TEORÍA DE ALGORITMOS 1
Redes de Flujo

Por: ING. VÍCTOR DANIEL PODBEREZSKI
vpodberezski@fi.uba.ar

1. Problema del Flujo máximo

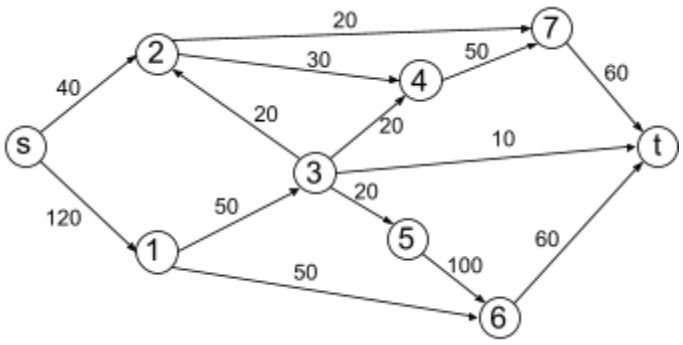
Tanto en tiempos de guerra como de paz la logística y la gestión de la cadena de suministros es de vital importancia. No es de extrañar que el estudio de la optimización de estos sea un campo privilegiado de estudio dentro de la informática desde hace varias décadas. Cómo transportar tropas al frente de batalla, como sabotear el suministro de pertrechos, que puntos de una red de transporte son más propensas a saturarse, cuál es el límite máximo de transporte son algunas de las preguntas frecuentes. Se han construido diferentes modelos y algoritmos para tratar de responderlas. Veremos un modelo simplificado que con algunas adaptaciones puede utilizar en una gran variedad de situaciones.

Llamaremos **red de flujo** a un grafo dirigido G=(V,E) en el que los ejes transportan algún tipo de flujo y en el que los vértices actúan como conmutador de tráfico entre ellos. Solicitaremos que se cumplen las siguientes características:

- Cada eje e ∈ E tiene asociado una capacidad C_e entera positiva que corresponde al valor máximo de tráfico que puede soportar.
- Existe un nodo que llamaremos “Fuente” (source) que puede generar tanto tráfico (flujo) como deseamos.
- Existe un nodo que llamaremos “Sumidero” (sink) que puede absorber tanto tráfico (flujo) como se requiera.
- El resto de los ejes son internos y no generan ni consumen flujo.

El siguiente grafo G=(V,E) está conformado por 9 nodos y 14 nodos. El nodo “s” corresponde a la fuente y puede generar tanto flujo como se le solicite. El nodo “n” es el sumidero y puede recibir

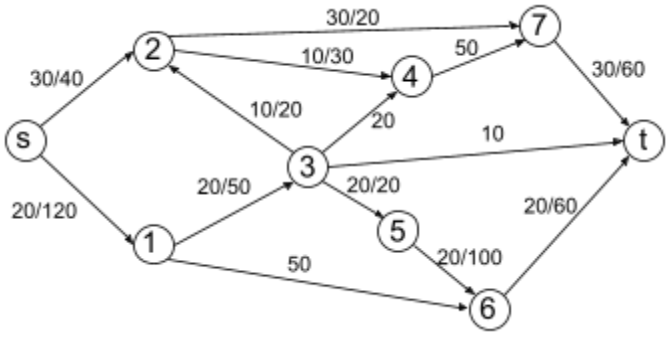
tanto flujo como se desee. Los nodos numerados del 1 al 7 son internos. Funcionan como conmutadores de flujo y distribuyen el flujo que reciben de sus ejes entrantes y salientes. Cada eje tiene definido una dirección en el que el flujo pudo dirigirse y un valor de capacidad máxima que puede transportar. Por ejemplo el eje 1→3 permite trasladar hasta 50 de flujo del vértice "1" al "3". Del nodo "3" salen 4 ejes con capacidades máximas 10,20,20 y 20 respectivamente.



El objetivo de una red de flujo es transportar una cantidad de flujo originada en la fuente y que utiliza los nodos internos y los ejes para llevarlos hasta el sumidero. Dentro de la red no hay pérdidas ni adiciones de flujo. Es decir que la misma cantidad de flujo que se expide es la que llega a su destino. Hablaremos entonces de un **Flujo s-t** como la representación de este transporte. Utilizaremos una función "f" que mapea a cada eje "e" a un valor real no negativo: f("e") → R^+. Este flujo estará restringido a las características del grafo. Observando los ejes y su capacidad solicitaremos que el flujo que pase por cada uno de ellos sea un valor entre cero y la capacidad del mismo. Llamaremos a ésta "**restricción de capacidad**". Observando los nodos internos solicitaremos que para cada uno de ellos la suma de los flujos de los ejes entrantes sea igual a la suma de los flujos de los ejes salientes. Llamaremos a ésta condición de "**conservación de flujo"**.

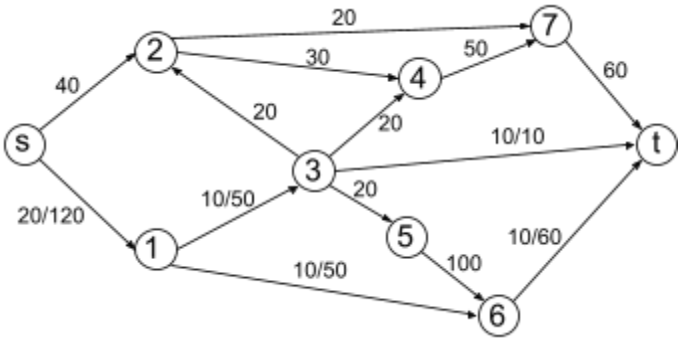
- Conservación de flujo: $\sum_{e=(u,v)} f(e) = \sum_{e=(v,u)} f(e)$ para todo nodo v
- Restricción de capacidad: $0 \leq f(e) \leq c_e$ para todo eje e=(u,v)

Para un determinado grafo con un determinado Flujo s-t tendremos un determinado valor de flujo v(f). Corresponde a la cantidad de flujo total transportada. La podemos medir sumando todos los valores de flujo que transportan los ejes salientes de la fuente. De igual forma se la puede obtener sumando el flujo de los ejes entrantes al sumidero.



Se muestran a continuación dos asignaciones de flujo para el mismo grafo. En la representación, aquellos ejes que tienen flujo están expresados con dos valores separados por una barra. El primer valor es el flujo y el segundo su capacidad. Aquellos ejes que no tiene flujo (flujo cero) solo tiene el valor correspondiente a su capacidad. La primera asignación de flujo corresponde a un flujo s-t no válido. Para constatar debemos verificar que las condiciones de restricción de capacidad y conservación no se cumplen. En primer lugar vemos que el flujo del eje 2→7 tiene un valor de f(2→7)=30. Supera en 10 a su capacidad máxima. En segundo lugar vemos que al nodo "3" llega 20 unidades de flujo por el eje 1→3. Es decir que f(1→3)=20. Mientras que de sus ejes salientes abandona un total de 30 de flujo: f(3→5)=20 y f(3→2)=10. Buscando podemos ver también la existencia de otras incongruencias adicionales.

La segunda asignación de flujo corresponde a una válida. Parte de la fuente 20 de flujo al nodo 1 que se divide por la mitad ingresando a los ejes 3 y 6. Desde estos llegan en igual cantidad al sumidero. Cada eje cumple con la restricción de capacidad y cada nodo con la condición de conservación. El valor del flujo s-t para esta red es v(f)=20.



Llegados a este punto ya podemos expresar el problema a resolver:

Problema del flujo máximo
Dada una red de flujo G=(V,E) con su ejes con una capacidad de transporte c_e, deseamos obtener una asignación de flujo S-T que maximice el transporte de flujo entre la fuente y el sumidero.

En la publicación de Alexander Schrijver "On the History of the Transportation and Maximum Flow Problems"[1] se detallan los primeros años en la búsqueda de la solución óptima a este y otros problemas. Estos primeros tiempos fueron atravesados por los años iniciales de la guerra fría y esto explica la poca publicación sobre el tema. En 1930, A. N. Tolstoi un matemático Ruso publica "Methods of finding the minimal total kilometrage in cargo-transportation planning in space" donde comienza a dar forma al problema y sus posibles estrategias de resolución. Su objetivo estaba de la mano de aprovechar el tendido ferroviario en la unión soviética para maximizar el traslado de suministros. En la otra margen del Océano Atlántico hay un trabajo que comienza a dar frutos en la segunda mitad de la década del 50. Varios autores destacan el trabajo de T. E. Harris y F. S. Ross en 1955. Su interés era también la red ferroviaria soviética, pero con el objetivo de encontrar sus puntos débiles. Su trabajo[2] fue clasificado por muchos años hasta finales del siglo veinte. Una propuesta originada por estos investigadores utiliza la "técnica de inundación" (flooding technique) para resolver el problema. Este corresponde a un algoritmo greedy propuesto por A.W. Boldyreff[3]. La idea era saturar desde la fuente a aquellos ejes que salen con un flujo igual a la capacidad de estos. Luego como una inundación ir derivando este a los ejes siguientes hasta llegar al sumidero. En el camino se podía separar el flujo entre varios ejes o unificar el flujo de varios. Si una cantidad de flujo no se podía entregar (se encontraba un cuello de botella) ese excedente se retornaba a la fuente. La decisión de qué flujo propagar antes y por qué camino se realizaba en forma heurística y tenía en cuenta varias cuestiones. Sin embargo, ningún set de reglas otorgaba un resultado óptimo para cualquier posible instancia del problema.

Una propuesta publicada solo un año después de Manos de L.R Ford y D.R Fulkerson[4] fue la responsable de hallar respuesta óptima para cualquier instancia del grafo. El algoritmo

[1] On the history of the transportation and maximum flow problems,Schrijver, Alexander, 2002 Mathematical Programming 91.3: 437-445.
[2] Fundamentals of a Method for Evaluating Rail Net Capacities,T.E Harris, F. S Ross, 1955, Research Memorandum RM-1573, The RAND Corporation, Santa Monica, California
[3] Determination of the Maximal Steady State Flow of Traffic through a Railroad Network, A.W Boldyreff, 1955, Research Memorandum RM-1532, The RAND Corporation, Santa Monica, California, [publicado en Journal of the Operations Research Society of America 3 (1955) 443–465].
[4] Maximal Flow through a Network L.R Ford, D.R Fulkerson, Research Memorandum RM-1400, The RAND Corporation, Santa Monica, California, [19 Noviembre de] 1954 [publicado en Canadian Journal of Mathematics 8 (1956) 399–404].

hoy es conocido por el apellido de sus autores Ford-Fulkerson. En 1962 publican el libro Flows in Networks[5] donde extienden sus resultados a varios problemas derivados

2. Ford Fulkerson

El algoritmo de Ford-Fulkerson corresponde a un proceso mejora incremental (iterative improvement) ideado para hallar el flujo máximo de una red de flujos. Comienza con un flujo s-t nulo (cero). Progresivamente busca aumentarlo construyendo soluciones intermedias factibles. Se entiende a la factibilidad en este contexto como el cumplimiento de las restricciones impuestas por el problema (restricción de capacidad y de conservación). El paso inicial (con flujo igual a cero) cumple con qué ninguna capacidad de los ejes es excedida y con que ningún nodo interno tiene un desbalance entre los flujos entrantes y salientes.

Para aumentar el flujo busca un camino que inicia en el nodo fuente y concluye en el sumidero utilizando ejes que tenga al menos parte de su capacidad sin utilizar. Este camino se conoce como **camino de aumento**. Buscará transportar la mayor cantidad de flujo posible que no exceda la máxima disponibilidad de los ejes por lo que pasa. Aquel eje que restringe el máximo en el camino se lo conoce como **cuello de botella**. El proceso se repite hasta que no existen caminos de aumento. Llamaremos **ejes saturados** a aquellos que no tiene remanente de capacidad para utilizar en otros caminos de aumento. No existe forma de aumentar el flujo cuando cualquier camino posible entre la fuente y el sumidero debe pasar por algún eje saturado. Esto da por finalizado el proceso.

¿Es la solución obtenida óptima (la solución factible de mayor valor de flujo posible)? Lamentablemente la respuesta es negativa. Ford y Fulkerson se percataron de que una serie de elecciones desafortunadas podría llevar a agotar los caminos de aumento antes de maximizar el resultado buscado. Por lo tanto, para superar el inconveniente, proponen la creación de un grafo modificado como primer paso de su algoritmo. Llamaremos a este "Grafo residual".

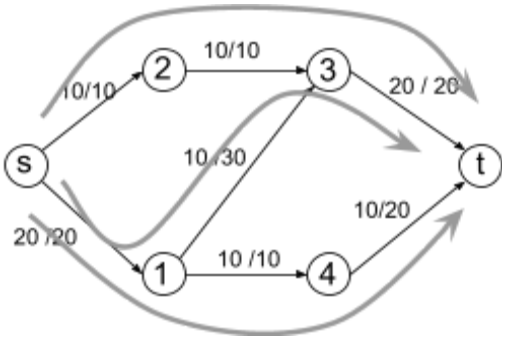
Deseamos maximizar el flujo en el grafo de la imagen. Inicialmente no tendremos flujo saliendo de la fuente. Buscamos un camino de aumento. Proponemos el conformado por s→1→3→t. Las

[5] Flows in Networks, L.R Ford, Jr., D.R Fulkerson, 1962, Princeton University Press, Princeton, New Jersey

capacidades disponibles de los ejes por los que pasa el camino son 20,30 y 20. El menor valor entre ellos corresponde al cuello de botella. Podemos por lo tanto transportar por ese camino 20 de flujo. Esta corresponde a la nueva solución factible en la que se está transportando al momento 20 unidades de flujo.

En este punto no es posible llevar más flujo desde la fuente al sumidero. El eje s→1 está saturado. El eje s→2 podría transportar hasta 10 unidades. De igual manera podría hacerlo el eje 2→3. Sin embargo el único camino de aumento que se podría construir para llegar al sumidero requiere el eje 3→t que está saturado.

Una elección diferente del camino inicial podría habernos acercado a un flujo s-t de v(f)=30 unidades utilizando los caminos s→2→3→t (10 unidades), s→1→4→t (10 unidades) y s→1→3→t (10 unidades). En la imagen se puede observar esta configuración óptima.



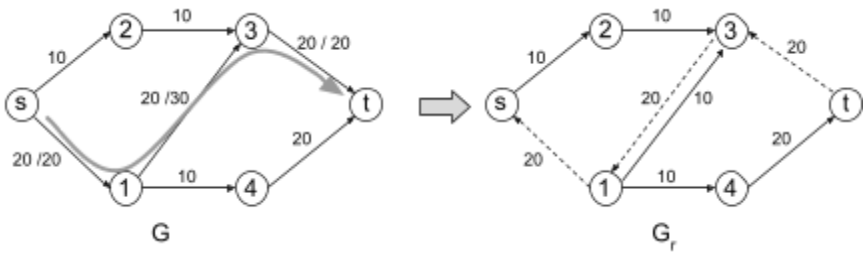
La definición del **grafo residual** G_r se realiza en base a la red de flujo G y el flujo s-t que lo atraviesa. Es un nuevo grafo que comparte los vértices del original. Además, por cada eje en G puede llegar a tener uno o dos en G_r dependiendo de la cantidad de flujo que pasa por este y su capacidad. En este nuevo grafo también los ejes tendrán capacidades. El proceso de creación lo podemos detallar de la siguiente manera:

- Por cada vértice $v \in V$ del Grafo G creamos un nuevo vértice v' en el grafo G_r.
- Por cada eje $e=(u,v) \in E$ del grafo G en el que el flujo f(e) que pasa por él es menor a su capacidad C_e, creamos un nuevo eje $e'=(u,v)$ con capacidad C_e-f(e). Llamaremos a estos ejes "**ejes hacia adelante**" y su valor de capacidad representa cuánto flujo aún puede trasladarse desde u hasta v.
- Por cada eje $e=(u,v) \in E$ del grafo G en el que el flujo f(e) es mayor a cero, creamos un nuevo eje e"=(v,u) con capacidad f(e). Llamaremos a estos ejes **"ejes hacia atrás"**

y su valor de capacidad representa cuánto flujo está pasando actualmente por el eje "e" del grafo G.

Se puede observar que si el flujo s-t en una red de flujo G es igual a cero, su grafo residual G_r es equivalente a G.

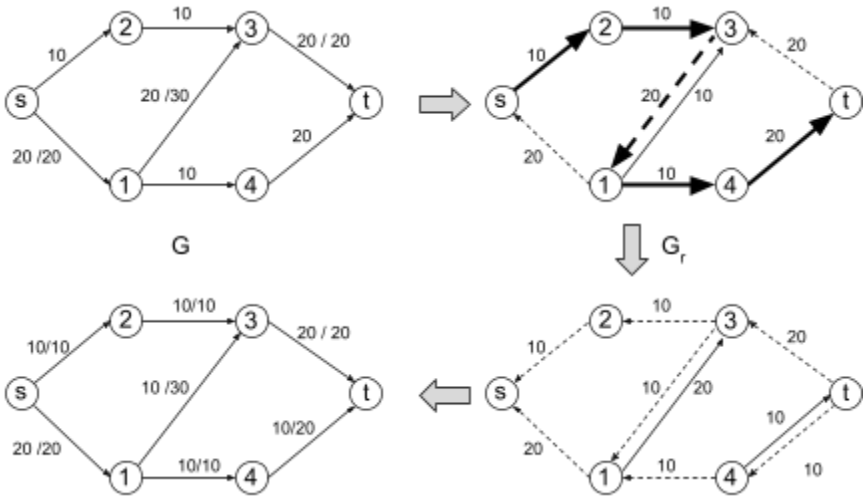
Para la red de flujo de la imagen con un flujo s-t de 20 procedemos a construir su grafo residual correspondiente. Los vértices se mantienen. El eje s→2 tiene capacidad 10 y no tiene flujo: f(s→2)=0. En el grafo residual se crea únicamente un eje hacia adelante con capacidad 10. El eje s→1 tiene capacidad 20 y un flujo f(s→1)=20. Se crea un eje hacia atrás 1→s con capacidad 20. Conceptualmente corresponde a el flujo que se envía desde s a "1". No se crea un eje hacia adelante por no existir una capacidad residual a utilizar por otros camino de aumento. El eje 1→3 se desglosa en un eje hacia adelante y uno hacia atrás. La capacidad del eje hacia adelante es 10. Se calcula como su capacidad (30) menos el flujo que actualmente lo atraviesa (20). La capacidad del eje hacia atrás es 20. Corresponde al flujo que lo atraviesa. De igual manera se procesan el resto de los ejes.



El algoritmo de Ford-Fulkerson busca la existencia de caminos de aumento utilizando el grafo residual. Gracias a su estructura permite reencauzar flujo de ciertos ejes a otros. Una elección de un camino de aumento desafortunada puede ser subsanada en una iteración posterior. Un camino puede utilizar tanto ejes e' hacia adelante como ejes e" hacia atrás. Supongamos un camino de aumento lleva "w" unidades de flujo y recorre tanto ejes hacia adelante como hacia atrás. Al utilizar un eje e'=(u,v) hacia adelante indica que se aumenta el flujo en "w" en el eje e=(u,v) del grafo G. Al utilizar un eje e"=(v,u) hacia atrás indica que se reduce "w" de flujo del eje e=(u,v) del grafo G. Esta reducción corresponde a un reencauzamiento.

Continuando con el ejemplo anterior en el grafo G se puede encontrar el camino de aumento s→2→3→1→4→t en el grafo residual G_r. Su cuello de botella permite llevar 10 unidades de flujo. Todos los ejes del camino, excepto uno, son hacia adelante. Al pasar el flujo por estos

aumentan en 10 el caudal de sus correspondientes ejes del grafo G. El eje 3→1 corresponde al eje hacia atrás. Su utilización reduce en 10 el flujo del eje 3→1 del grafo G. El resultado produce que 10 de flujo pasaban del vértice 1 al 3 dejen de hacerlo. Esto permite que esa misma cantidad pueda llegar desde el vértice 2 al 3 y desde allí al vértice t. Los 10 de flujo que antes se enviaban del vértice 1 al 3 se encauzan y se envían al vértice 4 y de este a t.



En el siguiente pseudocódigo se muestra el proceso de agregar flujo a través de un camino de aumento obtenido del grafo residual:

Aumento de flujo por camino de aumento P

```
Sea P el camino de aumento
Sea f el flujo actual en la red de flujo G (con su correspondiente grafo residual Gr)
Sea w el cuello de botella de P
Para cada eje e=(u, v) ∈ P
        Si e = (u, v) eje hacia adelante
                f(e)+=w en G
        Sino si es eje para atrás
                e' = (v, u)
                f(e')-=b en G
Retornar f
```

El camino de aumento utilizando el grafo residual aumenta el flujo s-t total del grafo G y obtiene una nueva solución factible. Podemos comprobarlo verificando que aun se cumplen las condiciones de conservación y de restricción de capacidad.

Para validar la restricción de capacidad, consideremos que el camino de aumento adiciona un valor de w al flujo s-t.

- Un eje del camino e'=(u,v) hacia adelante tiene como capacidad c_e-f(e) y sabemos que el flujo w tiene que ser menor a este valor o igual (en el caso que este eje corresponda al cuello de botella). El aumento de flujo provocará que la capacidad residual disminuya y que el flujo que pasa por e sea menor o igual a la capacidad c_e del eje e:

  $0 \leq f(e) \leq f(e)+w \leq f(e)+(c_e-f(e)) \leq c_e$

- Un eje del camino e"=(v,u) hacia atrás tiene como capacidad f(e) que es menor o igual a C_e. Sabemos que el flujo w tiene que ser menor a este valor o igual por ser parte del camino de aumento. El incremento del valor en e" indica que el eje e=(u,v) disminuye en w su flujo:

  $c_e \geq f(e) \geq f(e)-w \geq f(e)-f(e) \leq 0$

Para validar la condición de conservación, identificamos que los únicos vértices que sufren cambio de flujo por su interior son aquellos que pertenecen al camino de aumento. Consideremos cualquier vértice interno "i" del mismo. Este vértice tendrá un eje que llega a él y otro que lo abandona pertenecientes al camino. Cada uno de ellos podría ser hacia atrás o hacia adelante. Si el eje ingresante es hacia adelante indica que aumenta en w el flujo incidente a "i". Si es hacia atrás indica que w de flujo que previamente lo abandonaba deja de hacerlo. En ambos casos tenemos un incremento transitorio de w en el nodo. Si el eje saliente es hacia adelante indica que abandona w flujo en "i". Si el eje saliente es hacia atrás indica que w de flujo que previamente ingresaba al nodo deja de hacerlo.En ambos casos tenemos un decremento transitorio de w en el nodo. Cualquiera sea el tipo de eje ingresante y saliente, uno adiciona y el otro disminuye la misma cantidad de flujo. Por lo tanto se cumple la conservación de flujo.

Podemos resumir el algoritmo de Ford-Fulkerson con el siguiente pseudocódigo:

Ford Fulkerson

```
Inicialmente el flujo f(e) = 0 para todo eje e en el grafo G
Inicializar Gr grafo residual según G y flujo s-t f = 0
Mientras haya un canino s-t en Gr
        Sea P un camino s-t en Gr
```

Sea w el valor de flujo del cuello de botella del camino P

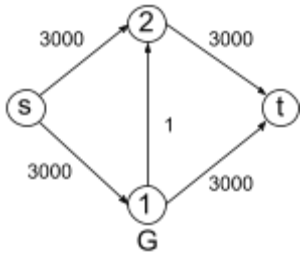
Actualizar el flujo w en el camino P en G
Actualizar Gr

Retornar f

Restan responder algunas cuestiones importantes acerca del algoritmo. Nos detendremos ordenadamente cada una de ellas ¿Podemos esperar que el algoritmo finalice en un número finito de pasos?. La siguiente demostración nos dará una respuesta afirmativa.

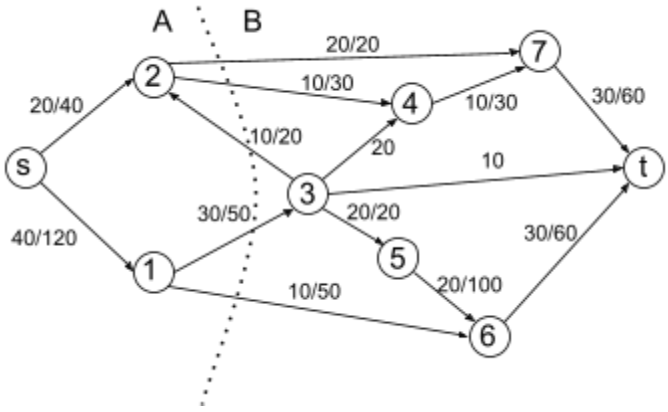
Sea el subconjunto S de ejes e=(s,u) salientes de la fuente. Llamamos C a la suma de sus capacidades: $C = \sum_{e \in S} c_e$. Cada camino de aumento P - obtenido en el grafo residual - corresponde a un camino que inicia en la fuente y finaliza en el sumidero. El primer eje de P corresponde a un eje, que sale de la fuente "s", hacia adelante e'=(s,v) y ninguno de los posteriores es un eje hacia atrás e"=(u,s). De serlo, tendríamos un ciclo regresando a la fuente que podemos quitar sin afectar la cantidad de flujo enviado. Cada capacidad de los ejes es un valor entero positivo, y también lo será en consecuencia el cuello de botella de P. Su valor mínimo es la unidad. Por lo tanto en el peor de los casos disminuye en 1 algunos de los ejes en S. Dado que C es un valor entero positivo finito, que en cada iteración se consume al menos en uno, tendremos como mucho C caminos de aumento. Al acabar los caminos de aumento, el algoritmo finaliza en un número finito de pasos.

Para el grafo de la siguiente imagen vemos que existen 2 ejes con capacidad 3.000 cada uno que salen de la fuente. Su suma C=6000. Una inspección rápida nos permitirá establecer que se podría llegar al resultado óptimo con solo dos caminos de aumento s→1→t y s→2→t . Sin embargo, Ford-Fulkerson no busca caminos de aumento en un orden particular y podría encontrar otros. En el peor de los casos, cada camino de aumento utiliza alternativamente el eje hacia adelante y hacia atrás que une los vértices 1 y 2. En estos caminos el cuello de botella es 1. En cada uno de ellos restará alternativamente la capacidad de s→1 y s→2 en uno. Tendremos entonces en el peor de los casos 6.000 caminos de aumento antes de finalizar.



Queremos asegurarnos que el resultado obtenido al ejecutar Ford-Fulkerson para cada instancia de una red de flujo nos retorna el resultado óptimo. Para eso requerimos de algunas definiciones adicionales que nos serán de utilidad en la demostración

Sea un grafo grafo G=(V,E) por el que atraviesa un flujo f. Llamaremos **corte s-t** a una partición de los nodos en dos subconjuntos disjuntos. El conjunto A contará con el nodo fuente y cero o más nodos internos. El conjunto B contará con el nodo sumidero y el resto de los nodos internos no incluidos en A. Existen un subconjunto de ejes e=(u,v) en el grafo G que conectan vértices de A con los B. Algunos de ellos e=(a,b) tendrán sentido de A a B y otros e=(b,a) de B a A. Debido a la propiedad de conservación de flujo sabemos que todo el flujo que sale de la fuente llega al sumidero. Si por el grafo circula un total de flujo f, este debe atravesar los ejes del corte. Llamaremos $f(A)_{out}$ al flujo que pasa por los ejes e=(a,b). Asimismo llamaremos $f(A)_{in}$ al flujo que pasa por los ejes e=(b,a). Como ninguno de los vértices de B genera flujo, $f(A)_{in}$ debe ser flujo que previamente llegó de A y regresa a A. Por lo tanto el flujo total que circula en el grafo, lo podemos calcular utilizando el corte como $f(A)_{out} - f(A)_{in}$.



En el grafo de la imágen se muestra un Corte s-t. En el subconjunto A se incluyen la fuente, el vértice 1 y el 2. En el subconjunto B están el resto. Los ejes que salen de A a B son 1→3, 1→6, 2→4, 2→7. El valor de flujo $f(A)_{out}$ se puede calcular como $f_{1\to3} + f_{1\to6} + f_{2\to4} + f_{2\to7}$=30 + 10 + 10 + 20 = 70. El valor de flujo $f(A)_{in}$ se puede calcular como $f_{3\to2}$ = 10. Por lo tanto el flujo s-t de la red G tiene un valor de $v(f)=f(A)_{out} - f(A)_{in}$ = 70 -10 = 60. Cualquier otro corte s-t nos dará el mismo valor final.

Llamaremos **capacidad del corte s-t** c(A,B) a la suma de las capacidades de los ejes e=(a,b) cuyo origen corresponde a un vértice del subconjunto A y su destino a un vértice del subconjunto B. Existe una relación entre el flujo salientes de A y la capacidad del corte s-t. Podemos expresarlo mediante la siguiente desigualdad:

$$v(f) = f(A)_{out} - f(A)_{in} \leq f(A)_{out} = \sum_{e=(a,b)} f(e) \leq \sum_{e=(a,b)} c(e) = c(A,B)$$

Recordemos que el flujo en un eje es igual o menor a su capacidad. Entonces podemos concluir que la capacidad del corte será mayor o igual al flujo s-t que circula por el grafo.

Tomemos el grafo residual G_f resultante luego de ejecutar el algoritmo de Ford-Fulkerson. Buscaremos el subconjunto de vértices A* que son accesibles desde la fuente. Sabemos que al menos el sumidero no estará entre ellos. De lo contrario existiría un camino de aumento y el algoritmo no hubiese terminado en esa situación. Llamaremos B* al complemento de A* (aquellos no accesibles desde la fuente). Estos dos subconjuntos conforman el corte s-t c(A*,B*).

Existirán un conjunto de ejes e=(a,b) en el grafo G que conectan vértices del subconjunto A* con el B*. Cada uno de estos deben estar saturados. Es decir que c_e = f(e) de lo contrario existiría en G_r un eje hacia adelante e'=(a,b) con un valor de capacidad c_e - f(e) mayor a cero y sería falso que el vértice b es inaccesible desde la fuente.

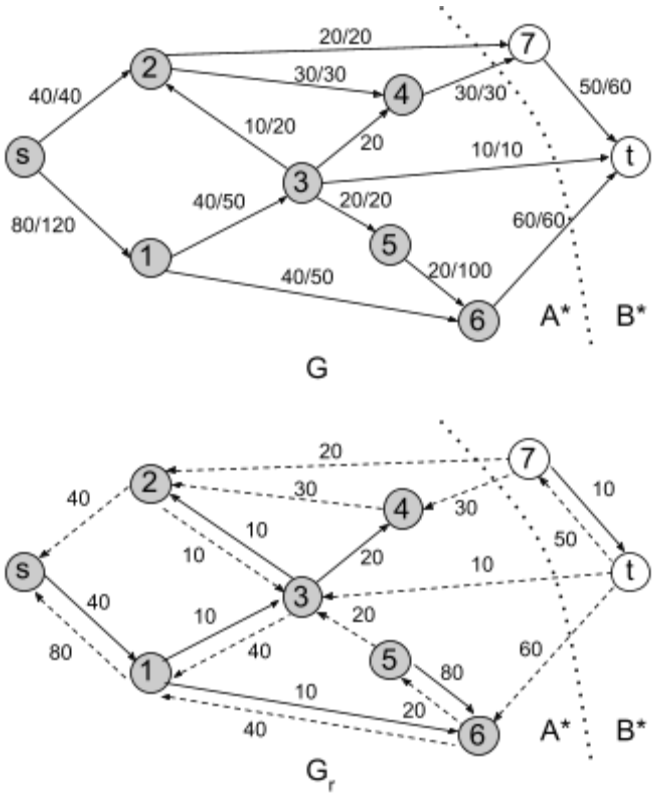
Existirán un conjunto de ejes e=(b,a) en el grafo G que conectan vértices del subconjunto B* y con A*. Cada uno de estos no debe tener flujo a través de ellos. Es decir f(e) = 0 de lo contrario existiría en G_r un eje hacia atrás e"=(a,b) con un valor de capacidad f(e) mayor a cero y sería falso que el vértice b es inaccesible desde la fuente.

Podemos concluir entonces que todos los ejes de A* a B* están saturados y que todos los ejes de B* a A* están sin utilizar. Si nos basamos en el fórmula obtenida para la capacidad del corte tendremos:

$$v(f) = f(A^*)_{out} - f(A^*)_{in} = f(A^*)_{out} - 0 = \sum_{e=(a,b)} f(e) = \sum_{e=(a,b)} c(e) = c(A^*, B^*)$$

Por lo tanto al finalizar la ejecución de Ford-Fulkerson tendremos un corte c(A*,B*) cuya capacidad de corte es igual al flujo s-t v(f) de G. Como cualquier flujo en G tiene que atravesar este corte para llegar al sumidero, este restringe el valor del flujo máximo. Entonces el flujo v(f) luego de ejecutar ford fulkerson corresponde al flujo máximo de

cualquier flujo en G. Más aún, el corte c(A*,B*) tiene la capacidad mínima de cualquier corte s-t en G. Llamaremos c(A*,B*) **corte mínimo**.



Se muestra el grafo G luego de ejecutar Ford Fulkerson. Observando los ejes que salen de la fuente se puede contabilizar un total de v(f)=120 de flujo. Utilizando el grafo residual se puede encontrar cuales son los vértices accesibles desde la fuente. Estos son 1,2,3,4,5 y 6. Junto a la fuente corresponden al subconjunto A. El complemento de este conjunto B* es el sumidero y el vértice 7. Los ejes entre los conjuntos son 2→7, 3→t, 4→7, 6→t. La capacidad de este corte es C(A*,B*)= $c_{2\to7}$ + $c_{3\to t}$ + $c_{4\to7}$ + $c_{6\to t}$ = 20 + 10 + 30 + 60 = 120. Como se puede ver el flujo de la red v(f) es igual a la capacidad del corte C(A*,B*). Por lo tanto el flujo obtenido por ford fulkerson corresponde al flujo máximo de la red y C(A*,B*) corresponde al corte mínimo.*

Considerando las definiciones y demostraciones pasadas podemos ahora afirmar que la ejecución de Ford-Fulkerson se realizará en un tiempo finito de pasos y que al concluir tendremos el flujo máximo del grafo.

En este punto está pendiente determinar la complejidad de Ford-Fulkerson. Afecta este proceso la forma de almacenar el grafo residual y del algoritmo seleccionado para buscar el camino de aumento. Hay varias alternativas para buscar un camino s-t. Entre ellos camino aleatorio, camino de mayor cuello de botella, camino de menor longitud, etc. Comencemos con un algoritmo conocido que nos permite encontrar si existe un camino entre la fuente y el sumidero: Depth First Search (DFS). Su complejidad temporal corresponde a O(V+E) si se utiliza como estructura interna para almacenar al grafo una lista de adyacencias o $O(V^2)$ cuando se utiliza una matriz de adyacencia. Tendremos en el caso extremo en el grafo residual el doble de la cantidad de los ejes del grafo (los ejes hacia

adelante y hacia atrás). El tamaño del camino de aumento tiene como mucho una longitud de V-1 ejes. Se debe modificar las capacidades al aumentar el flujo de estos ejes. Proceso que se puede realizar en O(V). Buscar caminos de aumento y actualizar su flujo lo tendremos que realizar a lo sumo C veces (suma de las capacidades de los ejes que salen de la fuente). Tenemos entonces una complejidad pseudo polinomial de O(C*(costo de construir el camino de aumento+actualizar el flujo)). Con lista de adyacencias corresponde a O(C*(E+V)). En ocasiones - sobre todo si la cantidad de ejes es mayor a los vértices se suele expresar la complejidad como O(CE).

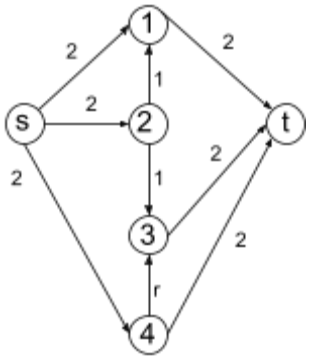
La importancia del corte mínimo radica no sólo como parte de la demostración de optimalidad del algoritmo de Ford-Fulkerson. Encontrar el corte mínimo corresponde a un problema de interés por sí mismo. Este brinda información importante sobre la topología de la red de flujo. El corte mínimo c(A*,B*) nos señala cuales son los ejes cuya pérdida o reducción de capacidad afectan irremediablemente el flujo máximo de la red. Podemos pensar en el corte mínimo como el eslabón más débil de la red. Puede existir más de un corte mínimo. Por lo que aumentar la capacidad de un eje que atraviesa el corte encontrado no necesariamente aumenta el flujo máximo de la red.

Para obtener el corte mínimo, se puede calcular el flujo máximo y analizar el grafo residual resultante. Partiendo desde el sumidero se deben obtener todos los vértices accesibles desde este. Esto corresponde a un problema de conectividad del grafo. Podemos utilizar nuevamente DFS para resolverlo. Esto conformará el subconjunto A* y su complemento el B*. La complejidad de lograrlo es la suma de ambos procesos O(C*(E+V) + (E+V)). Podemos simplificarlo como O(C*(E+V)).

Antes de finalizar debemos revisar una cuestión adicional. Al determinar la función de flujo se definió que la función de flujo f(“e”) pertenecía al espacio de los números reales. Sin embargo, al definir la capacidad de los ejes se especificó que correspondía a un valor entero positivo. Al demostrar la finalización de la ejecución de ford fulkerson en un número finito de pasos se determinó que el mínimo valor de flujo por iteración incrementado es la unidad. Esto modifica las capacidades residuales de los ejes dentro del camino de aumento y asegurando sus valores como enteros positivos o cero. Pero, ¿qué pasaría si las

capacidades también son reales? Mediante un simple ejemplo veremos que Ford-Fulkerson podría no finalizar. El mismo es una variante al propuesto por Uri Zwick[6]

Sea el grafo G correspondiente al de la imagen. Todos sus ejes tienen capacidad entera, excepto el 4→3 de valor $r = (\sqrt{5} - 1)/2$. *Se puede observar que el flujo máximo existe y corresponde a 5. Lo alcanzaremos por los caminos de aumento s→1→t (cuello de botella = 2), s→2→3→t (cuello de botella = 1) y s→4→t (cuello de botella = 2). El grafo residual resultante mostrará que no quedan caminos de aumento s-t disponibles. Sin embargo, la elección inadecuada de los caminos de aumento nos llevará a una iteración infinita del algoritmo. Veamos la manera en que esto ocurra. Definiremos los caminos* p_1=s→4→3→2→1→t, p_2=s→2→3→4→t *y* p_3=s→1→2→3→t. *Realizaremos como primera iteración el camino s→2→3→t con cuello de botella 1. Luego realizaremos los caminos de aumento p1,p2,p1,p3 con respectivamente flujo* r^1,r^1,r^2 *y* r^2. *En ese momento tendremos un flujo s-t de* $1+2r+2r^2$. *Además todas las capacidades serán reales excepto 2→3 que estará en cero. Como aun existen caminos de aumento podremos continuar. Repetiremos de forma continua la secuencia de caminos de aumentos p1,p2,p1,p3 y aumentaremos progresivamente el flujo total transportado. Aunque este será menor en cada ocasión. La tabla realiza un resumen de las primeras iteraciones del algoritmo. En cada iteración los 3 ejes internos del grafo tendrán capacidades de la forma 0,* r^a, r^{a+1} *con un valor de "a" entero positivo. Al iterar veremos que el flujo enviado converge a* $1 + 2\sum_{i=1}^{\infty} r^i = 3 + 2r$. *Reemplazando r,* $2 + \sqrt{5}$ *que es menor al flujo máximo posible calculado 5.*



paso	Camino de aumento	flujo	2→1	4→3	2→3
0			r^0=1	r	1
1	s→2→3→t	1	r^0	r^1	0
2	p_1	r^1	r^2	0	r^1
3	p_2	r^1	r^2	r^1	0
4	p_1	r^2	0	r^3	r^2
5	p_3	r^2	r^2	r^3	0
...					

De esa forma podemos ver que el algoritmo de Ford Fulkerson no nos asegura la convergencia a la solución óptima y no nos asegura su finalización si las capacidades iniciales son números reales.

[6] "The smallest networks on which the Ford-Fulkerson maximum flow procedure may fail to terminate", Uri Zwick, 21 Agosto 21 1995, Theoretical Computer Science Volumen 148 1 pp 165–170

El flujo máximo existe, sin embargo una elección desafortunada de caminos de aumento, nos llevaría a un caso de no finalización del algoritmo. Caso diferente corresponde al uso de capacidades racionales. Dado que el valor de las mismas se pueden multiplicar todas por un factor para lograr que los valores sean enteros. Una vez hallado el flujo solo resta dividirlo por ese factor para conocer su valor.

3. Flujo máximo en tiempo polinomial

La complejidad pseudo-polinomial de Ford-Fulkerson en ciertas situaciones puede convertirse en un inconveniente para su elección. Si los valores de capacidad son muy grandes comparado al tamaño de la red el tiempo de ejecución podría ser bastante elevado. Afortunadamente existen diversos algoritmos que son polinomiales que se pueden implementar en su lugar. Algunos de ellos no son más que una variante de Ford-Fulkerson. Iterativos e incrementales. Estos definen una manera específica de buscar el próximo camino de aumento. Posiblemente los más antiguos y conocidos entre ellos corresponden a Dinic[7] de 1970 y Edmonds-Karps[8] de 1972. Se basan en buscar como próximo camino de aumento al camino de menor longitud disponible (shortest path).

A continuación detallaremos la propuesta de **Dinic** y su demostración de optimalidad.

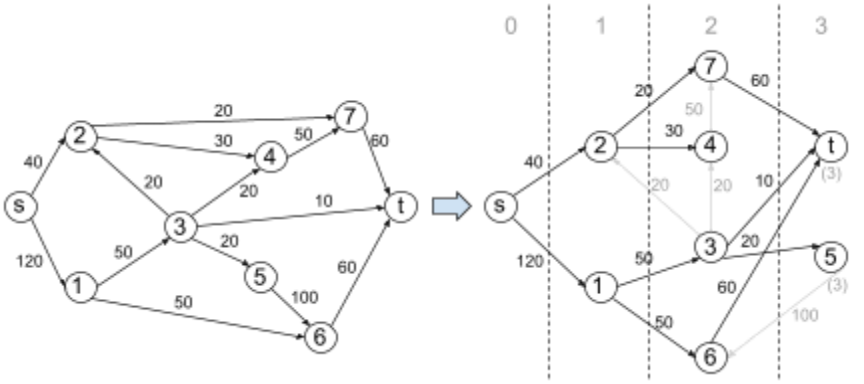
Llamaremos **nivel del vértice** v en el grafo residual Gr o nivel(v,G_r) a la longitud del camino mínimo desde la fuente al mismo utilizando el Gr.

Llamaremos **grafo de nivel** (Level Graph) L_g=(V,E”). a un subgrafo de G_r=(V,E’). Con V los nodos de la red de flujo y E’ los ejes hacia atrás y hacia adelante del grafo residual. A cada nodo “v” ∈ V se le asigna un nivel(v,G_r) . Únicamente se incluyen en E” aquellos ejes e=(u,v) ∈ E’ tal que nivel(v,G_r) = nivel(u,G_r) + 1. Es decir que se remueven los ejes laterales (que conectan nodos del mismo nivel) y inversos (que conecta a un nodo de un nivel con otro de un nivel inferior). El grafo de nivel carece de ciclos. Esta misma estructura es llamada por un conjunto de la literatura sobre el tema (entre ellos por el autor) como **red en capas** (layered network). Calcularemos el grafo de nivel de un grafo residual mediante Breadth

[7] Algorithm for solution of a problem of maximal flow in a network with power estimation, E. A. Dinic, 1970, Soviet Math. Doklady, 11:1277-1280
[8] Theoretical improvements in algorithmic efficiency for network flow problems, Edmonds Jack; Karp Richard M., 1972, Journal of the ACM. 19 (2): 248–264

First Search (BFS). Dejando únicamente aquellos ejes según la definición. Lo podemos hacer en O(V+E).

El grafo de la imagen corresponde al grafo residual de una red de flujo en su primera iteración. Calculamos el grafo de nivel aplicando BFS. Tendremos el nivel de cada vértice o la longitud del camino más corto desde la fuente. Por ejemplo el nodo 4 tiene un nivel de 2. El camino mínimo a 4 desde la fuente utiliza dos ejes: s→2→4. Incluiremos únicamente aquellos ejes que conectan vértices de un nivel con otro del siguiente nivel. Existen tres caminos alternativos de aumento de longitud 3.



En el grafo de nivel tendremos únicamente caminos de aumento s-t de la menor longitud posible. Los utilizaremos para aumentar el flujo del grafo residual y eliminar del grafo de nivel aquellos ejes saturados (los cuellos de botella del camino de aumento). Para buscar los caminos de aumento (correspondientes a los de menor longitud) podemos utilizar Depth First Search (DFS) desde la fuente a cada uno del resto de los nodos mediante el grafo de nivel. Nótese que al no existir ni ciclos ni ejes que no se conecten a vértices de menor o igual nivel es imposible encontrar caminos de mayor longitud a la buscada. Lo podemos hacer en O(V+E).

En cada aumento de flujo utilizando un camino p al menos un eje se satura y por lo tanto se remueve del gráfico de nivel. Es por esto que este proceso lo podremos hacer a lo sumo O(E) veces. Además cada camino de aumento agrega, como mucho E ejes en el sentido opuesto del camino en el grafo residual. Esos ejes en el sentido contrario no pueden contribuir a que los caminos de aumento posteriores sean de menor longitud. Puesto que unen ejes de mayor nivel con otro de menor nivel.

Se conoce como **flujo de bloqueo** (blocking flow) al instante en el que todo camino de aumento en el grafo de nivel tiene un eje saturado. Esto impide seguir aumentando el flujo. Para continuar, se vuelve a generar el grafo de nivel pero con el grafo residual modificado por los últimos caminos de aumento. En cada nuevo grafo de nivel la longitud del camino

mínimo aumenta en al menos uno. Por lo tanto, teniendo V vértices la cantidad máxima de grafo de nivel a construir es O(V). El algoritmo culmina cuando en el cálculo del grafo de nivel el sumidero no es accesible desde la fuente.

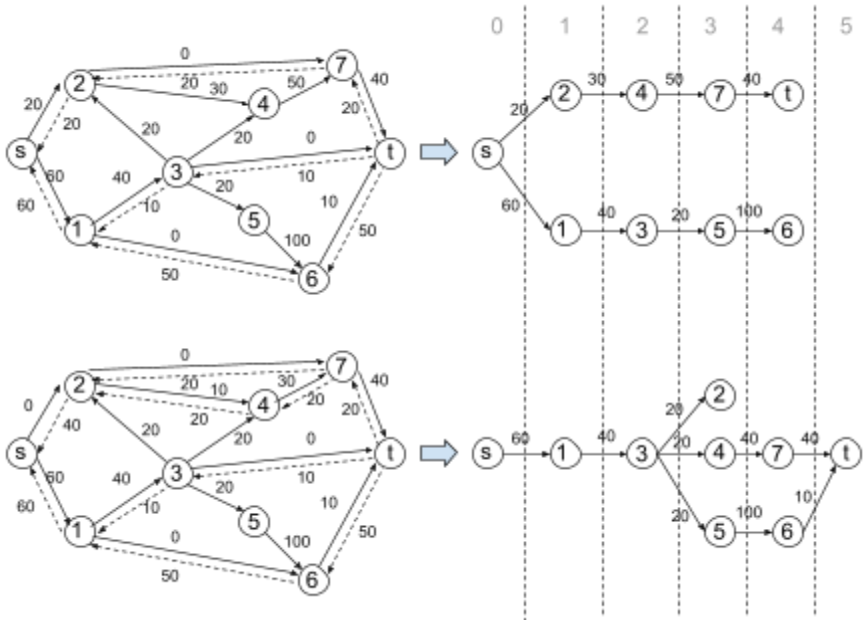
El pseudocódigo teniendo en cuenta la propuesta de Dinic es:

Flujo máximo - Dinic (*)

```
Inicialmente el flujo f(e) = 0 para todo eje e en el grafo G
Inicializar Gr grafo residual según G y flujo s-t f = 0
Obtener el grafo de nivel Lg en base a Gr
Mientras sea accesible el sumidero desde la fuente en Lg
        Mientras exista un camino de aumento en Lg
                Sea P un camino s-t en Lg
                Sea w el valor de flujo del cuello de botella del camino P
                Actualizar el flujo w en el camino P en G
                Actualizar Lg y Gr
        Obtener el grafo de nivel Lg en base a Gr
Retornar f
```

Unificando todo el proceso podemos calcular la complejidad del algoritmo como O(V)*(O(V+E)+O(E)*O(V+E)). En forma resumida $O(E^2V)$ que corresponde a un algoritmo polinomial. Sabemos que el resultado es óptimo además por lo demostrado en Ford-Fulkerson: al finalizar no quedan caminos de aumento entre la fuente y el sumidero.

Siguiendo con el ejemplo anterior, podemos encontrar en el grafo de nivel tres caminos de aumento de longitud 3: s→1→3→t, s→1→6→t y s→2→7→t. Con cada camino de aumento se elimina un eje saturado del grafo de nivel y se actualiza el grafo residual. Luego de estos tres aumentos se llega al flujo de bloqueo. Se vuelve a calcular el grafo de nivel. En esta nueva iteración tendremos en el grafo de nivel solo un camino de aumento de longitud 4: s→2→4→7→t. Se aumenta el flujo según el cuello de botella y al llegar al flujo de bloqueo se vuelve a calcular el grafo de nivel. Se tendrán dos caminos de aumento de longitud 5. Se continuará hasta que no exista en el grafo de nivel un camino desde la fuente al sumidero. En la imagen se muestra lo descrito.



El algoritmo de **Edmonds-Karp** prescinde del grafo de nivel. Su funcionamiento es más simple. En rigor es el mismo algoritmo que Ford-Fulkerson, excepto que en vez de tomar un camino de aumento sin criterio específico, toma el camino disponible de menor longitud. Para eso utiliza BFS desde la fuente generando el camino de menor longitud posible.

Al realizar el aumento satura el cuello de botella del camino al actualizar el grafo residual. Al igual en con Dinic, cada camino de aumento agrega, como mucho E ejes en el sentido opuesto del camino en el grafo residual. Esos ejes en el sentido contrario no pueden contribuir a que los caminos de aumento posteriores sean de menor longitud.

En pseudocódigo, en un alto nivel, se puede expresar el algoritmo de la siguiente manera:

Flujo máximo - Edmonds Karp

```
Inicialmente el flujo f(e) = 0 para todo eje e en el grafo G
Inicializar Gr grafo residual según G y flujo s-t f = 0
Mientras haya un canino s-t en Gr
        Sea P el camino s-t en Gr de menor longitud posible
        Sea w el valor de flujo del cuello de botella del camino P

        Actualizar el flujo w en el camino P en G
        Actualizar Gr
```

```
Retornar f
```

De igual manera que utilizando Dinic, tendremos a los sumo caminos de longitud máxima O(V). Cada eje (hacia atrás o adelante) puede ser cuello de botella únicamente en un solo camino de una longitud (aunque podría volver a serlo nuevamente dos longitudes de camino después). Por lo tanto existirán como mucho O(VE) camino de aumento. Cada uno de ellos calculado en O(V+E). Por lo tanto llegando a una complejidad de $O(E^2V)$.

Si comparamos entre Edmonds-Karp y Dinic tenemos teóricamente una complejidad equivalente. Se podría considerar que el primero al evitar el proceso de construir los grafos de nivel tendrá en la práctica un mejor desempeño. Sin embargo esto no es así… dado que, el explicado hasta el momento, realmente no es el algoritmo completo de Dinic. Resta una parte que permite reducir la complejidad final del algoritmo. Pero antes de continuar, se debe tener en cuenta que hay 2 alternativas que resuelven el fragmento restante.

Según cuenta Dinitz[9], el conocido "algoritmo de Dinic" corresponde a una versión incorrecta de la propuesta original. Nótese la similaridad entre los apellidos "Dinitz" y "Dinic". En rigor corresponde a la misma persona: Yefim Dinitz científico informático de origen Sovietico y nacionalizado Israeli. Este autor cuenta que su propuesta se realizó en plena guerra fría y por lo tanto fue para el occidente parcialmente vedado su conocimiento. Quien intentó comprender basándose en información parcial el funcionamiento del mismo fue Shimon Even, investigador Israelí de ciencias de la computación. Utilizando una publicación reducida de sólo cuatro páginas y sin contacto con el autor interpretó su funcionamiento. Luego, en sus lecturas y su publicaciones - como el libro Graph Algorithms[10] - desde mediados de la década del 70 se encargó de difundirlo. La mayoría de la bibliografía[11] [12] (occidental) tomó la propuesta de Even. Ambas logran una complejidad de $O(V^2E)$. Dado que en muchas aplicaciones la cantidad de vértices son sensiblemente menores a los vértices, está cota resulta preferible a la otra.

[9] Dinitz' Algorithm: The Original Version and Even's Version, Dinitz, Y.,2006, En: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds) Theoretical Computer Science. Lecture Notes in Computer Science, vol 3895. Springer, Berlin, Heidelberg.
[10] Graph Algorithms,S. Even, 1979, Computer Science Press, Rockville, MD.
[11] Data structures and network algorithms, Robert Endre Tarjan, 1983. Society for Industrial and Applied Mathematics, USA.
[12] The design and analysis of algorithms, Dexter C. Kozen, 1992, Springer-Verlag, Berlin, Heidelberg.

Se conoce como **fase** a cada cálculo del grafo de nivel y sus posteriores procesos de aumentos de flujo. Hasta el momento dentro de cada fase se busca uno a uno los caminos de aumento en el grafo de nivel. Sin embargo, se puede en un único proceso encontrar todos ellos disminuyendo la complejidad final. La propuesta de Even utiliza una sola ejecución de DFS para lograrlo. Esta es la versión que analizaremos a continuación.

En una fase contamos inicialmente con el grafo de nivel. Partiendo desde la fuente, lo recorreremos mediante una variante de DFS. En este proceso se buscarán y ejecutarán caminos de aumento, se irán eliminando los ejes saturados y eliminando ejes y nodos que ya no tengan posibilidad de llegar al sumidero. Al final del proceso se habrán consumido todos los caminos de aumento de longitud igual al nivel del grafo. Todo este proceso podrá ser realizado en O(VE). Detallaremos el proceso con 4 subprocesos: inicialización, avance, retroceso y aumento.

La **inicialización** comienza ubicándose simplemente en el nodo fuente del grafo de nivel y estableciendo su predecesor como este mismo. Luego llama al proceso *avance*.

El **avance** busca un eje e=(u,v) que salga del eje u en el que se encuentra. Establece como actual v y como predecesor de v a u. Si este corresponde al nodo sumidero entonces encontró un camino de aumento y pasa el proceso de *aumento*. Alternativamente si no encuentra un eje hacia afuera entonces pasa al proceso *retroceso*.

El **aumento** reconstruye el camino desde el nodo al sumidero en base al registro de predecesores. Al hacerlo también se obtiene el cuello de botella y aumenta el flujo del grafo residual. Aquellos ejes del camino que son saturados se eliminan del grafo de nivel. Además se establece el nodo actual al inmediatamente anterior al eje más cercano a la fuente del camino que fue eliminado. Luego ejecuta el proceso *avance*.

El **retroceso** verifica que el actual es el nodo fuente. En ese caso termina el proceso. De lo contrario, elimina el nodo u en el que está ubicado (y todos sus ejes adyacentes) del grafo de nivel. Se posiciona en el predecesor de u y ejecuta el proceso *avance*.
En el siguiente pseudocódigo se muestra el proceso resumido:

Flujo máximo - Fase Dinic

```
Sea Gr el grafo residual.
Sea Lg el grafo de nivel correspondiente a Gr
// inicializacion
Sea P el Camino desde la fuente
Agregar la fuente a P.
Repetir
        //Avance
        Sea e=(P.ultimo,v) un eje que sale del último nodo del camino en Lg
        Si e no existe
                //Retroceso
                Sea r = P.ultimo
                Eliminar nodo r de Lg y sus ejes adyacentes.
                Eliminar nodo r de P
                Si r es la fuente
                        Retornar
        Sino
                Agregar el nodo v a P.
                Si P.ultimo es el sumidero
                        //Aumento
                        Sea w el valor de flujo de cuello de botella en P
                        Actualizar el flujo w en el camino P en G
                        Actualizar Gr
                        Quitar de Lg los ejes saturados en P
                        quitar de P los nodos hasta el anterior al último eje saturado
```

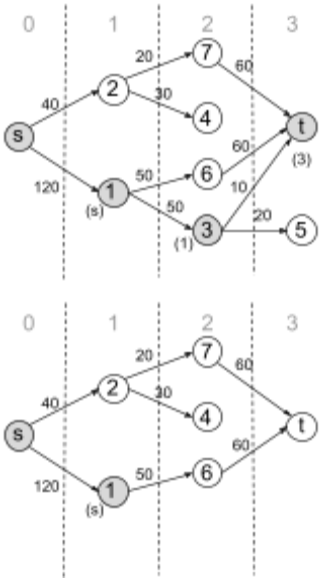
Analicemos la complejidad de este proceso. En primer lugar la inicialización se puede realizar en tiempo constante O(1). El retroceso elimina al menos un nodo y un eje del grafo de nivel cada vez que se ejecuta. Siendo entonces que contamos con |V| nodos y |E| ejes en el peor de los casos será un proceso global O(V+E) . El proceso de aumento se basa en procesos limitados por la cantidad de ejes en el camino. Cómo construimos caminos simples y la longitud máxima del camino es la cantidad de nodos menos 1, entonces tendremos que este proceso es O(V). Cada vez que se llama además satura al menos uno de estos. Por lo que en una fase podemos tener como mucho E aumentos. Globalmente este proceso es O(VE). Por último la cantidad de avances que podemos tener está limitada por la cantidad de ejes en el grafo de nivel. Por cada uno de ellos pasaremos una vez. En el avance llegaremos al sumidero o a un callejón sin salida y de allí aumentaremos el flujo o retrocedemos. En ambos casos quitaremos al menos un eje por el que no volveremos a pasar. Podemos ver este proceso como O(E) en forma global. Juntando todos las porciones

podemos ver que en forma conjunta la ejecución de una fase tiene una complejidad temporal de O(VE).

Actualizando el algoritmo de Dinic para la utilización de este proceso tendremos:

Flujo máximo - Dinic (versión Even)
Inicialmente el flujo f(e) = 0 para todo eje e en el grafo G Inicializar Gr grafo residual según G y flujo s-t f = 0 Obtener el grafo de nivel L_g en base a Gr Mientras sea accesible el sumidero desde la fuente en L_g Aumentar flujo en Gr mediante la fase con L_g Obtener el grafo de nivel L_g en base a Gr Retornar f

Dado que tendremos como mucho |V| fases, la complejidad total del algoritmo es $O(V^2E)$



Siguiendo el grafo de nivel de la imagen comenzaremos a buscar los caminos de aumento desde la fuente. Tenemos dos ejes. Tomamos el s→1 y realizamos un avance. Volvemos a repetir el avance primero al nodo 3 y luego llegamos al sumidero. Tenemos el camino de aumento s→1→3→t con cuello de botella 10. Realizamos el proceso de aumento. A consecuencia se satura el eje 3→t que se retira del grafo de nivel. Además el proceso continua desde el nodo 3. Se procede nuevamente a realizar un avance. Tenemos un eje saliente al nodo 5. Al llegar ahí realizamos un retroceso: quitamos el nodo 5 y sus adyacentes (3→5). Nos ubicamos en el nodo 3. Este tampoco tendrá ejes salientes así que también realizaremos el proceso de retroceso. En ese momento el grafo de nivel tendrá la estructura que se muestra en la segunda imagen. Se continuará con el proceso hasta eliminar finalmente la fuente y finalizar.

Existen gran cantidad de algoritmos adicionales que resuelven el problema de flujo máximo en tiempo polinomial. La siguiente tabla detalla algunos de ellos:

Algoritmo	Año	Complejidad

Karzanov[13]	1974	$O(V^3)$
Cherkasky[14]	1977	$O(V^2E^{1/2})$
Sleator y Tarjan[15]	1981	O(VE logV)
Goldberg y Tarjan[16]	1986	$O(VE \log(V^2/E))$
Orlin[17]	2013	O(VE)

Con la elección de alguno de ellos tendremos un algoritmo bueno (eficiente) para resolver nuestro problema.

4. Descomposición de flujo

A medida que calculamos el flujo máximo se van seleccionando caminos de aumento. Dependiendo del algoritmo seleccionado el criterio de elección es diferente. Diferentes algoritmos tienen diferente cota máxima de caminos de aumento que se espera utilizar para hallar el resultado final. Todos los algoritmos de aumento de flujo utilizan la red residual para poder encauzar flujo ante la elección previa de un camino no conveniente. Actualmente el algoritmo de Orlin en el 2013 es el que consigue un resultado de complejidad menor. La pregunta que cabe realizarse es si con el paso de los años se podrá encontrar uno mejor. Para eso desarrollaremos a continuación el **teorema de descomposición de flujo**.

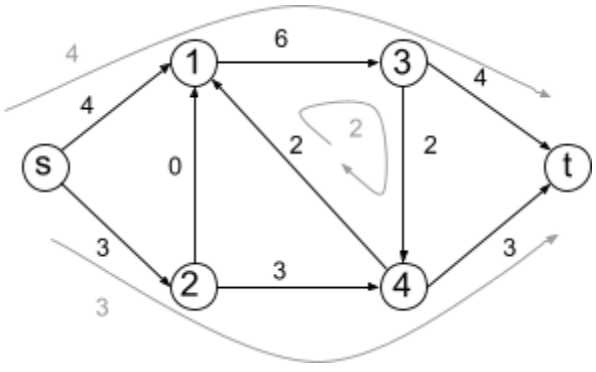
Dado un grafo G=(V,E) con una asignación de flujo f, podemos descomponer el flujo como un conjunto de caminos s-t y/o ciclos dentro del grafo. Más aún pueden existir como mucho |E| caminos de aumento y |E| ciclos con flujo diferente a cero. Cabe aclarar que

[13] Determining the maximal flow in a network by the method of preflows,Karzanov Alexander, 1974, Doklady Mathematics. 15. 434–437.
[14] Algorithm of construction of maximal flow in networks with complexity of $O(V^2 \sqrt{E})$ operations, R. V. Cherkasky, 1977, Mathematical Methods of Solution of Economical Problems, 7), pp. 112-125.
[15] "A data structure for dynamic trees.", Sleator, Daniel Dominic y Robert Endre Tarjan, 1981, Symposium on the Theory of Computing.
[16] A new approach to the maximum flow problem. A V Goldberg y R E Tarjan, 1986, En Proceedings of the eighteenth annual ACM symposium on Theory of computing (STOC '86). Association for Computing Machinery, New York, NY, USA, 136–146.
[17] Max flows in O(nm) time, or better, James B. Orlin, 2013, en Proceedings of the forty-fifth annual ACM symposium on Theory of Computing (STOC '13). Association for Computing Machinery, New York, NY, USA, 765–774. https://doi.org/10.1145/2488608.2488705

puede existir diferentes descomposiciones (algunas con más o menos caminos y/o ciclos) sin embargo la cota es aplicable para cada una de ellas.

Demostraremos el teorema mediante un procedimiento algorítmico. A partir del grafo G de la instancia con una asignación de flujo f podemos descomponer el flujo mediante un proceso iterativo. Se comienza hallando un eje e=(u,v) con flujo positivo. Posteriormente se busca un camino simple desde la fuente al nodo u. Este camino se completa buscando un camino desde el nodo v hasta el sumidero. Habremos encontrado un camino de aumento s-t P con un cuello de botella w. Descontamos de cada eje del camino un valor w de flujo. Sabemos que al menos uno quedará en cero. Deberemos repetir como mucho |E| veces dado que nos aseguramos que cada camino descuenta al menos un eje que queda sin flujo. Por lo tanto podemos tener como O(E) caminos de aumento. Cada uno de esos caminos si son simples como mucho pasarán por O(V) vértices. Por lo tanto en un proceso iterativo que busca un camino de aumento y luego aumenta el flujo en este tendrá como límite inferior una complejidad de O(VE). No se puede conseguir algo menor, al menos con este tipo de algoritmo.

Una cuestión adicional que no se debe despreciar es la posibilidad que existan ciclos de flujo si se permite en el algoritmo la construcción de caminos de aumento que puedan no ser simples. En ese caso en la búsqueda de la fuente o del sumidero se podrá regresar a algún nodo previamente visitado. Podemos extraer ese ciclo y buscar su propio cuello de botella para descontarlo del flujo total. En ese caso también al menos un eje queda sin flujo y también el ciclo puede tener como mucho una longitud O(V). Por lo que el proceso aún se mantiene O(VE)



El grafo G de la imagen tiene una asignación de flujo f que se detalla sobre cada eje del mismo. Si tomamos el eje 2→4 podemos realizar un camino de aumento de s-t como s→2→4→3 con un flujo de 3. Descontando ese flujo en G, podemos volver a buscar un camino de aumento que pase por 1→3. La misma puede ser s→1→3→t con un valor de flujo 4.

Quedará un ciclo 4→1→3→4 con una cantidad de flujo de 2. Luego de eso no quedan flujos disponibles sin camino en el grafo.

El siguiente pseudocódigo asume que no existen ciclos de flujo en la asignación de flujo f de un grafo G. Permite descomponer el flujo en la menor cantidad posible de caminos de aumento:

Descomposición de flujo

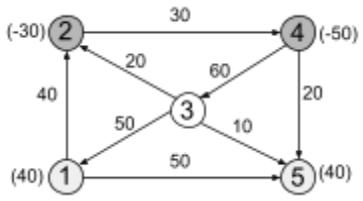
```
Sean Paths={} los caminos de aumento de flujo en G
Dado el Grafo G y una asignación de flujo f
Mientras existe un eje e=(u,v) con f(e)>0
        Sea e=(u,v) un eje con f(e)>0
        Buscar el camino P1 desde en G desde s hasta u pasando por ejes de flujo positivo.
        Buscar el camino P2 desde en G desde v hasta t pasando por ejes de flujo positivo.
        Construir camino de aumento P con P1,e y P2.
        Sea W el valor de flujo mínimo transportado por algún eje en P
        Descontar W en el flujo de cada eje de P.
        Agregar a Paths el camino P con flujo W.
Retornar Paths
```

5. Circulación de flujo con demandas y límites inferiores

En el problema del flujo máximo se plantean tres supuestos fuertes que condicionan la aplicabilidad del mismo. El primer supuesto fue establecer que solo existe un nodo que genera flujo al que llamamos fuente. En el mismo sentido, dispusimos un único nodo - el sumidero - que es el receptor de todo el caudal. El segundo supuesto corresponde a que lo que deseamos es obtener el mayor caudal de flujo posible. El tercer supuesto es que nos da lo mismo por cuales ejes circula el flujo y en qué cantidad. En este apartado modificaremos estos postulados. En primera instancia trabajaremos con los dos primeros y más adelante con una de las variantes del tercer postulado.

Llamamos **productores** "p" a aquellos nodos que tienen la capacidad de generar flujo. Cada uno de ellos tendrá una cantidad máxima que puede generar. De igual manera llamaremos **consumidores** "c" a aquellos nodos que tienen la capacidad de absorber flujo. Tendrán también una capacidad máxima para hacerlo. En tercer lugar existirán nodos que no producen ni consumen y son llamados nodos internos. Llamaremos **demanda** d_v al

valor numérico de unidades de flujo que produce o demanda el nodo v. Siendo para los productores un valor negativo y para los consumidores un valor positivo. Los nodos internos tendrán una demanda igual a cero. Cualquier nodo tiene la capacidad de conmutar flujo entre sus ejes de entrada y salida. Sin importar a qué tipo pertenece. Podrán arribar ejes a los nodos productores y salir de los ejes consumidores. Cada eje de la red tiene una capacidad de flujo que puede transportar.



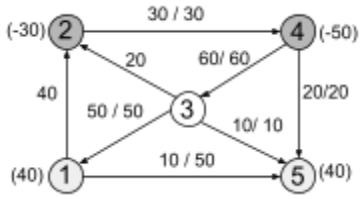
La red de flujo de la imagen cuenta con 5 nodos. Los nodos 1 y 5 son consumidores. Los nodos 2 y 4 son productores. El nodo 3 corresponde a uno interno. La demanda para cada nodo es d_1*=40,* d_2*=-30,* d_3*=0,* d_4*=-50,* d_5*=40.*

En este problema nos interesa saber - teniendo en cuenta la topología de la red - si todos los nodos consumidores pueden satisfacer su necesidad de flujo. A diferencia del problema de flujo máximo, este corresponde a un problema de decisión. Podemos responder negativamente de forma instantánea si la suma de las demandas de los nodos es diferente a cero. Corresponde al caso donde la capacidad de producción de la red es diferente a su demanda. Si ocurre lo contrario, aun así la respuesta podría ser negativa. Necesitamos un algoritmo que para cualquier instancia del problema nos otorgue la respuesta de forma óptima. A tales efectos realizaremos una definición que será clave en la resolución del problema.

Llamaremos **circulación** en el grafo G a la función que asigna un valor numérico real no negativo a cada eje de la red de flujo que satisface dos condiciones. En primer lugar, se mantiene la conocida **restricción de capacidad**: el flujo que pase por cada eje debe ser un valor entre cero y la capacidad del mismo. En segundo lugar modificaremos la condición de conservación de flujo, que ahora llamaremos **conservación de oferta/demanda**. Pediremos que en cada nodo la diferencia entre la suma de los flujos ingresantes por sus ejes de entrada y la suma de los flujos salientes por sus ejes de salida sean igual al valor de demanda del nodo. En forma resumida:

- Conservación de oferta y demanda: $\sum_{e=(u,v)} f(e) - \sum_{e=(v,u)} f(e) = d_v$ para todo nodo v
- Restricción de capacidad: $0 \leq f(e) \leq c_e$ para todo eje e=(u,v)

El problema de decisión se resume en establecer si existe o no una circulación para la instancia de la red de flujo. Más aún, en caso de existir deseamos poder expresarla para su análisis posterior.



Se muestra en la imagen una posible circulación de flujo que satisface la instancia del problema. Cada uno de los ejes tiene un flujo igual o menor a su capacidad. Cumplen con la restricción de capacidad. Los ejes productores 2 y 4 generan un flujo combinado de 80. Los ejes consumidores 1 y 5 absorben la misma cantidad. Podemos validar la conservación de oferta y demanda para cada uno de los nodos. El nodo 1 tiene un eje entrante 3→1 con flujo 50. Tiene los ejes salientes 1→2 y 1→5 con flujo 0 y 10 respectivamente. Además consume 40. Podemos expresar la ecuación de conservación como (50) - (0+10) = 40. Se observa que la igualdad se cumple. El nodo 2 tiene dos ejes entrantes con 0 de flujo. Por otro lado tiene un eje saliente con flujo 30. Siendo un nodo productor su d_2 es igual a -30. La ecuación de conservación se cumple 0 - 30 = -30- De igual manera podemos comprobar el resto de los nodos. Por lo tanto corresponde a una circulación con demanda y la instancia del problema se puede satisfacer.

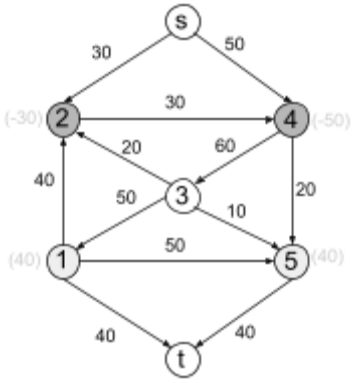
¿Cómo construimos un algoritmo que resuelva este problema? No requerimos hacerlo. Podemos utilizar el problema de flujo máximo para lograrlo. Solo requiere una transformación de la instancia del problema. La aplicación de un algoritmo para calcular el flujo máximo y una posterior interpretación de los resultados analizando el grafo residual resultante.

Para transformar un problema comenzamos con la adición de dos nodos: una fuente s* y un sumidero t* (los llamaremos siguiendo la nomenclatura tradicional “super fuente” y “super sumidero”). Estos nodos podrán generar y absorber una cantidad ilimitada de flujo. En segundo lugar crearemos un eje por cada nodo productor. Este eje partirá de la super fuente y arribará al productor. La capacidad del mismo será el valor opuesto de su demanda. En tercer lugar se creará un eje por cada nodo consumidor. Partirá desde el consumidor y culminará en el super sumidero. Su capacidad es el valor de demanda del nodo. Expresado en pseudocódigo:

Transformación de instancia de circulación con demanda a flujo máximo

Sea G=(V,E) red de flujo
Construir G'=(V',E') red de flujo con los nodos E y vértices V de G.
Agregar a V' un nodo s* 'super fuente'
Agregar a V' un nodo t* 'super sumidero'
Por cada nodo v en V'-{s,t}
    Si d_v<0
        Agregar eje e=(s,v) con capacidad -d_v
    Si d_v>0
        Agregar eje e=(v,t) con capacidad d_v
Retornar G'

El nuevo grafo tendrá 2 nodos nuevos y como mucho |V| ejes adicionales. El proceso de construcción del mismo - partiendo de una representación del grafo como una lista de adyacencias - es O(V+E).



Para la instancia del ejemplo se agrega los nodos super fuente y super sumidero. Los nodos 2 y 4 correspondientes a productores tendrán un eje ingresante desde la super fuente. Respectivamente con capacidad 30 y 50. Se puede observar que corresponde al opuesto de sus valores de demanda. Los nodos consumidores 1 y 5 tendrán ejes salientes que se conectan con el super sumidero. La capacidad de estos ejes es igual al valor de demanda de los nodos. Ambos 40. El nodo 3 correspondiente a un nodo interno no ve modificado sus ejes entrantes ni salientes.

Con la nueva red de flujo construida utilizaremos como una caja negra un algoritmo de resolución del problema de flujo máximo y obtendremos un grafo residual y un valor de flujo máximo. Si el flujo máximo es igual a la suma de las demandas de los consumidores podemos afirmar que existe una circulación con demanda. Además, la podemos obtener en base al grafo transformado. Simplemente quitando los ejes y los nodos agregados. Proceso que podemos hacer en tiempo polinomial O(V).

Transformación de solución de flujo máximo a circulación con demanda

Sea f asignación de flujo y G'=(V',E') red de flujo
Reemplazar G´por G (la red de flujo original)
Remover de f las asignaciones de flujo de los ejes que se conectan con s* o con t*

Retornar G

Veamos que efectivamente para cualquier instancia del problema esta metodología nos retorna una solución óptima. Queremos ver que existe una circulación factible si y sólo si en el grafo transformado existe un flujo máximo igual a la suma de las demandas de los consumidores.

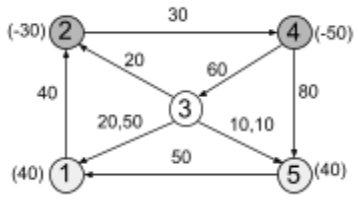
Llamaremos D a la suma de las demandas de los consumidores del grafo original G. Los únicos ejes que llegan al super sumidero s* en G' son los agregados en la transformación. Estos salen de los nodos consumidores. Su capacidad corresponde a la demanda del consumidor de donde salen. Por lo tanto la suma de todos los ejes que llegan al super sumidero t* corresponde al valor D. Realizamos un corte s-t en G' dejando todos los nodos, excepto el super sumidero, con la super fuente. Este corte tiene una capacidad D. Entonces, podemos afirmar que ningún flujo en G podrá ser mayor a D.

Supongamos que al aplicar el flujo máximo en G' conseguimos un flujo de D. Entonces en el corte s-t propuesto todos los ejes están saturados. Por la condición de conservación de flujo sabemos que en G' la suma del flujo entrante de cada nodo v debe ser igual al saliente. Matemáticamente $\sum_{e=(u,v)} f(e) = \sum_{e=(v,u)} f(e)$. Para los nodos consumidores uno de los ejes salientes es el agregado con destino a t* con capacidad d_v. Este eje está saturado por lo que su flujo es d_v. Podemos reexpresar la ecuación de conservación de flujo como $\sum_{e=(u,v)} f(e) = \sum_{e=(v,u)\,/\,u \neq t^*} f(e) + d_v$. Que podemos reescribir como $\sum_{e=(u,v)} f(e) - \sum_{e=(v,u)\,/\,u \neq t^*} f(e) = d_v$. Dado que en el grafo original el eje al supernodo no está presente. podemos ver que esa expresión cumple con la conservación de oferta y demanda en el grafo original $\sum_{e=(u,v)} f(e) - \sum_{e=(v,u)} f(e) = d_v$. También se cumple la restricción de capacidad en cada eje dado que la misma es válida para el flujo máximo y lo será también para la circulación.

Podemos hacer la misma evaluación para los ejes productores y ejes internos. Con eso podemos afirmar lo que queríamos probar: existe una circulación factible si y sólo si en el grafo transformado existe un flujo máximo igual a la suma de las demandas de los consumidores. Concluyendo que esta forma de resolver el problema es adecuada.

Es momento de trabajar sobre el tercer supuesto. Hay varias situaciones donde tenemos alguna condición adicional de flujo en los ejes. Tendremos en cuanta una en particular. Supondremos que algunos ejes pueden tener como requerimiento que circule por ellos un valor mínimo de flujo. Llamaremos **límite de capacidad inferior** (lower capacity bound) a la restricción que determina que para un determinado eje e del grafo el valor de flujo circulante sea mayor o igual a l_e. A su vez este valor debe ser menor o igual a la capacidad del eje. Nuestro nuevo problema incluirá los tres cambios propuestos al problema original. Lo llamaremos **circulación con demandas y límites**. Para reflejar la nueva situación modificaremos la restricción de capacidad.

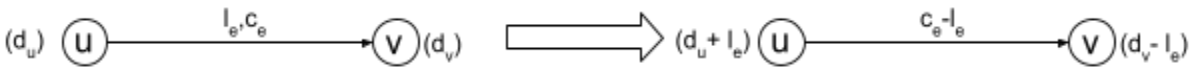
- Conservación de oferta y demanda: $\sum_{e=(u,v)} f(e) - \sum_{e=(v,u)} f(e) = d_v$ para todo nodo v
- Restricción de capacidad: $l_e \leq f(e) \leq c_e$ para todo eje e=(u,v)



En el ejemplo de la imagen se observa una instancia posible del problema de circulación con demandas y límites. Dos nodos productores contribuyen con valores de 30 y 50 de flujo. Dos nodos consumidores demandan 40 y 40 de flujo. Existen dos ejes con límite inferior. El eje 3→5 puede transportar 10 de flujo, pero a la vez requiere que el mínimo que pase sobre este sea esta misma cantidad. El eje 3→1 puede transportar 50 y requiere un mínimo de 20. El resto de los ejes no tiene un límite inferior. Que equivale a decir que el l_e de cada uno de ellos es igual a cero. La circulación a construir debe tener en cuenta la restricción de capacidad (con límites inferiores) y la conservación de oferta y demanda.

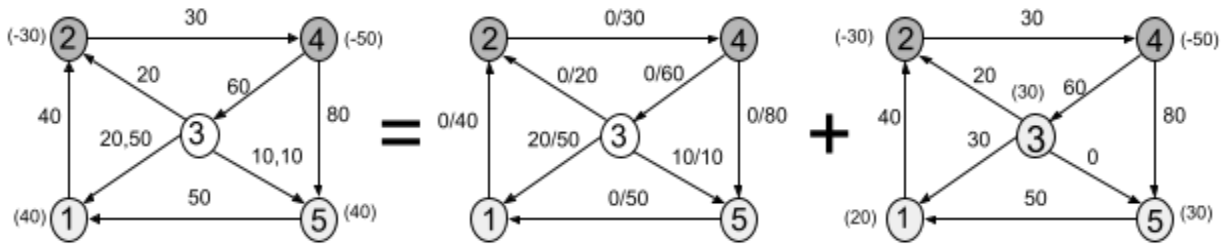
Para resolver este problema transformaremos primero el problema en uno de demanda (sin límites inferiores). Luego, utilizaremos una segunda transformación al problema de flujo máximo como vimos anteriormente. De esta forma resolveremos el problema para luego analizar los resultados.

En primer lugar, podemos intentar asegurar el flujo requerido por los ejes que tienen un límite de capacidad inferior. Para eso crearemos una “pseudo circulación” que cumpla con estos mínimos aunque no lo haga con el resto de las restricciones. Por cada eje con $l_e > 0$ supondremos un flujo de l_e que tendremos reservado. A esta pseudo circulación la llamaremos f*(e) y la sumaremos a una posterior que calcularemos con el grafo transformado. El nuevo grafo debe reflejar esa reserva. En primer lugar modificaremos la capacidad máxima de cada eje e=(u,v) con límite inferior: Al aseguramos que pasará este mínimo, solo nos queda como capacidad residual $c_e - l_e$ para cumplimentar la circulación. En segundo lugar debemos modificar los nodos u y v para que nos aseguren que el flujo pasará por el eje e. Esto lo podemos hacer forzando a que el nodo u “consuma” c_e y que el nodo v “produzca” c_e.



En la imagen se muestra la transformación de un eje y sus dos nodos adyacentes. Estamos obligando a que llegue a “u” la cantidad requerida por el eje l_e y como “u” la consume nos aseguramos que no se redirija a ningún otro eje. Además estamos obligando de forma ficticia al nodo “v” a producir l_e que no vendrá de otro eje. La transferencia de u a v se reservará en la pseudo circulación. El mismo proceso se debe realizar con todos los ejes.

Continuando con el ejemplo anterior se presenta la transformación de la red de flujo con demandas y límite inferior en una pseudo circulación más una nueva red de flujo con demandas pero sin límites inferiores en sus ejes. Existen en este caso dos ejes con límite inferior 3→1 y 3→5. Para la transformación primero crearemos una pseudo circulación f(e)=c_e para cada eje e. En nuestro caso f*(3→1)=20 y f*(3→5)=10. El resto de los ejes tendrán valor cero.*



A continuación transformaremos la red de flujo. Restamos de la capacidad de los ejes su límite inferior. Para el eje 3→1 su nueva capacidad será $c'_{3\to1} = c_{3\to1} - l_{3\to1}$ *.= 50 - 20 = 30. Lo mismo realizamos para el eje 3→5. En este caso quedará con capacidad cero. El resto de los ejes no se modifica. Por último se tienen que adecuar las demandas de los nodos en los extremos de los ejes modificados. El nodo 1 al ser el destino del eje 3→1 debe producir* $l_{3\to1}=20$*. Como antes consumía 40, pasará ahora a demandar* $d'_{3\to1} = d_{3\to1} - l_{3\to1} = 40 - 20 = 20$*. El nodo 5 al ser destino del eje 3→5 debe producir* $l_{3\to5}=10$*. Antes consumía 40 y pasará a consumir* $d'_{3\to5} = 40 - 10 = 30$*. El último nodo que se verá afectado es el 3. Este es el origen de dos nodos con límite inferior 3→1 y 3→5. Era un nodo interno por lo que su demanda era cero. Pero ahora debe consumir 20+10=30. De esa forma pasa ahora a ser un nodo consumidor.*

Podemos expresar la transformación con el siguiente pseudocódigo:

Transformación de circulación con demanda y límites a circulación con demanda

Sea G=(V,E) red de flujo
Construir G'=(V',E') red de flujo con los nodos E, vértices V de G, con sus demandas y capacidades (sin límites inferiores).
Sea f* una pseudo circulación

Por cada eje e=(u,v) de E
    $f^*(e) = l_e$

    $c'_e = c_e - l_e$

    $d'_u = d'_u + l_e$
    $d'_v = d'_v - l_e$

Retornar G' y f*

La transformación en su conjunto tiene una complejidad de O(V+E).

Una vez que realizamos la transformación reservamos la pseudo circulación y con el grafo G' modificado resultante - que corresponde a una red de flujo con demandas - calculamos la circulación con demandas. Lo resolvemos con el método presentado anteriormente. Con el resultado del flujo máximo podemos realizar el mismo análisis de optimalidad. Si el flujo máximo tiene como valor la suma de las demandas de los nodos consumidores, entonces

la circulación con demandas y límites inferiores es posible. El resultado lo obtenemos sumando la pseudo circulación con la asignación de flujo obtenido de G'.

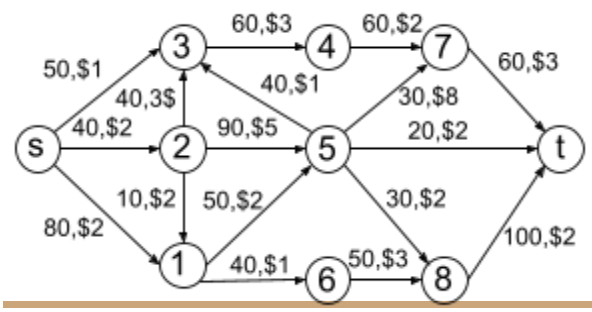
La complejidad total de solucionar este problema corresponde a la suma de las complejidades de cada una de las transformaciones y de resolver el problema de flujo máximo.

6. Flujo máximo y costo mínimo.

En el problema de la maximización de flujo al momento de buscar la solución óptima no nos interesaba cuál de los ejes utilizaba la solución final. Simplemente llegar al mayor valor de flujo posible para cada instancia posible del problema. Sin embargo en situaciones específicas el uso de los ejes para el transporte de flujo tiene un costo asociado. El uso de caminos de aumento diferentes puede hacer que soluciones que transportan la misma cantidad de flujo tengan costos diferentes. En muchas ocasiones nos interesará conseguir el menor costo transportando el máximo de flujo posible. En este apartado trabajaremos en esa hipótesis. Definiremos el problema como:

Problema del flujo máximo y costo mínimo (Max Flow - Min Cost)
Dada una red de flujo G=(V,E) con capacidades y costo por unidad de transporte en sus ejes, deseamos obtener una asignación de flujo S-T que maximice el transporte de flujo entre la fuente y el sumidero a la vez que minimice el costo total incurrido.

Llamaremos u_e al costo por unidad de flujo transportada en el eje e=(u,v). Este parámetro caracteriza a cada eje con un valor entero mayor o igual a cero. Podemos calcular el costo de un determinado flujo s-t como la sumatoria de los costos unitarios por el volumen de flujo de cada uno de los ejes de la red. Matemáticamente $p(f) = \sum_{e \in E} f_e * u_e$.



Se presenta una red de flujo con 10 nodos, entre ellos la fuente y el sumidero. Cada eje presenta una capacidad y un valor unitario. Por ejemplo el eje s→1 tiene una capacidad máxima de 50 y por cada unidad de flujo que

*transporta tiene un costo de $1. De forma similar el eje 2→5 puede transportar como límite 90 y por cada unidad tendrá un costo de $5. Podemos construir un flujo s-t de 40 utilizando el camino de aumento s→3→4→7→t. El costo de este transporte corresponde a la suma de los costos por unidad por el flujo de cada uno de los ejes de este camino: 40*1$+40*3$+40*2+40*$3 = $360. Otro flujo s-t con el mismo caudal se pueda armar utilizando el camino de aumento s→1→6→8→t. En este caso el costo total del transporte corresponde a 40*$2+40*$1+40*$3+40*$2 = $320. Esta segunda opción presenta menor costo y por lo tanto es preferida. El objetivo final es encontrar entre todas las opciones de flujo máximo aquella que otorgue el menor costo posible.*

Para la resolución de este problema existen varias propuestas. Estudiaremos una por su simpleza y su similitud a los algoritmos previamente analizados. Utilizaremos como base el algoritmo de Ford-Fulkerson. El órden en el que se seleccionan los caminos de aumento es determinante en la resolución. De forma similar a Edmonds-Karp donde buscábamos el camino de menor longitud, en esta ocasión buscaremos el camino de menor costo. Se conoce a este algoritmo como **successive shortest path**. Este algoritmo fue descubierto y demostrado por diversos autores. Entre William Jewell[18], Masao Iri[19] y Robert Busacker y Paul Gowen[20] durante finales de la década del 50 e inicios del 60.

Utilizaremos el grafo residual propuesto por Ford Fulkerson. Los ejes e'=(u,v) hacia adelante - propios de la instancia del grafo - mantendrán su costo u_e. Adicionalmente se asociará a cada eje hacia atrás e"=(v,u) el costo negado del eje $u_{e''}=-u_e$. El costo de cada camino de aumento P estará dado por la suma de los costos positivos de los ejes hacia adelante y los costos negativos de los ejes hacia atrás que atraviesa $p(P) = \sum_{e \in P} f_e * u_e = \sum_{e' \in P} f_{e'} * u_e - \sum_{e'' \in P} f_{e''} * u_e$. Se estableció al definir Ford-Fulkerson, que al utilizar un eje e"=(v,u) hacia atrás en un camino de aumento P con cuello de botella "w" se indica la reducción "w" de flujo del eje e=(u,v) del grafo G. Esta reducción corresponde a un reencauzamiento de flujo y este reencauzamiento lleva a una reducción del costo asociado al uso del eje.

[18] Optimal flow through networks, William S. Jewell. 1962, Operations Research, 10(4):476–499.
[19] A new method for solving transportation-network problems, Masao Iri, 1960, Journal of the Operations Research Society of Japan, 3(1,2):27–87.
[20] A procedure for determining a family of miminum-cost network flow patterns, Robert G. Busacker y Paul J. Gowen, 1960, Technical Paper 15, Operations Research Office, Johns Hopkins University.

En cada iteración buscaremos un camino de aumento con el menor costo posible utilizando el grafo residual. Buscaremos el cuello de botella y aumentaremos ese valor el flujo s-t. Terminaremos cuando no exista un camino de aumento disponible. En pseudocódigo:

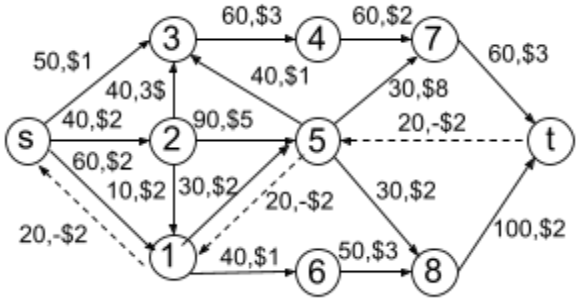
Max Flow - Min cost: successive shortest path

```
Inicialmente el flujo f(e) = 0 para todo eje e en el grafo G
Inicialmente el costo total de transporte T=0
Inicializar Gr grafo residual según G y flujo s-t f = 0
Mientras haya un canino s-t en Gr
        Sea P el camino s-t en Gr de menor costo disponible
        Sea w el valor de flujo del cuello de botella del camino P
        Sea v el valor de transportar el flujo w por P

        Actualizar el flujo w en el camino P en G
        Actualizar el costo total T sumando v.
        Actualizar Gr

Retornar f,T
```

La búsqueda del camino más corto en un grafo la podemos hacer con Dijsktra si no existen ejes con peso negativo o Bellman Ford si no existen ciclos negativos.

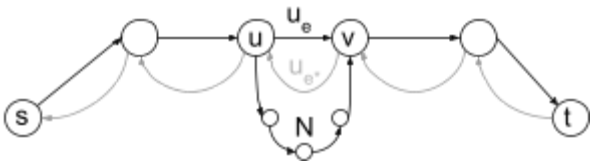


Utilizaremos el grafo presentado anteriormente para ejemplificar el funcionamiento del algoritmo. Comenzamos buscando en el grafo residual el camino de menor costo entre s y t. Aplicando Dijsktra o Bellman Ford hallaremos a P como s→1→5→t con costo total $5 y cuello de botella 20. Realizando un aumento de flujo tendremos un flujo de 20 con un costo de 120 (20 de flujo transportado con un costo unitario de 6). La actualización del grafo residual nos muestra los primeros ejes con costo negativo. Debemos volver a iterar dado que aún quedan camino de aumento. Aplicamos Bellman Ford y buscamos el camino mínimo también utilizando los ejes hacia atrás. Repetiremos hasta que no exista un camino de aumento posible.

Debemos analizar la optimalidad de este algoritmo. En primer lugar el algoritmo itera buscando un camino de aumento en el grafo residual y finaliza cuando no existe. Por lo analizado en Ford Fulkerson sabemos que cuando el sumidero se encuentra desconectado de la fuente el flujo s-t obtenido es máximo.

En segundo lugar queremos ver si en algún momento puede existir un ciclo negativo. Sabemos que a medida que se van agregando ejes hacia atrás con una capacidad diferente a cero se posibilita el recorrido por ejes de costo negativo. Por lo tanto es válido preguntarse si un ciclo negativo no puede surgir. Al crearse el grafo residual no hay ejes hacia atrás, por lo tanto no hay ejes negativos. al buscar el primer camino de aumento s-t no hay posibilidad de la existencia de ciclos negativos.

Supongamos que luego de un camino de aumento P aparece en nuestro grafo residual G_r el primer ciclo negativo. Como antes del aumento no había ciclos negativos, significa que al menos un eje e=(u,v) de P generó en el grafo un eje e*=(v,u) que cerró el ciclo negativo junto a un segmento N en G_r luego del mismo. Sabemos que el costo del ciclo lo podemos calcular como U_N la suma de los ejes de N y u_{e*}. Como es un ciclo negativo: $U_N + u_{e*} < 0$ y podemos reemplazar u_{e*} por $-u_e$. Por lo tanto $U_N < u_e$. Nótese que más de un eje de P podría generar el ciclo negativo, pero para el caso la demostración es la misma. Vemos que el segmento "N" existía previo a encontrar el camino de aumento P de costo mínimo. Además vemos que los extremos que U_e y N son los mismos nodos u y v. Por lo tanto, nada impide reemplazar en P el eje "e" por el segmento N. Lo que nos asegura disminuir el costo total del camino P. Lo que resulta absurdo dado que afirmamos que P era el camino de menor costo entre s y t en el grafo residual. Por lo tanto es imposible que al seleccionar el camino de menor aumento se produzca un ciclo negativo. Y esto no ocurrirá en ningún momento del proceso.



Ante la ausencia de ciclos negativos podemos asegurar el uso de Bellman-Ford para el camino menos costoso. Pero la importancia de la demostración va más allá de eso.

Antes de continuar, necesitamos un andamiaje extra en el proceso demostrativo. Buscaremos una forma de comparar dos soluciones factibles de una determinada red de flujo (entendidas estas como de máximo flujo aunque posiblemente diferente costo total). Llamaremos a esta como **teorema de ciclos de aumento**:

Sean dos soluciones factibles cualesquiera X e Y de una determinada red de flujo. Entonces X es igual a Y más el flujo de como mucho |E| ciclos de aumentos dirigidos en el grafo residual de X. Más aún, el costo de X es igual al costo de Y más el costo de los flujos en esos ciclos de aumento.

Llamamos ciclo de aumento dirigidos con respecto a un flujo X a el proceso de aumento un valor positivo w de flujo f por un ciclo C en el grafo residual y que el resultado siga siendo una solución factible. El ciclo contendrá ejes hacia adelante donde aumentará el flujo y ejes hacia atrás donde el mismo incrementará. Podemos expresar la variación del costo como

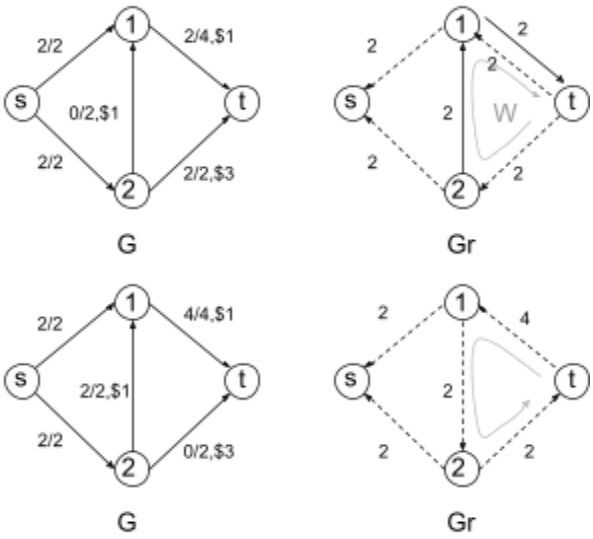
$$\sum_{e \text{ ejes hacia adelante en } C} u_e * w \; - \sum_{e \text{ ejes hacia atras en } C} u_e * w$$

Se puede ver este proceso como un reencauzamiento de flujo por otros caminos. En el caso extremo el ciclo puede incluir la fuente y el sumidero. Allí lo que se estará realizando es cancelar un P camino de aumento s-t para dejar lugar a otro P' con el mismo caudal de flujo. También es posible reencazar flujo de solo un segmento de un camino de aumento P. En cualquiera de esas situaciones, sabemos que la nueva solución tiene el mismo caudal de flujo. Sin embargo pueden variar en su costo total puesto que los nuevos ejes transitados pueden tener otros valores por unidad.

Por el teorema de descomposición de flujo sabemos que podemos descomponer cada uno de los flujos de X e Y en como mucho |E| caminos de aumento y ciclos. Por lo que podemos utilizar el mismo orden de ciclos de aumento para transformar una solución factible en otra. En cada ciclo llevaremos a al menos un eje de X al caudal de flujo indicado en Y.

En el ejemplo vemos un grafo con su correspondiente asignación de flujo acompañado de su grafo residual. Corresponde al flujo máximo al estar saturadas todos los ejes que salen de la fuente. Podemos descomponer el flujo en dos caminos: s→1→t y s→2→t. Ambos transportan 2

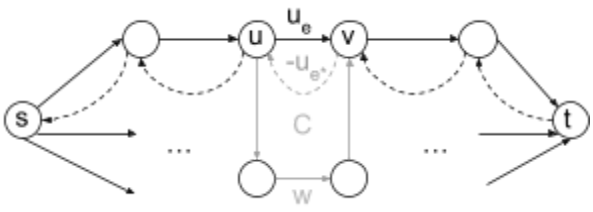
unidades. Analizando el grafo residual se puede encontrar el ciclo W como 2→1→t→2. Por ese ciclo se puede aumentar hasta 2 unidades de flujo que modificará la solución por otra factible.



El proceso de aumento del ciclo lo que realiza es reencauzar el mismo manteniendo. El cuello de botella en el ciclo W es 2 por lo que aumenta a cada eje en ese ciclo lo que hace es reducir en 2 los ejes por los que pasa, y aumentar en la misma cantidad a sus ejes inversos. En la segunda fila de la imagen se muestra el resultado del grafo luego del aumento de flujo en W. Ahora el grafo se puede descomponer en los siguientes 2 caminos de aumento con flujo de 2 unidades cada uno: s→1→t y s→2→1→t. El aumento en el ciclo trae como consecuencia el cambio del costo total del flujo. Los ejes fuera del ciclo no aportan cambios. Sin embargo, todos los ejes hacia atrás dentro de W reducirán el costo total del transporte en dos veces el costo por unidad y los ejes hacia adelante aumentarán también en dos veces su costo por unidad. Concretamente en W se producirá un cambio total en el costo de flujo de $-u_{t\to 2} * 2 + u_{2\to 1} * 2 + u_{1\to t} * 2 = (-u_{t\to 2} + u_{2\to 1} + u_{1\to t}) * 2 = 2(-3+1+1) = -2$.

Que corresponde a la suma de los costos del ciclo por el volumen transportado. Observar que el ciclo W también se puede pensar como una devolución de flujo de "t" a "s" t→2→s y un posterior camino de de aumento de "s" a "t": s→2→1→t.

Ahora si, continuamos: Afirmamos (y demostraremos) que una solución factible (con flujo máximo) S* es una solución óptima si y sólo si su grafo residual no contiene ciclos negativos.



Supongamos que obtengo una solución S que tiene un ciclo negativo C en el grafo residual. Sabemos que inicialmente no teníamos ciclos negativos, por lo tanto el mismo aparece luego de un camino de aumento. Al menos

uno de esos ejes traslada flujo de la fuente al sumidero y es un eje hacia atrás. Podemos tomar como cuello de botella de C el valor de capacidad más pequeño w entre los ejes que lo conforman. Realizando un ciclo de aumento dentro del ciclo de w podemos mantener el flujo máximo y disminuir el costo total. Es decir que quitaremos w de flujo que pasa por e y lo derivaremos por el camino alternativo que ofrece el ciclo. Sabemos que al ser un ciclo negativo $U_{C-\{e^*\}} < u_e$. Actualmente el costo total del transporte lo podemos dividir $p(f) = \sum_{e \in E-C} f_e * u_e + \sum_{e \in C} f_e * u_e$. La primera sumatoria no se modifica luego de aumentar el flujo en el ciclo. La segunda pasa ser ser $f_e * u_e + \sum_{e \in C-\{e\}} f_e * u_e$ a $u_e * (f_e - w) + \sum_{e \in C-\{e\}} u_e * (f_e + w) = \left[f_e * u_e + \sum_{e \in C-\{e\}} f_e * u_e \right] - \left[w \left(u_e - \sum_{e \in C-\{e\}} u_e \right) \right]$ y como $u_e - \sum_{e \in C-\{e\}} u_e$ es un valor positivo vemos que el costo total disminuye. Por lo tanto si S* es un flujo óptimo entonces su grafo residual Gr no puede contener un ciclo negativo.

Comprobemos ahora la afirmación complementaria: Supongamos S* es un flujo factible y que su grafo residual Gr no tiene ciclos negativos. Queremos ver que corresponde al flujo factible de menor costo posible. Partimos de S un flujo óptimo teórico diferente a S*. Por el teorema de ciclos de aumento, podemos representar la diferencia de flujos en S y S* como mucho por m ciclos de aumento con respecto al ciclo S*. Además la suma de los costos de los flujos en estos ciclos son iguales a p(S)-p(S*). Como por hipótesis en Gr no tiene ciclos negativos entonces su aumento no puede disminuir el costo total. Por lo tanto p(S)-p(S*)≥0 o p(S)≥p(S*). Pero como S es un flujo óptimo, no puede existir otra asignación de menor costo. Entonces p(S)≤p(S*). Unificando ambas expresiones vemos que p(S)≤p(S*) y S* también corresponde a un flujo óptimo.

En conclusión, podemos afirmar que si el grafo residual no contiene ciclos negativos S* debe ser óptimo. Por lo cual el algoritmo finaliza con un flujo máximo y de menor costo posible como queríamos demostrar.

Resta determinar la complejidad del algoritmo. En primer lugar podemos fácilmente ver que en cada iteración el algoritmo busca el camino mínimo y realiza un camino de aumento en O(V). Si usamos Bellman Ford para buscar el camino mínimo en el grafo

residual tendremos una complejidad de O(VE). La cantidad de caminos de aumento queda determinado por esta elección de camino. Nada impide que el camino de menor costo tenga un cuello unitario. En ese caso, si llamamos $C = \sum_{e \in S} c_e$, donde S es el subconjunto de ejes que sale de la fuente, la complejidad final será O(CVE). Corresponde a un algoritmo pseudo polinomial.

Cabe aclarar que es posible mediante una transformación de la red de flujo evitar la aparición de ejes negativos. De esa forma se puede utilizar siempre Dijkstra. No nos vamos a explayar en el mismo, pero un lector interesado lo puede encontrar por ejemplo en el libro "Network Flows: Theory, Algorithms, and Applications"[21].

Además de este algoritmo, existen otros algoritmos que resuelven el problema planteado de forma óptima. Muchos de estos se los puede encontrar también en el libro recién nombrado. Entre ellos "Out of Kilter"[22], "Cycle Cancelling algorithm"[23], "Capacity Scalling Algorithm", "Cost Scalling Algorithm" y "Minimum Mean Cycle-Cancelling Algorithm"[24] entre otros. Siendo el último nombrado totalmente polinomial.

[21] Network flows: theory, algorithms, and applications, Ravindra K. Ahuja, Thomas L. Magnanti, y James B. Orlin. 1993. Prentice-Hall, Inc., USA.
[22] Monotone networks. George J. Minty. 1960. In Proceedings of the Royal Society of London A, pages 194–212
[23] A primal method for minimal cost flows with applications to the assignment and transportation problems. Morton Klein. 1967, Management Science, 14(3):205–220.
[24] Finding minimum-cost circulations by canceling negative cycles. Andrew V. Goldberg y Robert E. Tarjan. Journal of the ACM, 36(4):873–886, 1989.