x10^6)} = 2x10^6

$$Flujo\ \ Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j) = (0) + (2x10^6 + 2x10^6) = 4x10^6$$

3.3 NOTAS IMPORTANTES

Para la solución de problemas de flujo máximo también puede intervenir la programación lineal.

El modelo usando programación lineal es el siguiente:

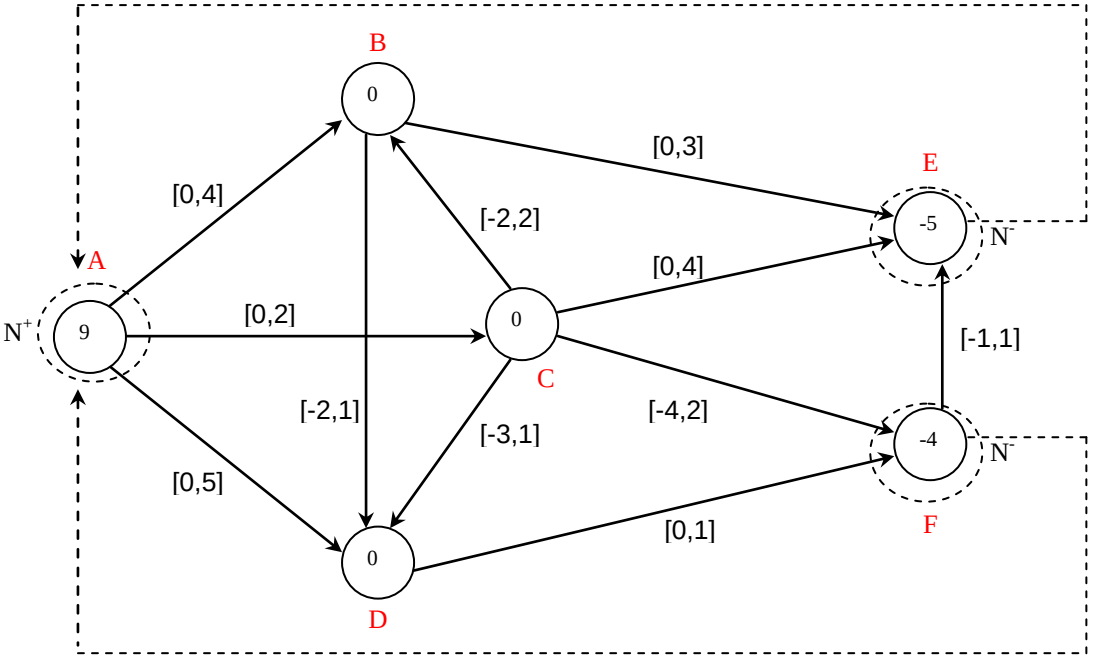
Maximizar z = f

Donde f es la cantidad desconocida de flujo.

Sujeto a:

Nodo inicial	$\sum x_{ij} = f$
Nodos de Transbordo	$\sum x_{ij} = 0$
Nodo destino	$\sum x_{ij} = f$
Artificial	$\sum x_{ij} = 0$
	$x_{ij} \geq 0 \quad \forall \ i, j$

Veamos cómo se realiza el planteamiento al problema de flujo máximo del ejemplo anterior, gráficamente la problemática seria la siguiente:



El modelo es el siguiente:

$$\max \quad z = x_{EA} + x_{FA}$$

$$s.a$$

$$\begin{aligned}
&nodo\ (A) && x_{AB} + x_{AC} + x_{AD} = 9 \\
&nodo\ (B) && x_{BE} + x_{BD} - x_{AB} - x_{CB} = 0 \\
&nodo\ (C) && x_{CE} + x_{CD} + x_{CF} + x_{CB} - x_{AC} = 0 \\
&nodo\ (D) && x_{DF} - x_{CD} - x_{BD} - x_{AD} = 0 \\
&nodo\ (E) && x_{BE} + x_{CE} + x_{FE} = 5 \\
&nodo\ (F) && x_{DF} + x_{CF} - x_{FE} = 4 \\
&artificial && x_{FA} + x_{EA} - x_{AB} - x_{AC} - x_{AD} = 0
\end{aligned}$$

$$\begin{aligned}
&4 \geq x_{AB} \geq 0 \\
&2 \geq x_{AC} \geq 0 \\
&5 \geq x_{AD} \geq 0 \\
&3 \geq x_{BE} \geq 0 \\
&2 \geq x_{CB} \geq -2 \\
&4 \geq x_{CE} \geq 0 \\
&2 \geq x_{CF} \geq -4 \\
&1 \geq x_{CD} \geq -3 \\
&1 \geq x_{DF} \geq 0 \\
&1 \geq x_{FE} \geq -1 \\
&1 \geq x_{BD} \geq -2
\end{aligned}$$

Este modelo resuelve el problema de flujo máximo utilizando programación lineal, y esto es de gran ayuda pues en la actualidad es mas sencillo realizar el modelo de un problema y resolverlo haciendo uso de algún *software* como LINDO, QSB o QM-for Windows entre otros. Así como la interpretación de la solución que este genere.

La modelación de programación lineal para resolver el problema de flujo máximo servirá de gran ayuda para la validación de los resultados obtenidos por el algoritmo de Ford-Fulkerson mejorado, dado que no es posible realizar una demostración formal.

■

CAPÍTULO 4

CASO DE APLICACIÓN

4.1 ANTECEDENTES

LICONSA, a través de la subdirección de distribución, ha detectado la necesidad de que se cuente con un servicio confiable, seguro y económico, que le permita realizar de forma optima sus actividades cotidianas de abasto y apoyo al desarrollo económico-social del país.

En este sentido se concluye que existe la necesidad de identificar los problemas que enfrenta LICONSA de manera explicita, para formular las alternativas y acciones de solución. LICONSA, tiene la enorme encomienda de contribuir con sus acciones, a disminuir los rezagos históricos en materia social, a través de su programa de abasto de leche, mediante el cual produce y abastece de este bien a familias de las zonas urbanas y rurales del país, en condiciones de pobreza extrema.

En la actualidad con el programa social de abasto de leche, LICONSA atiende a 4,537,995 beneficiados, de 2,629,532 familias, a través de 7,915 puntos de venta distribuidos en 1,811 municipios del país. Con el fin de atender esta demanda, se cuenta con un total de 11 plantas industriales, produciéndose en 8 de ellas solamente leche liquida, en 2 solo leche en polvo y en una leche liquida y en polvo.

La ubicación actual de estas plantas industriales es en los estados de: Colima, Guadalajara, Michoacán, Tlaxcala, D.F., Edo. de México, Veracruz y Oaxaca. La producción global de leche es de 3.5 millones de litros al día, de los cuales aproximadamente el 75% es leche liquida y el 25% es leche en polvo. Los lugares por donde ingresa la materia prima son: Nuevo Laredo para la leche liquida y los puertos de Manzanillo y Veracruz para la leche en polvo.

La distribución se realiza diariamente para la leche líquida y semanalmente para la leche en polvo, lo que implica atender la demanda que se produce en los 1,811 municipios del país y transportar hasta sus 7,915 puntos de venta, las cantidades de leche que requieran, situación que implica tener aproximadamente mas de 400 rutas de distribución. En la actualidad para hacer la distribución, se contrata transporte (camiones) de carga pesada, para enviar el producto de las plantas industriales a los centros de consolidación, ubicados en diferentes municipios del país y la distribución local hacia los puntos de venta, se realiza con vehículos de LICONSA y DICONSA.

4.2 PROBLEMA DE DISTRIBUCIÓN FACTIBLE

El caso particular del problema de flujo máximo mejor conocido como distribución factible que se utilizara para ilustrar el método de Ford-Fulkerson consiste principalmente en utilizar en nuestro caso el D.F para realizar nuestro análisis en el cual se tomara en cuenta las dos plantas de leche en polvo la cuales serán los nodos fuente, los arcos serán representados por las calles y avenidas, donde la intersección de calles o avenidas formaran nodos, los almacenes formaran parte de los nodos sumidero y el flujo que se requiere maximizar es el numero de camiones que van de los nodos fuente a los nodos sumidero en un tiempo determinado porque si no, no tendría sentido el análisis pues el flujo máximo seria el máximo numero de camiones de la empresa Liconsa. Además se debe tomar en cuenta que la capacidad de cada arco es la cantidad máxima de camiones que pueden circular por dicha avenida al mismo tiempo. En este caso el D.F no cuenta con leche Diconsa pero hay Liconsa.

Lo que se realizara en este trabajo siendo realistas es un caso de aplicación ilustrativo donde se pueda observar la ventaja de utilizar el caso de distribución factible mejorado que es un caso particular del algoritmo de Ford-Fulkerson y se comparara con el caso particular de distribución factible original y los resultados se validaran con un modelo de programación lineal. Para se manejan varios supuestos donde se utilizaran como arcos las delegaciones pertenecientes al distrito federal y los nodos serán representados por la intersección de las delegaciones, no se utilizaran las calles o avenidas como se utilizaría en la realidad puesto que el problema se expandiría demasiado y realmente no resolvería gran cosa. Los nodos fuente y los nodos sumidero seguirán siendo las empresas de leche en polvo y los almacenes respectivamente.

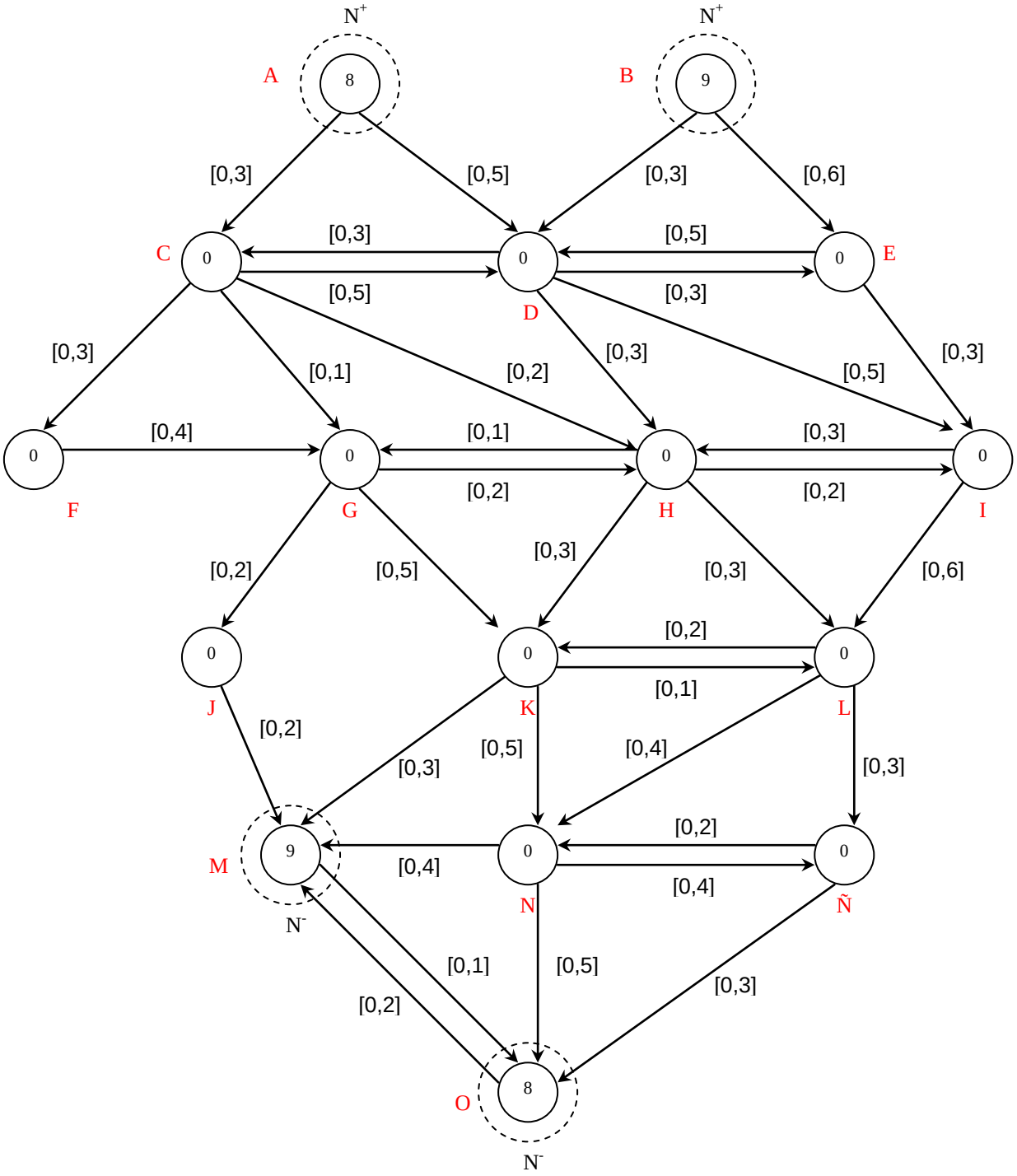
Datos:

Delegaciones del Distrito Federal:

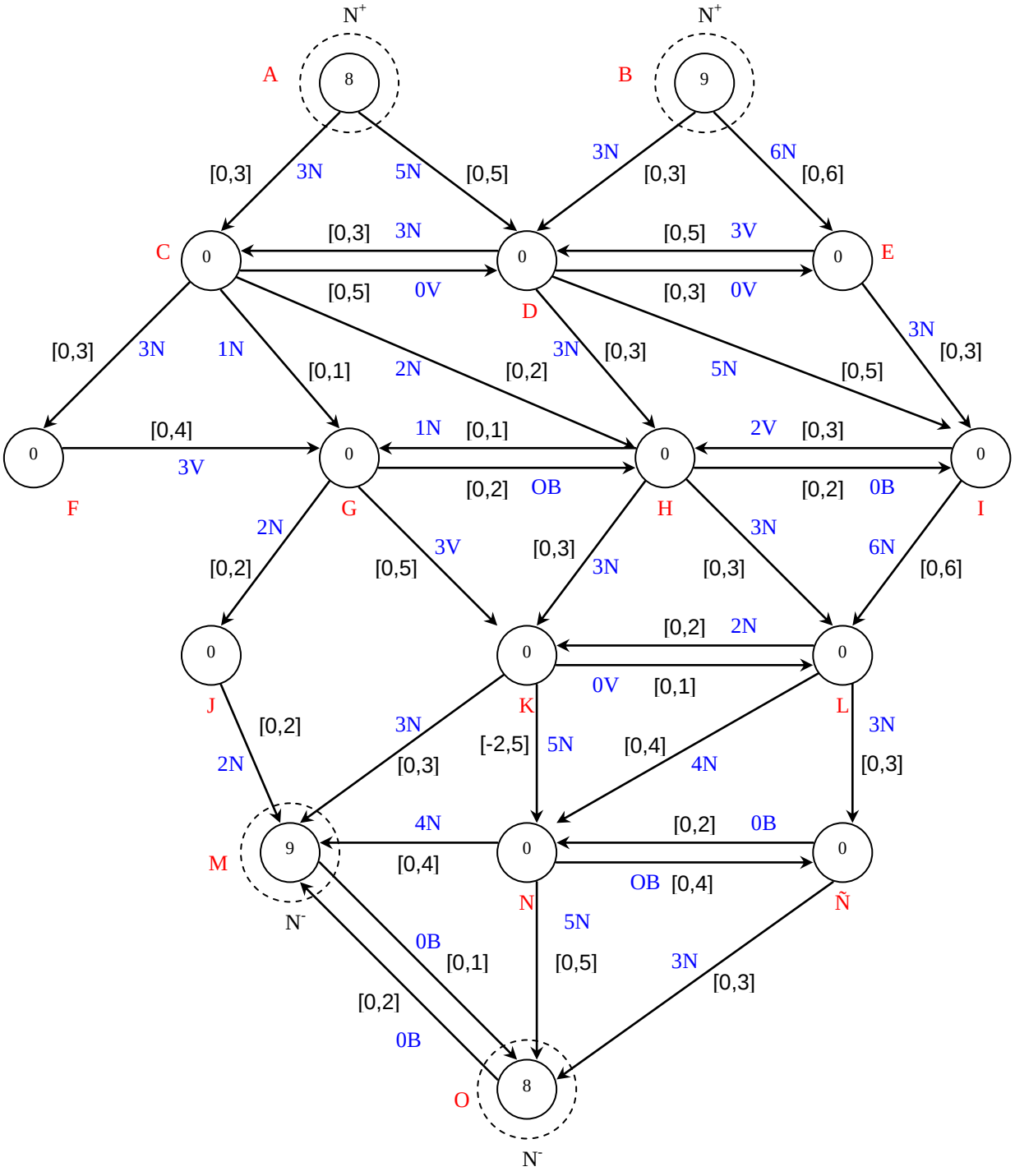
A = Azcapotzalco
B = Gustavo A. Madero
C = Miguel Hidalgo
D = Cuauhtemoc
E = Venustiano Carranza
F = Cuajimalpa
G = Alvaro Obregon
H = Benito Juarez
I = Iztacalco
J = La Magdalena Contreras
K = Coyoacan
L = Iztapalapa
M = Tlalpan
N = Xochimilco
Ñ = Tlahuac
O = Milpa Alta

El siguiente diagrama se elaboró tomando en cuenta la colindancia entre delegaciones del Distrito Federal donde las flechas indican las posibles rutas que pueden tomar los camiones y además los nodos fuentes son representados por la delegación Azcapotzalco y la Gustavo A. Madero en donde se tomo en cuenta que a pesar de que tiene colindancia, esta no es tomada en cuenta puesto que no tendría sentido enviar camiones de una planta a otra, mientras nuestros nodos sumidero son representados por la delegación Tlalpan y Milpa Alta. Donde la capacidad de cada arco representa la cantidad de camiones que pueden trasladarse de una delegación a otra en un tiempo determinado.

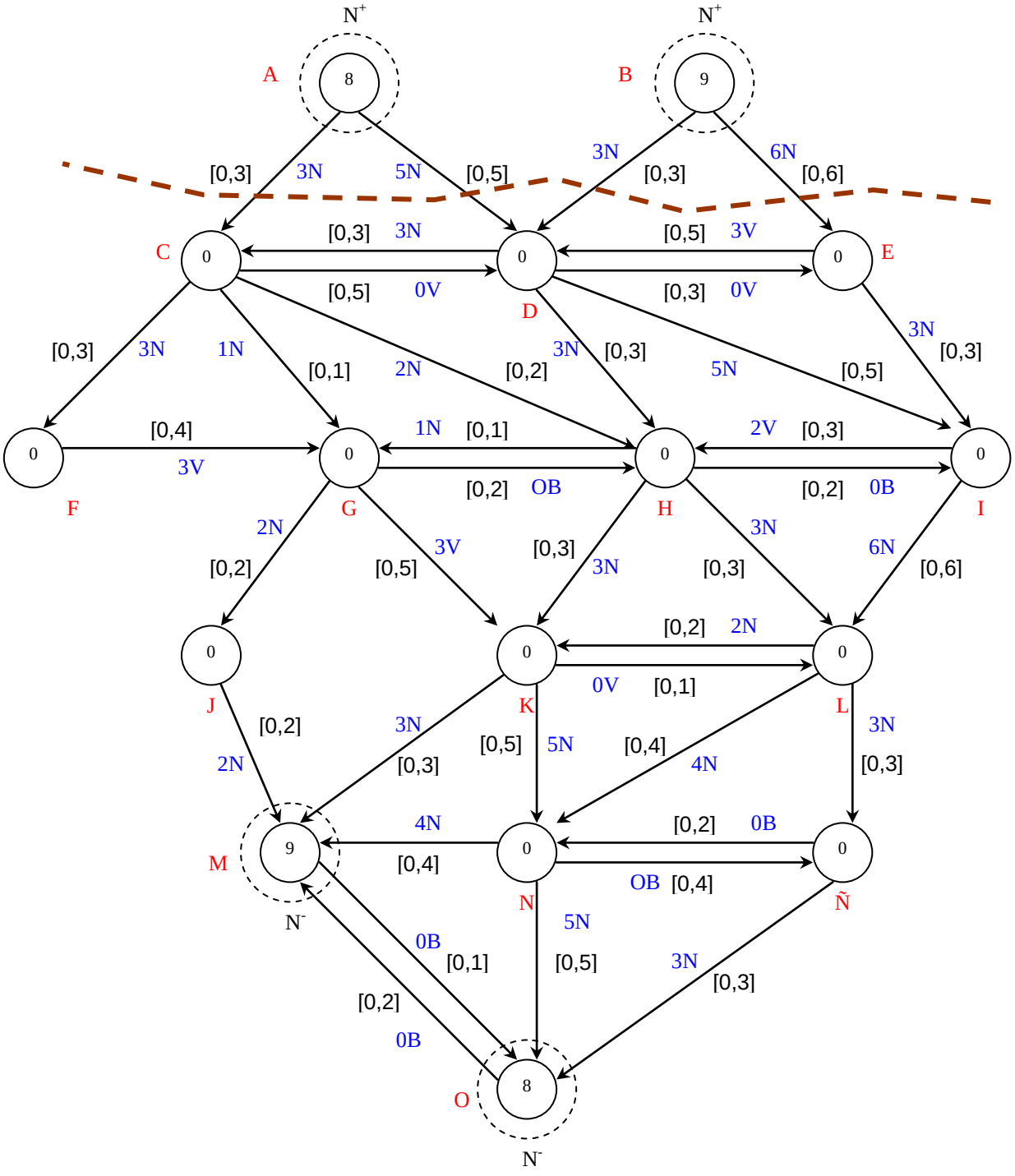
La siguiente grafica representa el problema:



Utilizando el caso de distribución factible mejorado se tiene el siguiente flujo inicial propuesto puesto que cumple con las restricciones de los nodos y se trata de violar lo menos que se pueda las restricciones de los arcos:



Como se puede observar el conjunto $A^- = \{\}$ y $A^+ = \{$ por lo tanto el flujo propuesto es óptimo. Pues se cumplen las restricciones de nodos y arcos, generando un corte $Q = [S, N/S]$ pues no existe una trayectoria compatible con la coloración que vaya de $N^+ \rightarrow N^-$.



$S = \{A, B\}$

$y(A) = 3+5 = 8$

$y(B) = 3+6 = 9$

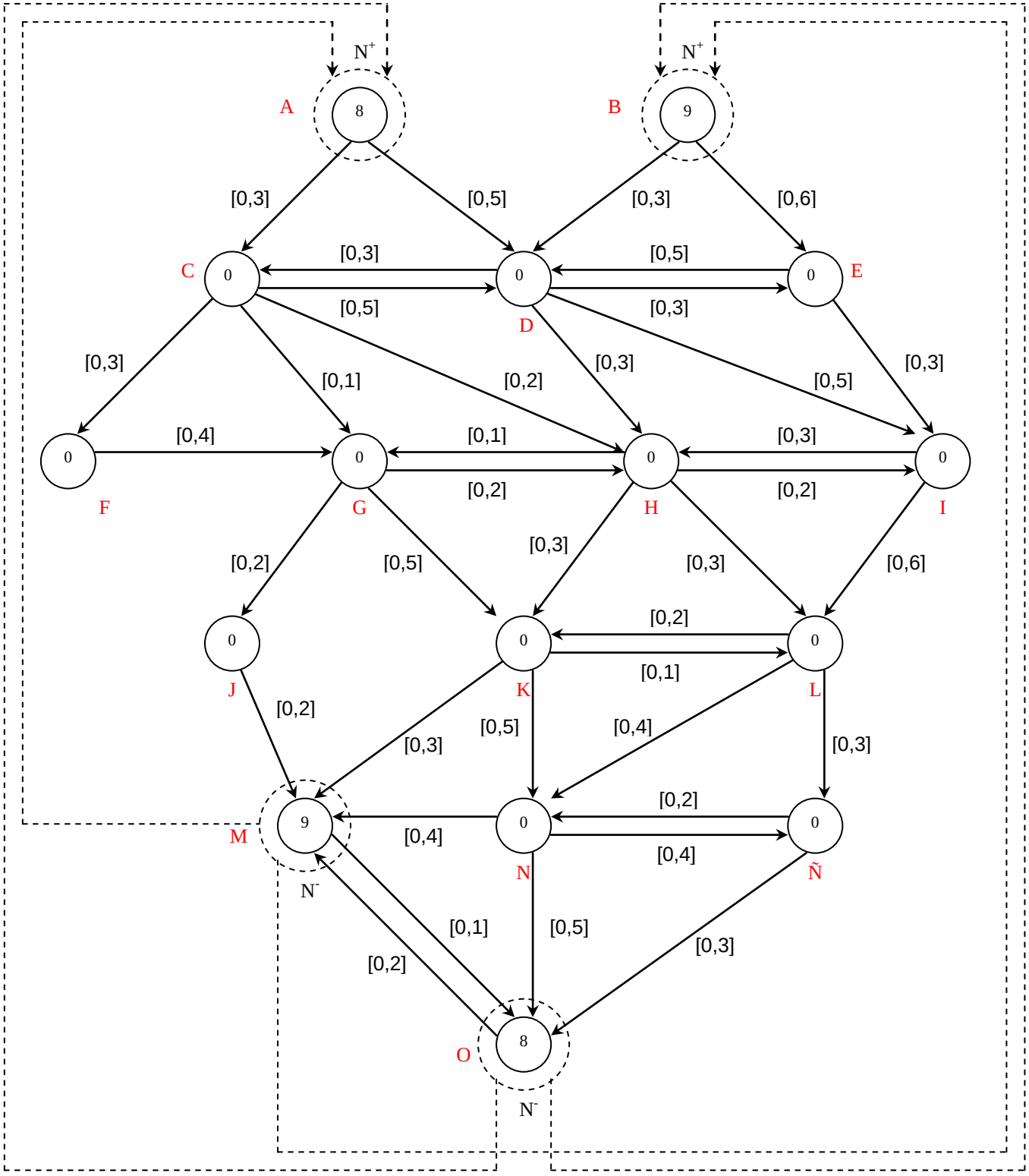
$\therefore y(s) = 17$

$Q^+ = \{(A,C),(A,D),(B,D),(B,E)\}$ $3+5+3+6 = 17$

$Q^- = \{\}$

$Q = Q^+ - Q^- = 17$

Para validar la solución se resolverá el problema de flujo máximo realizando un modelo de programación lineal.



El modelo es el siguiente:

x_{ij} : Es la cantidad de camiones que van de la delegación i a la delegación j.

$$\max \quad z = x_{MA} + x_{MB} + x_{OA} + x_{OB}$$

$$s.a$$

$$\begin{aligned}
&nodo\ (A) && x_{AC} + x_{AD} = 8\\
&nodo\ (B) && x_{BD} + x_{BE} = 9\\
&nodo\ (C) && x_{CD} + x_{CF} + x_{CG} + x_{CH} - x_{AC} - x_{DC} = 0\\
&nodo\ (D) && x_{DC} + x_{DE} + x_{HD} + x_{DI} - x_{AD} - x_{BD} - x_{CD} - x_{ED} = 0\\
&nodo\ (E) && x_{ED} + x_{EI} - x_{BE} - x_{DE} = 0\\
&nodo\ (F) && x_{FG} - x_{CF} = 0\\
&nodo\ (G) && x_{GH} + x_{GJ} + x_{GK} - x_{CG} - x_{FG} - x_{HG} = 0\\
&nodo\ (H) && x_{HG} + x_{HI} + x_{HK} + x_{HL} - x_{CH} - x_{DH} - x_{GH} - x_{IH} = 0\\
&nodo\ (I) && x_{IH} + x_{IL} - x_{DI} - x_{EI} - x_{HI} = 0\\
&nodo\ (J) && x_{JM} - x_{GJ} = 0\\
&nodo\ (K) && x_{KL} + x_{KM} + x_{KN} - x_{GK} - x_{HK} - x_{LK} = 0\\
&nodo\ (L) && x_{LK} + x_{LN} + x_{L\tilde{N}} - x_{HL} - x_{IL} - x_{KL} = 0\\
&nodo\ (N) && x_{NM} + x_{N\tilde{N}} + x_{NO} - x_{KN} - x_{LN} - x_{\tilde{N}N} = 0\\
&nodo\ (\tilde{N}) && x_{\tilde{N}N} + x_{\tilde{N}O} - x_{L\tilde{N}} - x_{N\tilde{N}} = 0\\
&nodo\ (M) && x_{JM} + x_{KM} + x_{NM} + x_{OM} - x_{MO} = 9\\
&nodo\ (O) && x_{MO} + x_{NO} + x_{\tilde{N}O} - x_{OM} = 8\\
&artificial\,(1) && x_{MA} + x_{OA} - x_{AC} - x_{AD} = 0\\
&artificial\,(2) && x_{MB} + x_{OB} - x_{BD} - x_{BE} = 0\\
&adicional(1) && x_{MA} + x_{OA} = 8\\
&adicional(2) && x_{MB} + x_{OB} = 9
\end{aligned}$$

$$3 \geq x_{AC} \geq 0$$
$$5 \geq x_{AD} \geq 0$$
$$3 \geq x_{BD} \geq 0$$
$$6 \geq x_{BE} \geq 0$$
$$5 \geq x_{CD} \geq 0$$
$$3 \geq x_{CF} \geq 0$$
$$1 \geq x_{CG} \geq 0$$

$2 \geq x_{CH} \geq 0$
$3 \geq x_{DC} \geq 0$
$3 \geq x_{DE} \geq 0$
$3 \geq x_{DH} \geq 0$
$5 \geq x_{DI} \geq 0$
$5 \geq x_{ED} \geq 0$
$3 \geq x_{EI} \geq 0$
$4 \geq x_{FG} \geq 0$
$2 \geq x_{GH} \geq 0$
$2 \geq x_{GJ} \geq 0$
$5 \geq x_{GK} \geq 0$
$1 \geq x_{HG} \geq 0$
$2 \geq x_{HI} \geq 0$
$3 \geq x_{HK} \geq 0$
$3 \geq x_{HL} \geq 0$
$3 \geq x_{IH} \geq 0$
$6 \geq x_{IL} \geq 0$
$2 \geq x_{JM} \geq 0$
$1 \geq x_{KL} \geq 0$
$3 \geq x_{KM} \geq 0$
$5 \geq x_{KN} \geq 0$
$2 \geq x_{LK} \geq 0$
$4 \geq x_{LN} \geq 0$
$3 \geq x_{L\tilde{N}} \geq 0$
$1 \geq x_{MO} \geq 0$
$4 \geq x_{NM} \geq 0$
$4 \geq x_{N\tilde{N}} \geq 0$
$5 \geq x_{NO} \geq 0$
$2 \geq x_{\tilde{N}N} \geq 0$
$3 \geq x_{\tilde{N}O} \geq 0$
$2 \geq x_{OM} \geq 0$

■

Este modelo resuelve el problema de flujo máximo utilizando programación lineal.

Utilizando el paquete Lindo para resolver el problema de flujo máximo se obtiene:

F:\JUAN\TESIS10.LTX

```
max XMA + XMB + XOA + XOB

ST

XAC + XAD = 8
XBD + XBE = 9
XCD + XCF + XCG + XCH - XAC - XDC = 0
XDC + XDE + XDH + XDI - XCD - XED - XBD - XAD = 0
XED + XEI - XBE - XDE = 0
XFG - XCF = 0
XGH + XGJ + XGK - XCG - XFG - XHG = 0
XHG + XHI + XHK + XHL - XCH - XDH - XGH - XIH = 0
XIH + XIL - XDI - XEI - XHI = 0
XJM - XGJ = 0
XKL + XKM + XKN - XGK - XHK - XLK = 0
XLK + XLN + XLÑ - XHL - XIL - XKL = 0
XNM + XNÑ + XNO - XKN - XLN - XÑN = 0
XÑN + XÑO - XLÑ - XNÑ = 0
XJM + XKM + XNM + XOM - XMO = 9
XMO + XNO + XÑO - XOM = 8
XMA + XOA - XAC - XAD = 0
XMB + XOB - XBD - XBE = 0
XMA + XOA = 8
XMB + XOB = 9

 XHI <= 2              XAC <= 3
 XHK <= 3              XAD <= 5
 XHL <= 3              XBD <= 3
 XIH <= 3              XBE <= 6
 XIL <= 6              XCD <= 5
 XJM <= 2              XCF <= 3
 XKL <= 1              XCG <= 1
 XKM <= 3              XCH <= 2
 XKN <= 5              XDC <= 3
 XLK <= 2              XDE <= 3
 XLN <= 4              XDH <= 3
 XLÑ <= 3              XDI <= 5
 XMO <= 1              XED <= 5
 XNM <= 4              XEI <= 3
 XNÑ <= 3              XFG <= 4
 XNO <= 5              XGH <= 2
 XÑN <= 2              XGJ <= 2
 XÑO <= 3              XGK <= 5
 XMA <= 8              XHG <= 1
 XOB <= 9
END
```

Reports Window

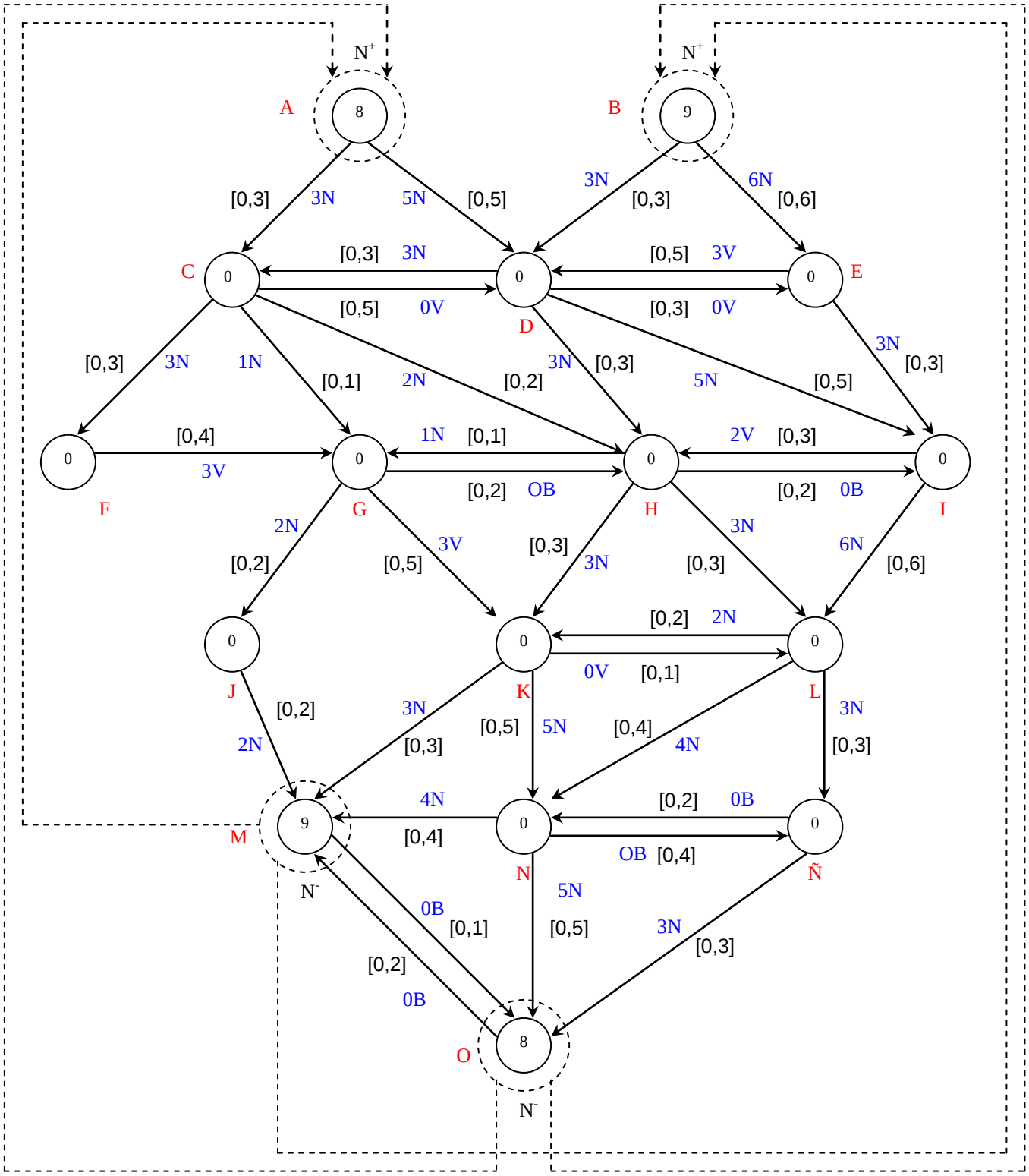
OBJETIVE FUNCTION VALUE

1) 17.00000

VARIABLE	VALUE	REDUCED COST
XMA	0.000000	0.000000
XMB	9.000000	0.000000
XOA	8.000000	0.000000
XOB	0.000000	0.000000
XAC	3.000000	0.000000
XAD	5.000000	0.000000
XBD	3.000000	0.000000
XBE	6.000000	0.000000
XCD	0.000000	0.000000
XCF	3.000000	0.000000
XCG	1.000000	0.000000
XCH	2.000000	0.000000
XDC	3.000000	0.000000
XDE	0.000000	0.000000
XDH	3.000000	0.000000
XDI	5.000000	0.000000
XED	3.000000	0.000000
XEI	3.000000	0.000000
XFG	3.000000	0.000000
XGH	0.000000	0.000000
XGJ	2.000000	0.000000
XGK	3.000000	0.000000
XHG	1.000000	0.000000
XHI	0.000000	0.000000
XHK	3.000000	0.000000
XHL	3.000000	0.000000
XIH	2.000000	0.000000
XIL	6.000000	0.000000
XJM	2.000000	0.000000
XKL	0.000000	0.000000
XKM	3.000000	0.000000
XKN	5.000000	0.000000
XLK	2.000000	0.000000
XLN	4.000000	0.000000
XLÑ	3.000000	0.000000
XNM	4.000000	0.000000
XNÑ	0.000000	0.000000
XNO	5.000000	0.000000
XÑN	0.000000	0.000000
XÑO	3.000000	0.000000
XOM	0.000000	0.000000
XMO	0.000000	0.000000

NO. ITERATIONS= 22

Gráficamente se tiene la siguiente solución:



Como se puede observar el caso particular de distribución factible mejorado del algoritmo de Ford-Fulkerson reduce el número de iteraciones considerablemente comparado con el algoritmo original de distribución factible puesto que este requiere de 11 iteraciones para llegar a la solución óptima mientras que el mejorado de tan solo una. Las trayectorias del algoritmo original hasta llegar al óptimo son las siguientes:

$P_1 : A \to C \to F \to G \to J \to M \ldots\ldots\ldots\ldots\ldots\ldots\alpha_1 = \min\{3,4,2,2\} = 2$

$P_2 : A \to C \to F \to G \to K \to M \ldots\ldots\ldots\ldots\ldots\ldots\alpha_2 = \min\{1,2,5,3\} = 1$

$P_3 : A \to D \to C \to G \to K \to M \ldots\ldots\ldots\ldots\ldots\ldots\alpha_3 = \min\{3,1,4,2\} = 1$

$P_4 : A \to D \to C \to H \to G \to K \to M \ldots\ldots\ldots\ldots\alpha_4 = \min\{2,2,1,3,1\} = 1$

$P_5 : A \to D \to C \to H \to K \to N \to M \ldots\ldots\ldots\ldots\alpha_5 = \min\{1,1,3,5,4\} = 1$

$P_6 : A \to D \to H \to K \to N \to M \ldots\ldots\ldots\ldots\ldots\ldots\alpha_6 = \min\{3,2,4,3\} = 2$

$P_7 : B \to D \to H \to L \to K \to N \to M \ldots\ldots\ldots\ldots\ldots\alpha_7 = \min\{1,3,2,2,1\} = 1$

$P_8 : B \to D \to I \to H \to L \to K \to N \to O \ldots\ldots\ldots\ldots\alpha_8 = \min\{5,3,2,1,1,5\} = 1$

$P_9 : B \to D \to I \to H \to L \to N \to O \ldots\ldots\ldots\ldots\ldots\alpha_9 = \min\{4,2,1,4,4\} = 1$

$P_{10} : B \to E \to D \to I \to L \to N \to O \ldots\ldots\ldots\ldots\ldots\alpha_{10} = \min\{5,3,6,3,3\} = 3$

$P_{11} : B \to E \to I \to L \to \tilde{N} \to O \ldots\ldots\ldots\ldots\ldots\ldots\alpha_{11} = \min\{3,3,3,3\} = 3$

Como se puede observar la solución óptima de ambos métodos es la misma, tiene un flujo máximo de 17 camiones además contienen las mismas trayectorias de los camiones, no se agregaron las graficas del algoritmo original pues son 11 iteraciones y cada iteración genera una recoloración y por lo tanto una grafica.

4.3 ANALISIS DE SENSIBILIDAD

El análisis de sensibilidad es una de las partes mas importantes en la programación lineal sobre todo para la toma de decisiones, pues permite determinar cuando una solución sigue siendo optima, dados algunos cambios ya sea en el entorno del problema o en los datos del mismo.

Este análisis consiste en determinar que tan sensible es la respuesta óptima del método *simplex*, al cambio de algunos datos en la función objetivo (costos, capacidades, ganancias, etc.) o la disponibilidad de recursos (términos independientes de las restricciones).

La variación en estos datos del problema se analizará individualmente, es decir, se analizará la sensibilidad de la solución debido a la modificación de un dato a la vez, asumiendo que los demás permanecen sin alteración alguna. Esto es importante porque estamos hablando de que la sensibilidad es estática y no dinámica, pues solo contempla el cambio de un dato a la vez.

Una vez realizando este análisis se puede obtener el intervalo en el que puede moverse nuestra variable analizada y más adelante poder realizar varias modificaciones al problema original, siempre y cuando tomemos en cuenta la regla del 100%.

4.4 REGLA DEL 100%

La regla consiste en que para todos los coeficientes de la función objetivo que sean cambiados, la suma de los porcentajes de los posibles incrementos o decrementos representados por los cambios no exceda al 100% asi la solución optima no cambiará.

Caso general:

$$\begin{aligned} &\max \quad Cx \\ &s.a \quad Ax = b \\ &\qquad x \geq 0 \end{aligned}$$

$$C = [C_B, C_N] \qquad \Rightarrow \qquad \max \; [C_B, C_N] \cdot \begin{bmatrix} x_B \\ x_N \end{bmatrix}$$

$$A = [B, N] \qquad \text{s.a} \quad [B, N] \cdot \begin{bmatrix} x_B \\ x_N \end{bmatrix} = b$$

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} \qquad x_B \geq 0, x_N \geq 0$$

$$\begin{aligned} &\max z = C_B x_B + C_N x_N \qquad \ldots\ldots\ldots\ldots(1) \\ &s.a \qquad Bx_B + Nx_N = b \qquad \ldots\ldots\ldots\ldots(2) \\ &\qquad x_B \geq 0, x_N \geq 0 \end{aligned}$$

Multiplicando (2) por B^{-1} tenemos:

$$\begin{aligned} B^{-1}Bx_B + B^{-1}Nx_N &= B^{-1}b \\ \mathrm{I}x_B + B^{-1}Nx_N &= B^{-1}b \\ x_B &= B^{-1}b - B^{-1}Nx_N \qquad .....(3) \end{aligned}$$

Sustituyendo (3) en (1) tenemos:

$$\max z = C_B(B^{-1}b - B^{-1}Nx_N) + C_N x_N$$
$$z = C_B B^{-1}b - C_B B^{-1}Nx_N + C_N x_N$$
$$z + 0x_B + (C_B B^{-1}N - C_N)x_N = C_B B^{-1}b \ldots\ldots\ldots(4)$$

$$Ix_B + B^{-1}Nx_N = B^{-1}b \quad \ldots\ldots(5)$$

Por lo tanto se tiene:

	z	x_B	x_N	LD
z	1	$\bar{o}$	$C_B B^{-1}N - C_N$	$C_B B^{-1}b$
x_B	$\bar{o}$	I	$B^{-1}N$	$B^{-1}b$

El siguiente ejemplo muestra con más detalle en que consiste la regla del 100%.

$$\begin{aligned} \max \quad & z = 2x_1 - x_2 + x_3 \\ s.a \quad & x_1 + x_2 + x_3 \le 6 \\ & -x_1 + 2x_2 \le 4 \\ & x_1 \ge 0, x_2 \ge 0, x_3 \ge 0 \end{aligned}$$

Resolviendo con el *simplex* se tiene:

	z	x_1	x_2	x_3	x_4	x_5	LD
z	1	0	3	1	2	0	12
x_1	0	1	1	1	1	0	6
x_5	0	0	3	1	1	1	10

$$C_B = \begin{matrix} x_1 & x_5 \\ \end{matrix} \begin{bmatrix} 2, 0 \end{bmatrix}$$

$$C_N = \begin{matrix} x_2 & x_3 & x_4 \\ \end{matrix} \begin{bmatrix} -1, 1, 0 \end{bmatrix}$$

$$B = \begin{matrix} x_1 & x_5 \\ \end{matrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \quad \Rightarrow \quad B^{-1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$N = \begin{matrix} x_2 & x_3 & x_4 \end{matrix}\begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

Realizar cambios sobre el vector C en la función objetivo tenemos:

$$\begin{aligned} C_B B^{-1} N - C_N &= [2,0] \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{bmatrix} - [1,1,0] \\ &= [2,0] \cdot \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{bmatrix} - [1,1,0] \\ &= [2,2,2] - [1,1,0] \\ &= [1,1,2] \end{aligned}$$

Lo que significa que para que no se pierda optimalidad, el intervalo de incremento y decremento a partir de la cantidad inicial para C_2, C_3 y C_4 es el siguiente:

$$C_2 = (-\infty, 3]$$
$$C_3 = (-\infty, 1]$$
$$C_4 = (-\infty, 2]$$

Observar cuanto se puede mover C_B

$$\begin{aligned} C_B B^{-1} N - C_N &= [2+t,0] \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{bmatrix} - [1,1,0] \\ &= [2+t,0] \cdot \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{bmatrix} - [1,1,0] \\ &= [2+t,2+t,2+t] - [1,1,0] \end{aligned}$$

$$2 + t + 1 \geq 0 \qquad 2 + t - 1 \geq 0 \qquad 2 + t - 0 \geq 0$$
$$t \geq -3 \qquad t \geq -1 \qquad t \geq -2$$

$\therefore t \geq -1$

$$C_B B^{-1} N - C_N = \begin{bmatrix} 2,0+t \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1,1,0 \end{bmatrix}$$

$$= \begin{bmatrix} 2+t,t \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1,1,0 \end{bmatrix}$$

$$= \begin{bmatrix} 2+3t,2+t,2+t \end{bmatrix} - \begin{bmatrix} 1,1,0 \end{bmatrix}$$

$$2+3t+1 \geq 0 \qquad 2+t-1 \geq 0 \qquad 2+t-0 \geq 0$$

$$t \geq -1 \qquad t \geq -1 \qquad t \geq -2$$

$\therefore t \geq -1$

Por lo tanto los cambios de incremento y decremento a partir de la cantidad inicial para C_1 y C_5 son los siguientes:

$$C_1 = \begin{bmatrix} -1,\infty \end{bmatrix}$$
$$C_5 = \begin{bmatrix} -1,\infty \end{bmatrix}$$

Aplicando la regla del 100% se tiene: $\left\{ \begin{matrix} C_1 \rightarrow -0.5 \rightarrow 50\% \\ C_2 \rightarrow +1.0 \rightarrow 33\% \\ C_3 \rightarrow +0.1 \rightarrow 10\% \end{matrix} \right\} = 93\%$

Con lo cual la solución seguirá siendo optima.

Ver ahora cuanto se puede mover el vector b

$$B^{-1}b = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \left[\begin{bmatrix} 6 \\ 4 \end{bmatrix} + \begin{bmatrix} t_1 \\ 0 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 6 \\ 10 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 6 \\ 10 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_1 \end{bmatrix}$$

$$= \begin{bmatrix} 6+t_1 \\ 10+t_1 \end{bmatrix}$$

$$6 + t_1 \geq 0 \qquad 10 + t_1 \geq 0$$
$$t_1 \geq -6 \qquad t_1 \geq -10$$

$$\therefore t_1 \geq -6$$

$$B^{-1}b = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \left[\begin{bmatrix} 6 \\ 4 \end{bmatrix} + \begin{bmatrix} 0 \\ t_2 \end{bmatrix} \right]$$
$$= \begin{bmatrix} 6 \\ 10 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ t_2 \end{bmatrix}$$
$$= \begin{bmatrix} 6 \\ 10 \end{bmatrix} + \begin{bmatrix} 0 \\ t_2 \end{bmatrix}$$
$$= \begin{bmatrix} 6 \\ 10 + t_2 \end{bmatrix}$$

$$10 + t_2 \geq 0$$
$$t_2 \geq -10$$

$$\therefore t_2 \geq -10$$

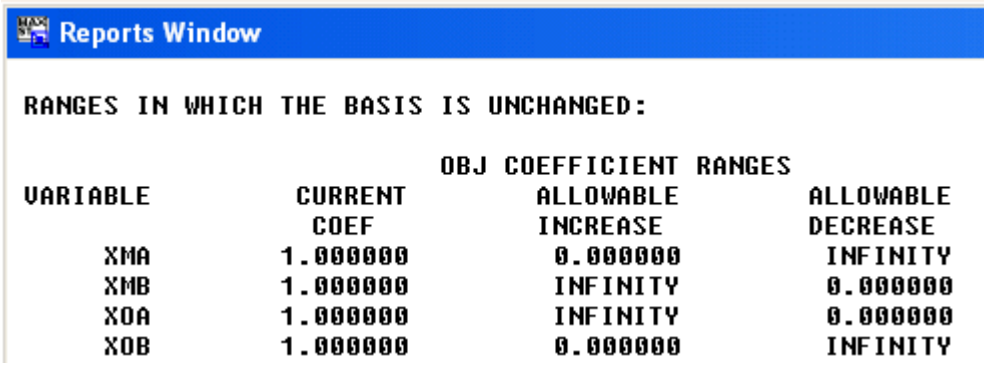
Por lo tanto los cambios de incremento y decremento a partir de la cantidad inicial para b_1 y b_2 son las siguientes:

	minimo	maximo
b_1	-6	∞
b_2	-10	∞

$\therefore$ Estos son los cambios permitidos tantos para el vector C en la función objetivo asi como para el vector b perteneciente a las restricciones este rango permite realizar un análisis de sensibilidad manteniéndonos en el óptimo.

Realizando el análisis de sensibilidad para el problema del flujo máximo de la leche Liconsa se tiene:

$\therefore$ Los incrementos o decrementos que puede sufrir el vector C en la función objetivo son las siguientes:



Reports Window

RANGES IN WHICH THE BASIS IS UNCHANGED:

	OBJ COEFFICIENT RANGES		
VARIABLE	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
XMA	1.000000	0.000000	INFINITY
XMB	1.000000	INFINITY	0.000000
XOA	1.000000	INFINITY	0.000000
XOB	1.000000	0.000000	INFINITY

Como se puede observar los coeficientes de los costos c_1, c_2, c_3, c_4 pueden variar de la siguiente manera:

$C_1 = (0,\infty) \rightarrow$ lo que significa que no se puede disminuir nada nuestra variable, pero si se puede incrementar hasta ∞ sin perder factibilidad.

$C_2 = (\infty,0) \rightarrow$ lo que significa que se puede disminuir hasta ∞ nuestra variable, pero si se puede incrementarla para no perder factibilidad.

$C_3 = (\infty,0) \rightarrow$ lo que significa que se puede disminuir hasta ∞ nuestra variable, pero si se puede incrementarla para no perder factibilidad.

$C_4 = (0,\infty) \rightarrow$ lo que significa que no se puede disminuir nada nuestra variable, pero si se puede incrementar hasta ∞ sin perder factibilidad.

Ahora bien las variantes que puede sufrir en el vector b son las siguientes:

Por ejemplo:

La variable b_1 perteneciente a la primera restricción no puede sufrir ningún incremento o decremento pues se perdería factibilidad $b_1 \rightarrow$ [illegible].

La variable b_{22} perteneciente a la restricción numero veinte y dos la cual puede sufrir un incremento de 3 o un decremento de 0 sin perder factibilidad $b_{22} \rightarrow$ [illegible].

La variable b_{37} perteneciente a la restricción numero treinta y siete la cual puede sufrir un incremento de infinito o un decremento de 2 sin perder factibilidad $b_{37} \rightarrow [0,2]$

Lo que se acaba de mencionar se justifica en la siguiente corrida de LINDO.

Reports Window

	RIGHTHAND SIDE RANGES		
ROW	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	8.000000	0.000000	0.000000
3	9.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000
10	0.000000	0.000000	0.000000
11	0.000000	0.000000	0.000000
12	0.000000	0.000000	0.000000
13	0.000000	0.000000	0.000000
14	0.000000	0.000000	0.000000
15	0.000000	0.000000	0.000000
16	9.000000	0.000000	0.000000
17	8.000000	0.000000	0.000000
18	0.000000	0.000000	0.000000
19	0.000000	0.000000	0.000000
20	8.000000	0.000000	0.000000
21	9.000000	0.000000	0.000000
22	3.000000	3.000000	0.000000
23	5.000000	INFINITY	0.000000
24	3.000000	INFINITY	0.000000
25	6.000000	2.000000	0.000000
26	5.000000	INFINITY	5.000000
27	3.000000	1.000000	0.000000
28	1.000000	1.000000	0.000000
29	2.000000	INFINITY	0.000000
30	3.000000	INFINITY	0.000000
31	3.000000	INFINITY	3.000000
32	3.000000	2.000000	0.000000
33	5.000000	1.000000	0.000000
34	5.000000	INFINITY	2.000000
35	3.000000	1.000000	0.000000
36	4.000000	INFINITY	1.000000

37	2.000000	INFINITY	2.000000
38	2.000000	INFINITY	0.000000
39	5.000000	INFINITY	2.000000
40	1.000000	INFINITY	0.000000
41	2.000000	INFINITY	2.000000
42	3.000000	1.000000	0.000000
43	3.000000	1.000000	0.000000
44	3.000000	INFINITY	1.000000
45	6.000000	INFINITY	0.000000
46	2.000000	INFINITY	0.000000
47	1.000000	INFINITY	1.000000
48	3.000000	2.000000	0.000000
49	5.000000	0.000000	0.000000
50	2.000000	0.000000	0.000000
51	4.000000	INFINITY	0.000000
52	3.000000	2.000000	0.000000
53	1.000000	INFINITY	1.000000
54	4.000000	0.000000	0.000000
55	3.000000	INFINITY	3.000000
56	5.000000	INFINITY	0.000000
57	2.000000	INFINITY	2.000000
58	3.000000	0.000000	0.000000
59	8.000000	INFINITY	8.000000
60	9.000000	INFINITY	9.000000

Por lo tanto en esta corrida se muestran las diferentes variantes que pueden sufrir las diferentes restricciones de nuestro problema sin perder factibilidad.

4.5 NOTAS IMPORTANTES

En cuanto a los costos del producto, se tiene que para la leche liquida un costo promedio de $3.3626 por litro, del cual el 4% corresponde a distribución, el 25% a operación, el 10% a producción, el 59% a materia prima y el 2% a material de envase. Para la leche en polvo se tiene que el costo promedio es de $4.2957 por litro, del cual el 6% corresponde a distribución, el 30% a operación, el 2% a producción, el 58% a materia prima y el 4% a material de envase. Esto implica que los costos de operación y los de materia prima, involucran el 88% del costo total para la leche en polvo, y para la leche liquida del 84%, correspondiendo a la distribución el 6% y el 4% de los costos totales, respectivamente.

Del análisis de esta información, se observa que si bien la etapa de distribución del producto (transportación de 3.5 millones de leche al día), es una de las partes principales de la Cadena de Suministro, esta importancia no se ve reflejada en los costos de producto, ya que solo representa menos del 6% del costo total, y realizar un estudio y análisis desde este punto de vista, trae como consecuencia una visión parcial del problema.

Situación que no permitirá visualizar de manera sistemática, las actividades involucradas en el proceso completo de flujo de materiales, esto es, desde los proveedores hasta los consumidores finales. Tales actividades incluyen transporte, mantenimiento de inventarios, procesamiento de pedidos, compras, empaquetamiento, almacenamiento, tratamiento de mercancías, gestión y transmisión de información, servicio al consumidor final, diseño y administración de canales de distribución, entre otras.

Es importante saber que se está analizando el flujo de los almacenes a las lecherías pues este problema seria realmente grande pero a continuación se muestra la siguiente tabla con los datos que indican el número de lecherías existentes por estado y su respectiva distribución estimada actual de leche en polvo en tiendas DICONSA.

ESTADOS	NUMERO DE LECHERIAS	CAJAS/MES
BAJA CALIFORNIA NORTE	22	710
BAJA CALIFORNIA SUR	20	602
CAMPECHE	33	988
COAHUILA	24	976
COLIMA	9	127
CHIAPAS	336	8,128
CHIHUAHUA	32	1,245
DURANGO	108	4,893
GUERRERO	120	5,111
HIDALGO	175	13,852
JALISCO	21	828
D.F	123	4,584
MICHOACAN	137	9,141
MORELOS	48	1,987
NAYARIT	72	3,132
NUEVO LEON	56	1,360
OAXACA	224	5,231
PUEBLA	108	3,004
QUERETARO	35	2,616
QUINTANA ROO	14	814
SAN LUIS POTOSI	170	6,243
SINALOA	21	824
SONORA	14	309
TABASCO	106	3,629
TAMAULIPAS	64	1,932
TLAXCALA	37	1,116
VERACRUZ	132	6,552
YUCATAN	123	4,133
ZACATECAS	50	1,915
TOTAL	**2,434**	**95,982**

Donde el factor de retito por beneficiario es de 419 ml. de leche en polvo. Debemos tomar en cuenta que para realizar un mejor análisis del problema deberían de tomarse los datos también de las lecherías y el problema se convertiría en un problema de trasbordo y el trasporte debería de ser a un costo mínimo, pero por cuestiones de ilustrar el algoritmo de Ford-Fulkerson no se utilizaron algunas variables importantes para el caso real.

Lo mas nuevo que se tiene es que tanto Liconsa y Diconsa estas dos empresas paraestatales, dependientes de la Secretaria de Desarrollo Social, han emprendido sendos programas de modernización tecnológica para servir mejor a sus beneficiarios. La primera distribuyo asistentes personales digitales entre sus promotores sociales y asi, Liconsa ha obtenido una operación mas imparcial y transparente durante el levantamiento de estudios socioeconómicos para quienes desean afiliarse a su programa de abasto social de leche.

Por su parte, Diconsa ha optado por desarrollar la red nacional de voz y datos basada en tecnología VPN (red privada virtual) para mejorar el abasto en sus más de 22 mil tiendas o puntos de venta. Los destinatarios de estos esfuerzos conforman la población más pobre y marginada del país; su interacción con ellos se realiza cara a cara todos los días, sea en sus hogares o en establecimientos dentro de su comunidad. De ahí el significado que pueden tener ambas empresas sociales para atender al sector más sensible con tecnología avanzada, “amable” y eficiente.

CONCLUSIONES Y RECOMENDACIONES

La mejoría que se propone para el algoritmo de Ford y Fulkerson en redes coloreadas consiste en utilizar un nuevo criterio el cual consiste en elegir un flujo inicial de nuestro nodo fuente y distribuirlo desde ahí hasta el nodo sumidero de tal manera que se cumplan las divergencias entre los nodos y se utilicen el menor numero de arcos posibles este flujo es bajo el criterio de la cota máxima perteneciente a todos los nodos contenidos en la red, este flujo saldrá del nodo fuente por cada arco perteneciente a este. Una vez propuesto este flujo inicial se aplica el algoritmo original, se colorean los arcos se busca una trayectoria compatible con la coloración se encuentra el α re actualiza el flujo sobre esa trayectoria de s a s' y se colorea y nuevamente se busca una trayectoria compatible con la coloración y asi sucesivamente hasta que no exista una trayectoria compatible con la coloración y se genere un corte. Con la aportación del criterio de cota máxima por capas se generó también una nueva forma de obtener el flujo máximo sin la necesidad de aplicar el teorema de flujo máximo corte mínimo para obtener la cantidad máxima que fluye en una red.

Una nueva forma de obtener el flujo máximo a partir de esta modificación es la siguiente:

$$Flujo \quad Maximo = \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{m} x(j)$$

Donde:

α_i : representa las cantidades de actualización de flujo de cada iteración.

$x(j)$: representa la cota máxima perteneciente a todos los arcos contenidos en cada capa la red.

n : representa el número de iteraciones.

m : representa la cantidad de capas de la red.

Es importante destacar que esta formulación no está en la bibliografía especializada consultada. Esta forma de obtener el flujo máximo de una red, sólo es aplicable para el algoritmo de Ford-Fulkerson mejorado que es una aportación de este trabajo.

Finalmente, se cumplió con el principal interés de esta tesis que fue mejorar el algoritmo de Ford-Fulkerson en su estructuración, con ello eliminar las anomalías y mejorar la complejidad computación del algoritmo.

La importancia de este trabajo es el mejorar la complejidad computacional del algoritmo original pues como se sabe, la complejidad del algoritmo de Ford-Fulkerson es $O(nmU)$, donde n representa el número de nodos iniciales, m el número de iteraciones y U representa que las capacidades en los arcos, son enteras, grandes y finitas. La propuesta de mejora, es muy significativa para valores muy grandes de U puesto que su complejidad se reduce a $O(nmu)$, donde $u = U - x(j)$ y representa que las capacidades en los arcos, son enteras, no tan grandes y finitas, puesto que se les está restando la cota máxima perteneciente a todos los arcos contenidos en cada capa de la red, y así evita la anomalía que presenta el algoritmo original: que para valores muy grandes de U, el algoritmo requiere de demasiadas iteraciones y utiliza mucho tiempo máquina para llegar a la solución, mientras que el algoritmo modificado requiere una cantidad pequeña de iteraciones.

Como se observa, los casos de distribución factible y rectificación de flujo del algoritmo Ford-Fulkerson sobre redes coloreadas, fueron realizados por Ford y Fulkerson con el objetivo de solucionar el problema cuando conocemos la cantidad inicial en nuestro nodo fuente, donde en la distribución factible se propone que el flujo es factible en los arcos y se inicia con un flujo factible x, donde x representa la mínima cota de todos los arcos pertenecientes a la red y se realiza un análisis sobre los nodos, donde los nodos cuya divergencia < demanda pertenecerán al conjunto N^+ y los nodos cuya divergencia > demanda pertenecerán al conjunto N^- enseguida se busca una trayectoria $P: N^+ \rightarrow N^-$ compatible con la coloración. Y se termina cuando los conjuntos de N^+ y N^- sean vaciós, puesto que se cumple que la divergencia es igual a la demanda.

En rectificación de flujo, se realiza proponiendo un flujo factible en los nodos y se inicia con un flujo factible x, donde x representa la mínima cota de todos los arcos pertenecientes a la red y se realiza un análisis sobre los arcos, donde los arcos que sobrepasan su capacidad, pertenecerán al conjunto A^+ y los arcos que estén por debajo de su capacidad pertenecerán al conjunto A^- ,después se determina un circuito elemental P compatible con la coloración que contenga al menos un arco perteneciente a A^+ o A^-, terminamos cuando los conjuntos de A^+ y A^- sean vacíos.

La mejoría que se propone para volver más eficientes tanto la distribución factible como la rectificación de flujo sobre redes coloreadas, es no iniciar el algoritmo proponiendo que el flujo inicial x sea el de la mínima cota entre todos los arcos pertenecientes de la red, si no que en el flujo inicial x, sea tomando en cuenta la capacidad máxima que se tiene en el nodo fuente, pues al ir transmitiendo dicho flujo en la red hasta llegar al nodo sumidero, se debe tomar el mínimo número de arcos posibles cuando se pasa de una etapa a otra y tratar de violar en lo más

mínimo, según sea el caso, las restricciones de arcos y nodos. Todo con el objetivo de que al colorear muchos de estos arcos, adquieran una coloración negra y con esto se saturen trayectorias más rápidamente, y esto produce que los algoritmos terminen en menor número de iteraciones por lo tanto se vuelven más eficientes en cuanto a tiempo.

Se recomienda, si se utilizan la distribución factible mejorada o la rectificación de flujo mejorado del algoritmo de Ford-Fulkerson de redes coloreadas, proponer un flujo inicial x tomar en cuenta las restricciones que se manejan para cada algoritmo, puesto que si se genera cualquier flujo inicial x, aleatoriamente este producirá un mayor número de iteración que el algoritmo original y en el peor de los casos, el algoritmo puede no converger.

Además, es recomendable realizar un análisis de sensibilidad, pues así se puede ver qué tan sensibles son nuestras variables, que tanto podemos incrementar o decrementar a partir del valor inicial nuestro vector c perteneciente a la función objetivo o al vector b perteneciente a las restricciones del problema, de tal manera que no se pierda factibilidad, y esto es posible mediante este análisis. Además en caso de que se quieran realizar algunas modificaciones simultaneas, hay que tomar en cuenta la regla del 100% la cual nos indica la cantidad máxima que podemos variarle al problema para no perder factibilidad.

La mejoría que se propone para volver mas eficiente el algoritmo de Ford-Fulkerson de etiquetado, es de gran ayuda pues corrige las anomalías que este presenta, desde la perdida de memoria , hasta cuando presenta valores irracionales en los arcos y además, utiliza la propuesta de evitar la cota mínima y utilizar el criterio de la cota máxima para cada capa, con el propósito de proponer un flujo inicial x partiendo de las capas que se generan desde el nodo fuente hasta el nodo sumidero, de tal modo que se inicie con un flujo máximo por capas, y en los arcos que no son parte de una capa, inicien con un flujo de cero que en la mayoría de los casos es igual a la cota mínima perteneciente a todos los arcos de la red, y en caso de que sea diferente de cero, hay que tomar en cuenta que se cumplan las restricciones de divergencia entre los nodos. Con esta metodología el algoritmo, se vuelve más eficiente, pues reduce considerablemente el número de iteraciones comparándolo con el algoritmo original.

La modelación de programación lineal para resolver el problema de flujo máximo sirvió de gran ayuda para la validación de los resultados obtenidos por el algoritmo de Ford-Fulkerson mejorado, dado que no es posible realizar una demostración formal. Finalmente, se recomienda comparar el algoritmo de Ford-Fulkerson mejorado con los nuevos algoritmos que sean desarrollado para atacar el problema de flujo máximo, y con esto se pueda visualizar de una mejor manera los alcances de este trabajo.

GLOSARIO

Árbol

Grafo conexo sin ciclos. Un subconjunto de arcos o aristas de un grafo que sea un árbol es un árbol parcial de dicho grafo.

Arco

Representación de la relación entre dos vértices de un grafo, cuando es relevante conocer el origen y el destino de la relación de otro modo, además de la dirección de la relación, debemos representar el sentido. Los arcos se encuentran en los grafos orientados.

Arista

Representación de la relación entre dos vértices de un grafo, cuando es relevante conocer el origen y el destino de la relación: de otro modo, además de la dirección de la relación, debemos representar el sentido. Los arcos se encuentran en los grafos orientados.

Arborescencia

Grafo orientado, fuertemente conexo, sin ciclos ni bucles, en que todos los vértices tendrán semigrado interior igual a la unidad, excepto uno, raíz de la arbolescencia, cuyo semigrado interior es 0. Es útil para representar procesos decisionales.

Bucle

Arco o arista cuyos vértices coinciden. Representa una conexión directa de un vértice consigo mismo.

Cadena

Camino no orientado, esto es, sucesión de aristas tal que el vértice extremo de cada una (exceptuando la última) coincide con el vértice extremo de la siguiente en la sucesión. Dos vértices de un grafo no orientado que no están conectados directamente con una arista pueden estar conectados indirectamente a través de una cadena.

Camino

Sucesión de arcos tal que el vértice extremo de cada uno (exceptuando el último) coincide con el vértice extremo del siguiente en la sucesión. Dos vértices de un grafo orientado que no estén conectados directamente con una arista pueden estarlo indirectamente por un camino.

Capas

Son aquellas trayectorias que salen del nodo fuente y llegan al nodo sumidero cubriendo a la red

Ciclo

Cadena que se inicia y termina en el mismo vértice. Por extensión, en un grafo no orientado se define como un conjunto de arcos que unen una serie de vértices, prescindiendo de su orientación.

Circuito

Camino que se inicia y termina en el mismo vértice. Representa una conexión indirecta de un vértice consigo mismo en un grafo orientado. Se trata de un concepto más fuerte que el de ciclo: todos los circuitos son ciclos, pero no todos los ciclos son circuitos.

Conexo (grafo)

Grafo en que existe al menos una cadena entre toda pareja de vértices. El concepto es aplicable tanto a grafos orientados como para no orientados.

Distancia

Valor numérico asociado a un arco o arista de un grafo que representa posibilidades de comunicación. La distancia de un camino es igual a la suma de las distancias de los arcos que componen dicho camino. Según la situación que queramos representar, los valores de distancia de los arcos pueden representar tiempo, costo u otros conceptos, en vez de distancia.

Flujo

Magnitud asociada a un arco, que representa la cantidad de determinada variable vehiculada a través de dicho arco por unidad de tiempo. Además del significado mas obvio de caudal de fluido, puede representar en ciertas situaciones magnitudes como productividad (producción por unidad de tiempo). El valor total de los flujos que llegan a un vértice ha de ser igual al flujo neto establecido para dicho vértice.

Flujo Total

Cantidad máxima de flujo que puede vehicular una red de transporte. Es igual al flujo que emerge de los vértices origen y al que incide en los vértices destino.

Fuertemente conexo (grafo)

Grafo en que existe al menos un camino entre toda pareja de vértices. Es una propiedad mas fuerte que la de grafo conexo, dado que todo grafo fuertemente conexo es conexo.

Grado

Es el número total de arcos que tienen origen o destino en el vértice. Es igual a la suma del semigrado interior y semigrado exterior.

Grafo

Representación de las relaciones existentes entre los elementos de un sistema. Los elementos se representan por los vértices del grafo, y las relaciones por arcos o aristas.

Grafo orientado

Representación de las relaciones existentes entre los elementos de un sistema, cuando la relación entre dos elementos i y j no tiene porque ser la misma que la existente entre j e i. en los grafos orientados, debemos distinguir entre el origen y el destino de la relación que establezcamos.

Grafo no orientado

Representación de las relaciones existentes entre los elementos de un sistema, cuando la relación entre dos elementos i y j es siempre la misma que la existente entre j e i.

Red de trasporte

Grafo que representa las posibilidades de vehicular flujos a través de un conjunto de vértices. Cuando es posible vehicular flujo entre dos vértices, tendremos un arco al que usualmente asociaremos un valor máximo de flujo.

Red de flujo

Una red de flujo es un grafo dirigido $G = (N, A)$ donde cada arco (u, v) perteneciente a la red tiene una capacidad no negativa. Se distinguen dos nodos: la fuente o nodo s, y el sumidero o nodo s'. Si existen múltiples fuentes y sumideros, el problema se puede simplificar añadiendo una fuente común y un sumidero común.

Semigrado exterior de un vértice

Número de arcos que emergen de un vértice. Un vértice de una red de transporte con semigrado exterior igual a cero es un destino de los flujos de dicha red.

Semigrado interior de un vértice

Número de arcos que inciden en un vértice. Un vértice de una red de transporte con semigrado interior igual a cero es un origen de los flujos de dicha red.

Vértice

Representación de los elementos del sistema cuyas relaciones vienen representadas en el grafo.

BIBLIOGRAFIA

1.- AHUJA, R, Orlin, J. B., Tarjan, R .E., "Improved time Bounds for the Maxium Flows Problem",*SIAM Journal of Computing* 18 (1989) 939-954.

2.- AHUJA, R, Kodialam, M., Mishra, A. K., Orlin, J. B., "computational investigations of Maxium flow algorithms", *European Journal of Operational research* 97 (1997) 509-542.

3.- AHUJA, R, Magnanti, T., Orlin, J. B., "*Network Flows*". In Optimization. Handbooks in Operations Research and Management Science 1 (1989) 211-369, G.L.Nemhauser, A.H.G. Rinnoy Kan, M.J.Todd (eds.), North Holland.

4.- AHUJA, Ravindra K.; Magnanti, Thomas L.; y Orlin, James B. *Network Flows*. Prentice Hall. New Jersey, EE. UU. 1993.

5.- AHUJA, R, Magnanti, T., Orlin, J. B., "*Network Flows*". Prentice-Hall, inc (1993)

6.- AHUJA, R, Orlin, J. B., "Distance-Directed Augmenting Path algorithms for Maxium Flow and Parametric Maxium flow problems", *Naval Research Logistics Quarterly* 38 (1991) 413-430.

7.- ANDERSON, R. J., Setubal, J. C., "Parallel and sequential implementations of Maximum-flow algorithms ", in: D.S. Johnson and C.C. McGeoch (eds), *Network flows and Matching*: First DIMACS Implementation challenge, DIMACS Series in Discrete Mathematics and Theoretical computer Science,12 (1993) American Mathematical Society.

8.- ANDERSON, V. L., Mclean, R. A., *Design of Experiments*. *A realistic approach*. Marcel Dekker (1974).

9.- ANEJA, Y. P. and Nair K. P., "Bicriteria transportation Problem", Magnageement Science, Vol.25,no 1, January (1979).

10.- AYUSO, Ma.del Carmen Hernández Ayuso., "Introducción a la teoría de redes", Segunda edición. Sociedad Matemática Mexicana (2005) 75-78.

11.- BAZARAA, M. S., Jarvis, J. J., Sherali, H. D., "Linear Programming and Network Flows", 2nd ed. Wiley, New York (1990).

12.- BERTSEKAS, D. P. Tseng, P., "Relaxation methods for minimum cost ordinary and generalized network flow problems", *Operations research* 36 (1988) 93-114.

13.- BOX, G. E. P., Cox, D.R., "An Analysis of transformation" Journal of the royal statistic Soc. 26 (1964) 211-252.

14.- BRADLEY, G., Brown. G., Graves, G., "Design and implementation of large scale primal transshipment algorithms", *Managament science* 21 (1977) 1-38.

15.- BRYSON, N., "Parametric programming and lagrangian relaxation: The case of the network problem with a single side-constraint", *computers and Operations Research* , 18 (2) (1991), 129-140.

16.- BUSAKER, R. G., Gowen, P.J., "A procedure for determining minimal-cost network flow patterns", *ORO technical report 15, Operation Research Office*, Johns Hopkins University, Baltimore, Md (1961).

17.- CALVETE, H. I. and Mateo, P.M., "An approach for the network flow problem with multiple objectives", *Computers and Operations research*, 22 (9) (1989) 64-78.

18.- CHANG, M. D. Chen., C. J., "An improved primal simplex variant for pure processing networks", *ACM transactions on Mathematical software* 15 (1989) 64-78.

19.- CHERIYAN, J., Hagerup, T., Mehlhorn, K., "An O(n3/logn)-Time Maxium-Flow Algorithm ", *SIAM Journal of Computing* 25 (1996) 1144-1170.

20.- CHERIYAN, J., Maheshwari, S.N., "Analysis of Preflow Push Algorithms for Maximum Network Flow", SIAM *Journal of Computing* 18 (1989) 1057-1086.

21.- CUNNINGHAM, W.H., "A Network Simplex Method", *Mathematical Programming* 11 (1976),105-106.

22.- CUNNINGHAM, W.H., "Theoretical properties of the network simplex method", *Mathematics of Operations Research*, 4 (1979), 196-208.

23.- CURRENT, J. and Min, H., "Multiobjective design of transportation networks: taxonomy and annotation", *European Journal of Operational Research* 26 (1986), 187-201.

24.- DANTZIG, G. B.,"Application of the simplex method to a transportation problem", In *activity Analysis and Production and Allocation*, edited by T.C. Koopmans. Wiley, New York (1951) 359-373.

25.- DERIGS, U., Meier, W., "Implementing Goldberg's Max-Flow-algorithm a Computational Investigation", *Methods and Models of Operations Research* 33 (1989) 383-403.

26.- DINIC, E. A., "Algorithms for solution of a problem of Maximun Flow in Networks with Power estimation", *Soviet Mathematical Doklady* 11 (1970) 1277-1280.

27.- EDMONDS, J, Karp, R. M.,"Theoretical Improvements in Algorithmic Efficiency of Network Flow problems". *Journal of ACM* 19 (1972) 248-264.

28.- FERNANDEZ-BACA, D., Martel, C. U.,"On the Efficiency of Maximum_Flow Algorithms on Networks with Small Integer Capacities", *Algoritmica* 4 (1989) 173-189.

29.- FORD, L. R., Fulkerson, D. R., "Maximal Flow through a Network", *Canadian Journal of Mathematics* 8 (1956) 399-404.

30.- FORD, L. R., Fulkerson, D. R., "A primal-dual algorithm for the capacitated Hitchcock problem", *Naval Research Logistics quarterly* 4 (1957) 47-54.

31.- FORD, L. R., Fulkerson, D. R., "Flows in Networks", Princeton University Press (1962).

32.- FULKERSON, D. R.,"An out-of-kilter method for minimal cost flow problems", *SIAM Journal on Applied Mathematics* 9 (1961) 12-27.

33.- FLORES de la Mota, Idalia. *Apuntes de Teoría de Redes.* Universidad Nacional Autónoma de México, 1999.

34.- GABOW,H.N., "Scaling Algorithms for Network Flow Problems", *Journal of computer And System Sciences* 31 (1985) 148-168.

35.- GAL,T., Postoptimal analysis, parametric programming and related topics. McGraw Hill, inc (1979).

36.- GASS,S. and Saaty, T., "The computational algorithm for the parametric objective function", *Naval research Logistics Quarterly* 2(1955) 39-45.

37.- GLOVER,F., Karney, D., Klingman, D., Napier, A., "A computational study on star procedures, basis change criteria and solution algorithm for transportation problem", *Management Science* 20 (1974), 793-813.

38.- GLOVER,F., Karney, D., Klingman, D., "Implementation and computational Comparisons of primal, dual and primal-dual computer codes for minimum cost network flow problem", *Networks* 4 (1974) 191-212.

39.- GOLDBERG, A. V., Tarjan, R. E., "A New approach to the Maximum flow problem", Proc. 18th ACM Symp. On the Theory of Computation (1986) 136-146.

40.- GOLDBERG, A. V., "A New Max-Flow Algorithm", Technical report MIT/LCS/TM-291, Laboratory for Computer Science, MIT, Cambridge, MA (1985).

41.- GOLDFARD, D., Hao, J., "Anti-stalling Pivot Rules for the network simplex Algorithm", *Networks* 20 (1990), 79-91.

42.- GOLDFARD, D., Grigoriadis M D.,"A Computational comparison of the Dinic and Network simplex Methods for Maximum Flow", Annals of Operations Research 13 (1988) 83-123.

43.- GONZALEZ-MARTIN, C., Métodos Interactivos en Programación Multiobjetivo, Secretariado de Publicaciones de la Universidad de La Laguna, serie monografías 24 (1986).

44.- GRIGORIADIS, M. D., "An efficient implementation of network simplex meted", Mathematical Programming study 26 (1986) 83-111.

45.- HU, T. C., "Multi-Commodity Network Flows", *Operations Research* 11 (1963) 344-360

46.- IRI, M., "A new method of solving transportations-network problems", *Journal of The Operations research Society of Japan* 2 (1960) 27-87.

47.- ISERMANN, H., "The enumeration of all efficient solutions for a linear multiple-Objective Transportation problem", *Naval Research Logistics Quarterly* 26 (1979) 123-139.

48.- JEWELL, T. C., "Optimal floe through networks", *interim technical Report* 8, Operations Research Center, MIT, Cambridge, MA (1958).

49.- JOHN, P. W., *Statistical Design and Analysis of experiments*. Macmillan Company (1971).

50.- JOHNSON, E. L., "Networks and basic solutions", *Operations Research* 14 (1966) 619-624.

51.- KARZANOV, A. V., "Determining the Maximal Flow in a Network by the method of Preflows",*Soviet Mathematical Doklady* 15 (1974) 434-437.

52.- KLEIN, M., "A primal method for minimal cost flows with application to the assignment and transportation problems", *Magnagement Science* 14 (1967), 205-220.

53.- KLINGMAN, D. and Mote, J., "Solution approaches for network flow problems with multiple criteria", *Advances in Management Studies* 1 (1982)(1).1-30

54.- KLINGMAN, D.,Napier, A. and Stutz, J., "NETGEN-a program for generating large scale (un) capacitated assignment, transportation and minimum cost flow network problems", *Management science* 20 (1974), 814-822.

55.- LEE, H., Pulat, S., "Bicriteria network flow problems: Continuous case", *European Journal of Oparational Research* 51 (19919, 119-126.

56.- LEE, H., Pulat, S., "Bicriteria network flow problems Integer case", *European Journal of Operational Research* 66 81993), 148-157.

57.- MALHOTRA, R. and Puri, M., C., "Bi-criteria network problem", cahiers du C.E.R.O 26 (1984), 95-102.

58.- MALHOTRA, V. M., Kumar, M. P, Maheshwari, S. N., "An $O(|v|^3)$ Algorithm for finding Maximum flows in Networks". *Inform. Process. Lett*. 7 (1978) 277-278.

59.- MILLIKEN, G. A., Johnson, D. G., Analysis of Messy data, Volume 1: *Designed experiments*. Chapman and hall (1992).

60.- MINTY, G. J., "Monotone Networks", Proceedings of the Royal society of London 257A (1960) 195-212.

61.- MULVEY, J., "Pivot strategies for primal-simplex network codes", *Journal of ACM* 25 (1978) 195-270.

62.- NICOLOSO, S.,Simeone, B., Classical and Contemporary Methods in Network *Optimization, part I: Network flows* (1992)

63.- SAKAROVITCH, M., "Two Commodity Network Flows and Linear Programming", Mathematical Programming 4 (1973) 1-20.

64.- ROCKAFELLAR, R.T. *Network Flows and Monotropic Optimization.* Wiley-Interscience Publication, Washington, EE. UU. 1984.

65.- SEDEÑO-NODA A.,Gonzalez-Sierra, M. G., Gonzalez-Martin, C., "An Algorithmic Study of the Maximum Flow Problem: A Comparative Statistical Analisis". *TOP* 8 (1)(2000)135-162.

66.- SEDEÑO-NODA A.,Gonzalez-Martin, C., "Una variante del algoritmo de Ahuja-Orlin para problemas de flujo Maximo: Experiencias computacionales y comparaciones", Questiió 20 83) (1996) 485-501.

67.-TAHA, Hamdy A. *Investigación de Operaciones.* Alfaomega. Arkansas, EE. UU. 5ª edición, 2003.