TEORÍA DE ALGORITMOS 1

Reducciones polinomiales (con aplicación a redes de Flujo)

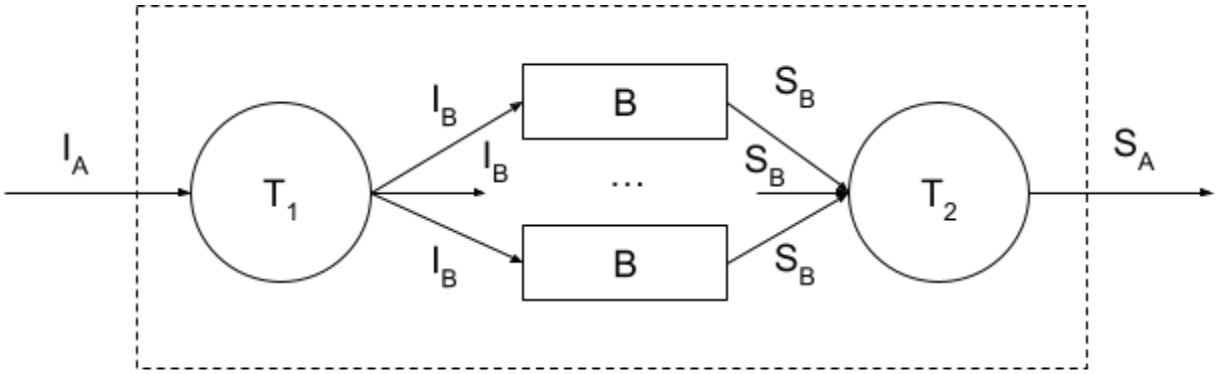
Por: ING. VÍCTOR DANIEL PODBEREZSKI
vpodberezski@fi.uba.ar

1. Reducciones

Al trabajar con diferentes problemas de redes de flujo pudimos observar que algunos de ellos se podían resolver transformando la instancia del mismo en la de otro problema. Posteriormente hacíamos uso de algún algoritmo que conocíamos para resolver este y finalmente convertimos la solución obtenida en la del problema original. Este procedimiento es ampliamente utilizado en diversas situaciones en la resolución de problemas de la vida cotidiana. Es tan habitual, incluso, el uso de este tipo de procedimientos que muchas veces se realiza de forma automática y sin conciencia de la misma.

Llamaremos **reducción** al procedimiento de utilizar un problema en el proceso de resolver otro. Partimos de un problema A que se quiere resolver y otro problema B que utilizaremos como parte del proceso de resolución. No nos importará como se resuelve B. Lo veremos como una caja negra que recibe una instancia y retorna su solución. El proceso de reducción toma cualquier instancia de A y la transforma en una o más instancias del problema B. Estas nuevas instancias de B - que pueden crearse todas juntas o a medida que se van requiriendo por el procesamiento - se resuelven mediante la caja negra que nos brinda la solución. Una vez obtenidas todas las soluciones se unifican en un nuevo proceso de transformación que las toma y combina en la solución esperada de A. Esperamos que para toda instancia del problema A el proceso completo nos retorne la solución correcta.

Esquemáticamente podemos ver la reducción como:

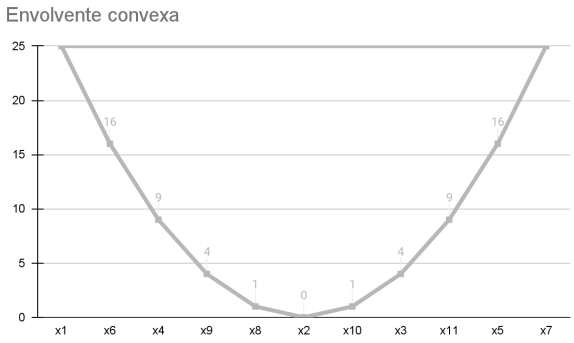


Reducción de A a B

Hablamos de **reducción polinomial** cuando el tiempo de procesamiento de las transformaciones y la cantidad de instancias de B generadas a partir de la transformación es polinomial en función del tamaño de la instancia del problema A. El tiempo de procesamiento de la caja negra no determina si la reducción es polinomial. Expresaremos una reducción polinomial entre un problema A a un problema B como $A \leq_p B$ y leeremos cómo "reducir polinomialmente el problema A al problema B".

Presentaremos dos ejemplos de reducciones polinomiales.

Contamos con un conjunto de "n" números enteros $\{x_1,x_2,..x_n\}$ y queremos ordenarlos de menor a mayor. Supongamos que no contamos con herramientas ni conocimientos para resolver este problema. Sin embargo sabemos resolver el problema envolvente convexa que tomaremos como nuestra caja negra. Podemos transformar cada uno de los x números y transformarlo en un elemento del plano $(x,y)=(x,x^2)$ utilizaremos esta instancia de n puntos como entrada del problema de envolvente y obtendremos como solución una secuencia de puntos que representa la componente convexa. Buscando al elemento menor y siguiendo a contra reloj la solución de la envolvente convexa tendremos todos los números ordenados. En la imagen se muestra la instancia de 11 números {-5,0,2,-3,4,-4,5,-1,-2,1,3} que se transforman en el conjunto de puntos {(-5,25),(0,0),(2,4),(-3,9),(4,16),(-4,16),(5,25),(-1,1),(-2,4),(1,1),(3,9)} cuya envolvente se muestra. Recorriendo los elementos desde el que tiene menor coordenada x podemos mostrar



los números ordenados. La complejidad de la primera transformación corresponde a convertir cada número en un punto en el plano. Lo podemos realizar en O(n). Luego se genera una única instancia de componente convexa por lo que se ejecutarán O(1) llamadas a la caja negra. Por último transformar la solución obtenida de la caja negra a los números ordenados corresponde a O(n). Se puede calcular la complejidad total del proceso conociendo la complejidad de la caja negra y unificando todos las complejidades de los procesos realizados.

Supongamos que tenemos un grafo G=(V,E) dirigido y ponderado sin ciclos negativos y queremos conocer cuál es el menor costo de llegar de un nodo a otro. Además queremos saber cuales son esos par de nodos. Llamaremos a este problema "Distancia mínima global de un grafo" (DMG). Supongamos que no conocemos el algoritmo para resolver este problema, pero tenemos un algoritmo para resolver el problema de la distancia mínima partiendo de un nodo del grafo (DM). Podría ser Bellman-Ford. Podemos reducir el problema de DMG a Dm para resolverlo. La primera transformación corresponderá a transformar la instancia del problema de DMG. Como el problema de DM requiere el grafo y un nodo de inicio, no tenemos que modificar el grafo. Lo que si realizaremos es construir O(V) instancias con ese grafo y uno de los nodos como inicio. Ejecutaremos |V| veces la caja negra y tendremos como resultado de cada uno la distancia mínima al eje de destino. La segunda transformación debe recorrer cada uno de estos resultados y analizar cual de todos las soluciones de DM tiene la distancia mínima global y entre cuales nodos se encuentra. Esto se puede realizar en $O(V^2)$. Podemos calcular la complejidad total de este proceso como generar |V| instancias de DM. ejecutar O(V) veces Bellman Ford O(VE) y luego analizar los resultados en $O(V^2)$. De forma resumida la complejidad total es $O(EV^2)$.

Es posible encadenar reducciones polinomiales entre varios problemas. El costo total de resolver será la suma del costo de cada uno de los procesos involucrados en las reducciones. Un ejemplo que ya analizamos en este sentido fue el problema de circulación con demandas y límites inferiores. Primero lo redujimos al problema de circulación con demandas y luego este al problema de flujo máximo.

Las reducciones polinomiales son poderosas herramientas para permitir resolver una gran cantidad de problemas. Sin embargo, en ocasiones si se usan sin los cuidados adecuados puede traer resultados inesperados. Algunos de los errores habituales corresponden a no cuidar que los procesos sean realmente polinomiales para toda instancia posible del

problema original. Por otro lado muchas veces analizan transformaciones que solo funcionan con un subconjunto de instancia del problema y fallan con otros. Otra cuestión habitual es realizar reducciones de forma implícita.

Una reducción es un proceso tan naturalizado que en ocasiones al conceptualizar un problema mentalmente se la utiliza sin darse cuenta. Para ejemplificar podríamos tener un problema donde tenemos estaciones de trenes que unen pueblos con un costo de viaje entre cada uno de ellos y queremos viajar con el menor costo posible. En ese caso es habitual ver el problema como el problema del camino mínimo de un grafo. Eso no es más que una reducción de un problema real a uno más abstracto.

A continuación se presentarán un conjunto de reducciones polinomiales donde reduciremos diferentes problemas al problema de flujo máximo.

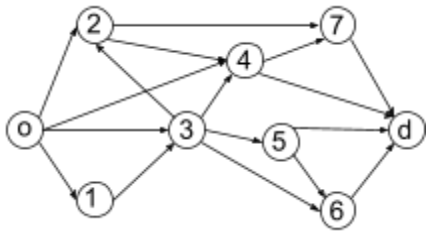
2. Caminos arco-disjuntos entre dos nodos en el grafo.

El siguiente problema que analizaremos concierne a uno de relativa importancia dentro de las redes de comunicación y en teoría de grafos en general. Corresponde a un problema de conectividad del grafo medido entre dos nodos del mismo:

Problema de caminos arco-disjuntos
Dado grafo G=(V,E) dirigido, un nodo o $\in$ V origen y un nodo d $\in$ V destino. Deseamos conocer la cantidad de caminos arco-disjuntos que se pueden trazar entre ellos.

Dos caminos son arco-disjuntos (o ejes-disjuntos) entre sí, si no comparten ejes entre ellos. Es decir que queremos conocer la máxima cantidad de caminos independientes entre sí que se pueden trazar desde un nodo origen a uno destino.

Con el objetivo de resolver este problema es útil conocer el **teorema de Menger**: El máximo número de caminos arco-disjuntos desde el nodo o al d en un grafo es igual al mínimo número de ejes que pueden ser eliminados para que no hayan tales caminos. Este enunciado - que luego demostraremos - es fácilmente vinculable al corte mínimo de la red residual. Por lo que el camino para buscar solución a este problema nos llevará a reducirlo al problema de flujo máximo.



En el grafo de la imagen queremos encontrar caminos arco-disjuntos que unen el nodo o con el nodo d. El camino c_1: o→2→7→d y c_2: 0→3→5→d son arco disjuntos entre sí. El camino c_3: 0→1→3→4→7→d es arco-disjunto con c_2, pero no lo es con c_1.

En la reducción, la primera transformación consistirá en la generación de una red de flujo utilizando como fuente y sumidero el nodo de origen y destino respectivamente. Mantendremos todos los ejes y nodos de la red original. Definiremos para cada eje una capacidad de valor unitario.

El pseudocódigo de la transformación correspondiente:

Transformación a instancia de problema de flujo máximo (T1)
Sea G=(V,E) el grafo dirigido Sea o,d ∈ V los nodos de origen y destino respectivamente. Construir G'=(V,E) red de flujo con nodo o como fuente y nodo d sumidero Para cada eje e=(u, v) ∈ E Establecer C_e = 1 Retornar G'

La complejidad corresponde a O(V+E) por crear la red residual con la capacidades unitarias en los ejes.

Una vez obtenida la red de flujo utilizaremos como caja negra un algoritmo que resuelve el problema del flujo máximo y obtendremos como resultado la asignación de flujo f y el grafo residual. La complejidad de este punto depende del algoritmo utilizado en la caja negra. Si utilizamos, por ejemplo, Dinic tendremos un $O(V^2E)$.

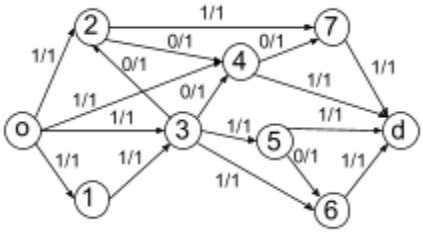
Llegados a este punto debemos transformar la solución del problema de flujo máximo en la de nuestro problema: poder determinar la cantidad de caminos disjuntos

Transformación de la solución del problema de flujo máximo a solución (T2)

Sea f la asignación de flujo
Sea G'=(V,E) la red de flujo y Gr el grafo residual resultante.

Establecer |P| como cantidad de caminos la suma de los flujos que salen de la fuente.
Reconstruir los caminos disjuntos P mediante el algoritmo de descomposición de flujos
Retornar P y |P|

Obtener |P| lo realizaremos observando los ejes que salen de la fuente, lo que corresponde a un proceso O(E) y la descomposición del flujo se puede hacer en O(VE). Por lo tanto la complejidad temporal de la segunda transformación la podemos expresar como O(VE).



En la imagen se muestra el grafo transformado en red de flujo y una posible solución de la asignación f de flujo que puede entregar la caja negra de solución de problema de flujo máximo. La descomposición de flujo nos puede entregar los siguientes 4 caminos arco disjuntos. c_1: o→2→7→d, c_2: o→4→d, c_3: o→3→5→d, c_4: o→1→3→6→d. que corresponde a la solución de la instancia del problema original.

La complejidad total de la reducción corresponderá a la suma de las complejidades de las transformaciones y la cantidad de veces que se ejecuta la caja negra junto a la complejidad de la misma. Podemos sumarizar como O(E+V) + O(1)*O(V^2E) + O(VE). Por lo tanto en forma resumida se puede afirmar que resolver el problema de caminos arco-disjuntos tiene una complejidad temporal de O(V^2E) si utilizamos Dinic para resolver el problema de flujo máximo. Y podemos mejorarlo si usamos un algoritmo con menor complejidad como el propuesto por Orlin.

¿Qué ocurre si utilizamos simplemente Ford-Fulkerson como el algoritmo de caja negra? ¿Es pseudo polinomial el resultado final? Recordemos que la complejidad de FordFulkerson es O(CE) (también se puede expresar como O(C(E+V)). El valor de P se calculaba como la suma de las capacidades de los ejes que salen de la fuente. En este caso al ser todos las capacidades unitarias el valor de C puede corresponder en el caso mayor |E| si todos los ejes parten de la fuente. Por lo tanto reemplazando podemos expresar O(E^2+VE) la complejidad de Ford Fulkerson para cualquier instancia de este problema. Aun utilizando Ford-Fulkerson tendremos como solución final una complejidad totalmente polinomial.

Resta el análisis de la vinculación entre la instancia-solución de cada uno de los problemas involucrados en la reducción. Las instancias de ambos problemas son similares. Se agrega las capacidades unitarias en los ejes como diferencia fundamental. Al buscar caminos de aumento solo se puede utilizar ejes no saturados. Corresponden a ejes aún no utilizados del problema original por caminos arcos-disjuntos. En cada uno de los caminos de aumento el cuello de botella es indefectiblemente el valor unitario. Al aumentar el flujo por cada camino de aumento, satura a todos los ejes que lo componen. Eso impide que puedan ser utilizados por otros caminos. Sabemos que Ford Fulkerson (u otro de los algoritmos de flujo máximo que busca caminos de aumento) finalizará al maximizar el flujo de la red de flujo. Como cada camino de aumento lleva flujo desde la fuente al sumidero y solo pueden ser utilizados los ejes en uno de ellos, entonces los ejes saturados corresponden a caminos arco-disjuntos. La cantidad de caminos de aumento encontrados corresponde a la cantidad total de caminos arco-disjunto que es lo que queríamos realizar.

El corte mínimo de la red de flujo tendrá el mismo valor del flujo enviado desde la fuente, la cantidad de caminos de aumento encontrados y por consiguiente la cantidad de ejes que se deben desconectar para que el origen y destino del grafo original queden desconectados. Que es justamente lo que afirma el teorema de Menger. Con este podremos afirmar que la reducción es correcta y óptima para cualquier instancia posible.

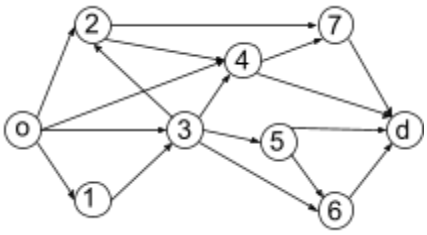
3. Caminos nodo-disjuntos entre dos nodos en el grafo

En el mismo camino que el problema anterior podemos medir la robustez de conectividad de un grafo entre un par de vértices como la cantidad de caminos que se puede formar no solo si repetir ejes, sino también nodos:

Problema de caminos nodo-disjuntos
Dado grafo G=(V,E) dirigido, un nodo o $\in$ V origen y un nodo d $\in$ V destino. Deseamos conocer la cantidad de caminos nodos-disjuntos que se pueden trazar entre ellos.

Dos caminos son nodo-disjuntos entre sí, si no comparten nodos entre ellos. Es decir que queremos conocer la máxima cantidad de caminos independientes entre sí que se pueden trazar desde un nodo origen a uno destino.

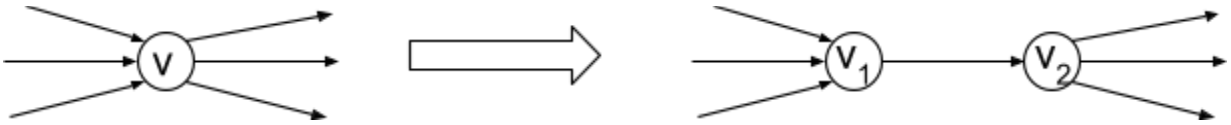
Nuevamente enunciamos el **teorema de Menger** (como también es conocido) pero para caminos nodo-disjuntos: El máximo número de caminos nodos-disjuntos desde el nodo o al d en un grafo es igual al mínimo número de nodos que pueden ser eliminados para que no haya tales caminos.



En el grafo de la imagen queremos encontrar caminos nodo-disjuntos que unen el nodo o con el nodo d. El camino c_1: o→2→7→d y c_2: 0→3→5→d son nodo disjuntos entre sí. El camino c_3: 0→1→3→6→d es nodo-disjunto con c_1, pero no lo es con c_1.

En este caso aprovecharemos el trabajo realizado para resolver el problema de los caminos arco-disjuntos. Realizaremos una reducción polinomial a este problema. Realizaremos una modificación en la instancia para asegurarnos el uso de cada nodo en un camino solo una vez.

En la reducción, la primera transformación consistirá en la generación de un nuevo grafo donde reemplazamos cada nodo v (excepto el nodo origen y destino) por un par de nodos v_1 y v_2 unidos por un eje dirigido. Cada una de las aristas e=(u,v) que arriba al nodo v se modificará para sea (u,v_1). De forma similar, cada eje e=(v,u) que sale de v se reemplazará por (v_2,u). La diferencia en la manipulación de los nodos origen y destino se explica por la necesidad de evitar que el primer camino encontrado impida que nuevos se generen al entender que estos nodos ya fueron utilizados.



El pseudocódigo de la transformación correspondiente:

Transformación a instancia de caminos arco-disjuntos (T1)
Sea G=(V,E) el grafo dirigido Sea o,d ∈ V los nodos de origen y destino respectivamente. Construir G'=(V',E') grafo dirigido Por cada nodo v ∈ V-{o,d}

```
        Crear nodos v1 y v2 en V'
        Crear el eje e=(v1,v2) en E'
Para cada eje e=(u, v) ∈ E
        Si u≠o y v≠d
                Crear el eje e=(u2,v1) en E'
        Sino si u≠o
                Crear el eje e=(d,v1) en E'
        Sino si v≠d
                Crear el eje e=(u2,d) en E'
        Sino
                Crear el eje e=(o,d) en E'

Retornar G'
```

La complejidad corresponde a O(V+E) por crear un nuevo grafo con 2|V| nodos y |V| ejes adicionales.

Una vez obtenido el grafo modificado utilizaremos como caja negra un algoritmo que resuelve el problema de caminos arco-disjuntos. Alternativamente podemos resolverlo utilizando la reducción que se analizó en la sección anterior que termina haciendo uso del problema de flujo máximo. Obtendremos como resultado la cantidad de caminos arco-disjuntos |P| y los caminos en sí.

Llegados a este punto debemos transformar la solución de caminos arco-disjuntos en la de nuestro problema: poder determinar la cantidad de caminos disjuntos sin utilizar los mismos nodos. Esto no es complicado: se deben reemplazar los pares de nodos ficticios por el nodo al que representan en el grafo original. En cada camino tendremos indefectiblemente un conjunto par de nodos. El primero y el último corresponden a los nodos origen y destino. Y luego cada par de nodos corresponde a un nodo en el grafo original.

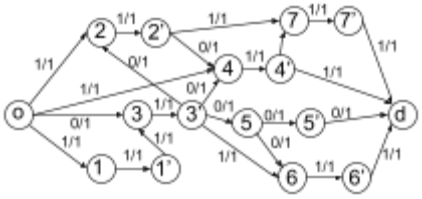
Transformación de la solución del problema de arco-disjunto a solución (T2)

```
Sea G el grafo original
Sea G' el grafo transformado
Sea P los caminos arco-disjuntos de G'
Sea |P| la cantidad de caminos arco-disjuntos.

Por cada camino p=(o,n1,n2...,d) en P
```

Mantener o y d.
Reemplazar cada par de nodos n_i y n_{i+1} por nodo el nodo original en G en P.

Retornar P y |P|

La cantidad de caminos arcos-disjuntos obtenidos del grafo modificado corresponde a la cantidad de caminos nodo-disjuntos del problema original. Y los caminos corresponden a los obtenidos volviendo a reemplazar los nodos duplicados por su nodo del grafo original. Como máximo tendremos |V| caminos (si cada camino pasa por un nodo diferente) o una menor cantidad de caminos pero con como mucho una longitud de |V|. Podemos entonces afirmar que recorrer los caminos y transformarlo es un proceso global O(V).



La imagen corresponde a la red de flujo modificada luego de las transformaciones hasta llegar a un problema de flujo máximo. Además expresa una posible solución correspondiente a 3 caminos nodo-disjuntos. Se puede descomponer el flujo para armar los caminos de aumento. Por ejemplo c_1: o→2→2'→7→7'→d que en el grafo original corresponde a c_1:o→2→7→d. El resto de los caminos son c_2: o→4→d y c_3: o→1→3→6→d.

La complejidad total del algoritmo corresponde a sumar las complejidades de las dos transformaciones y de la caja negra utilizada.

Podemos analizar la relación entre las instancias-soluciones de los problemas en la reducción. Al realizar el reemplazo de cada nodo por un par de nodos unidos por un eje direccionado estamos obligando que cualquier camino del origen al destino deba pasar por estos ejes "ficticios". Entonces esto restringe la posibilidad que cualquier otro camino puede pasar por él mismo, lo que equivale a decir que ningún otro camino podrá utilizar el mismo nodo. Esta construcción en la nueva instancia del problema que simula o representa parte del funcionamiento del problema original se conoce como ***artilugio*** (gadget). Utilizaremos profusamente a los gadgets y serán de mucha utilidad en las gran cantidad de reducciones.

Este mismo gadget puede ser tomado para resolver otro tipo de problema: las redes de flujo con capacidad en los nodos. Esta capacidad limita el máximo volumen de flujo que puede conmutar cada nodo. Este problema se puede reducir a un problema de flujo

máximo utilizando el gadget y agregando la capacidad del nodo en la capacidad del eje adicionado.

4. Emparejamientos en conjuntos disjuntos.

Previamente analizamos el problema de los emparejamientos estables (stable matching problem). En ese problema partíamos de dos conjuntos disjuntos que representaban diferentes conceptos entre los que se debía armar parejas. Se demostró que si ambos conjuntos eran del mismo tamaño y que se contaba con un listado de preferencias completo de cada elemento, se podía construir un matching estable.

A continuación planteamos una variante de este problema donde para ciertas instancias no es posible emparejar a todos los elementos. En primer lugar quitaremos la restricción de que ambos conjuntos tengan la misma cantidad de elementos. Por otro lado supondremos que no todas las parejas son compatibles entre sí. Es decir que existirán pares de elementos que pueden formar parejas y otras no. Por último, por un instante removemos el concepto de preferencia. Todas las contrapartes con las que puede formar un elemento pareja tienen la misma preponderancia para sí. Trabajaremos con estas hipótesis y lo que buscaremos entonces es maximizar la cantidad de parejas que se pueden construir.

Problema de maximización de emparejamientos
Dados un par de conjuntos A y B, un conjunto T de tuplas (a,b) con a $\in$ A y b $\in$ B de posibles emparejamientos. Deseamos construir la mayor cantidad de parejas posibles sin que los elementos aparezcan más de una vez en ellas.

Planteamos el siguiente ejemplo: el conjunto A tiene los elementos $\{a_1, a_2, a_3, a_4\}$ y el conjunto B los elementos $\{b_1, b_2, b_3, b_4, b_5\}$. Los posibles emparejamiento T corresponden a $\{(a_1,b_1), (a_1,b_2), (a_1,b_5), (a_2,b_1), (a_2,b_5), (a_3,b_5), (a_4,b_2), (a_4,b_3), (a_4,b_4)\}$. Por ejemplo, si incluimos la pareja (a_1,b_5) no podremos incluir (a_1,b_1) ni (a_3,b_5). Pero si podemos agregar el emparejamiento (a_4,b_2).

Para resolver este problema transformaremos la instancia del problema en un grafo bipartito. Un grafo es bipartito si sus nodos se pueden separar en dos conjuntos disjuntos, donde los ejes existentes únicamente conectan nodos que se encuentran en diferente

conjunto. Luego el grafo se transformará en una red de flujo donde la fuente se conectará mediante ejes con los elementos de uno de los conjuntos y el sumidero con los del otro. Todos los ejes tendrán una capacidad unitaria.

La primera transformación comienza al crear 2 nodos para representar los ejes y sumideros. Continuaremos con |A|+|B| nodos que representan a cada uno de los conjuntos. Luego se crean |A| ejes uniendo la fuente con los nodos que representan a los elementos de A. Similarmente se crean |B| ejes uniendo los nodos que representan a los elementos de B con el sumidero. Finalmente se representa con ejes entre los nodos que representan a los elementos de A y B que pueden conformar una posible pareja. Estos nodos serán direccionados. En el caso más completo existirán |A|*|B| de estos ejes. A todos los ejes se les otorgará, como se explicó, como capacidad el valor 1.

El pseudocódigo de la transformación correspondiente:

Transformación a instancia de problema de flujo máximo (T1)

```
Sea A y B conjuntos
Construir G=(V,E) red de flujo

Crear s fuente, insertarla en V
Crear t sumidero, insertalo en V
Por cada elemento a en A
        Crear un nodo n, insertarlo en V
        Crear un eje e=(s,n), insertarlo en E

Por cada elemento b en B
        Crear un nodo n, insertarlo en V
        Crear un eje e=(n,t) con capacidad Ce=1, insertarlo en E

Por cada tuplo t=(a,b) en T
        Crear un eje e=(a,b) con capacidad Ce=1, insertarlo en E

Retornar red de flujo G
```

Crear la instancia de la red de flujo tendrá un complejidad O(A*B).

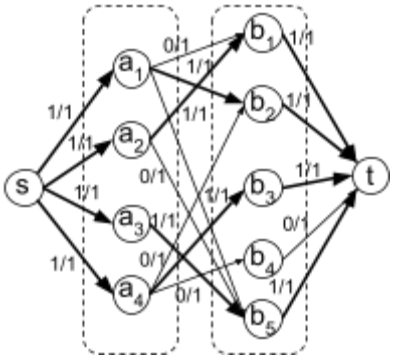
Una vez obtenida la red de flujo, debemos utilizar como caja negra un algoritmo para resolver el problema de flujo máximo. La complejidad de esta ejecución no debemos medirla en función de vértices y nodos, sino en función de los parámetros de nuestro problema original. Por la forma de construir la red podemos decir que la instancia de la red G=(V,E), tendrá |A|+|B|+2 nodos y como mucho |A|*|B|+|A|+|B| ejes. Supondremos que la caja negra ejecuta Ford-Fulkerson. Entonces la complejidad O(C(V+E)) se puede reemplazar en forma simplificada por O(C(A*B)). C es la suma de las capacidades que salen de la fuente y en este caso son capacidades unitarias. Tenemos |A| ejes que salen de la fuente y por lo tanto podemos expresar C como |A| y la expresión de Ford Fulkerson como $O(A^2B)$.

Obtendremos como solución una asignación de flujo máximo. Debemos transformarla en la solución de nuestro problema original. Para eso vemos que el valor de flujo obtenido corresponde a la cantidad total de parejas armadas. Además aquellos ejes saturados que unen nodos que representan elementos del conjunto de A con los del B representan las parejas formadas. Podemos entonces expresar esta segunda transformación como:

Transformación de la solución del problema de flujo máximo a solución (T2)
Sea G=(V,E) red de flujo Sea f flujo en la red Establecer |P| como cantidad de parejas la suma de los flujos que salen de la fuente. Por cada eje e=(u,v) en G con u≠s y v≠t Agregar pareja en P el elemento de A que representa u y el elemento de B que representa v. Retornar |P| y P

La complejidad de esta segunda transformación es O(A*B). La suma de todas las complejidades de las partes involucradas conforman la complejidad final para resolver el problema utilizando la reducción.

Para el ejemplo planteado, expresamos los conjuntos y sus emparejamientos posibles como un grafo bipartito y luego este como una red de flujo con el agregado de la fuente, el sumidero y en



cada eje un sentido y capacidad unitaria. La solución del problema del flujo máximo nos brinda la información necesaria para construir las parejas. Una posible construcción de 4 parejas corresponde a (a_1,b_2) , (a_2,b_1), (a_3, b_5) y (a_4,b_3).

Se puede resolver este problema con algunas variantes en la reducción. Cuál de los subconjuntos se conectan con la fuente y cual al sumidero y el sentido por lo tanto de los ejes puede invertirse sin afectar la solución. También es posible pensar que un conjunto corresponde a productores y el otro consumidores. Todos ellos con demanda - 1 y 1 respectivamente y resolverlo como un problema de circulación con demanda. En este caso aun no existiendo una circulación válida, como en última instancia se termina resolviendo con flujo máximo, podemos aprovechar el grafo residual para resolver nuestro problema original.

Una variante a este problema puede incluir cierta preferencia en el armado de algunas parejas sobre otras. Por ejemplo incluir una "penalidad de pareja". Cuanto mayor la penalidad, menos preferente. En este caso se puede reducir al problema de flujo máximo y costo mínimo. El resto del proceso es el mismo.

5. Diseño de encuestas

Podemos extender el problema de pareo entre dos conjuntos a casos más generales. En el ejemplo que trabajaremos a continuación tenemos también dos conjuntos: clientes y productos. En este caso lo que queremos es determinar cómo realizar una encuesta de satisfacción preguntando a clientes que opinan de los productos. Esta encuesta debe cumplir con ciertos requisitos adicionales.

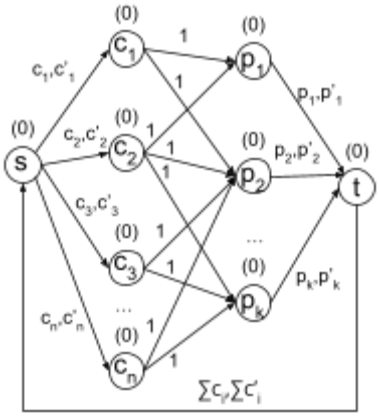
Queremos tener información de todos los clientes y sobre todos los productos. La opinión únicamente es válida si nos llega de un cliente que efectivamente compró un producto determinado. El deseo es no sobrecargar a los clientes con demasiadas preguntas pero tampoco desaprovechar el proceso al enviar muy pocas. Además queremos tener opiniones sobre todos los productos. Teniendo de cada uno de ellos también una mínima cantidad y una máxima. Podría interesar conocer más sobre ciertos clientes o sobre ciertos productos, por lo que los mínimos y máximos deben ser personalizados.

Formalizaremos el problema de la siguiente manera.

Problema de diseño de encuestas
Sean “k” productos que vende una empresa y “n” clientes que realizaron compras a la empresa. Se desea construir una encuesta de satisfacción “personalizada” con una serie de restricciones. Cada cliente puede responder únicamente por productos que haya comprado. El cliente “i” puede responder consultas sobre no más c'_i y no menos de entre c_i productos. El producto “j” debe tener entre p_j y p'_j respuestas de clientes

Para resolver este problema realizaremos una reducción a un problema de circulación con demandas y límite inferior. Crearemos una red de flujo, usaremos como caja negra un solucionador del problema seleccionado y luego transformaremos la solución obtenida a la solución del problema original. Sabemos cómo resolver el problema de circulación con demandas y límite inferior reduciéndolo a un problema al problema de circulación con demanda. Este último también sabemos resolverlo reduciéndolo a un problema de flujo máximo. Aprovecharemos en este caso para hacer un paso a paso completo.

Para la transformación al problema de circulación con demanda y límites comenzaremos creando un grafo G=(V,E). Luego:



- Por cada cliente “i”, agregamos un nodo Ci en V con demanda 0
- Por cada producto “j”, agregamos un nodo Pj en V con demanda 0.
- Creamos el nodo s (fuente) en V con demanda 0
- Creamos el nodo t (fuente) en V con demanda 0
- Por cada producto “j” que compró el cliente “i”, creamos el eje e=(Ci,Pj) en E con capacidad 1.
- Por cada cliente “i” agregamos el eje e=(s,Ci) con capacidad ci y límite inferior c’i.
- Por cada producto “j” agregamos el eje e=(Pj,t) con capacidad pj y límite inferior p’j.
- Creamos el eje e=(t,s) con capacidad igual a la suma de los ci y límite inferior igual a la suma de los c’i de todos los clientes.

En pseudocódigo lo podemos expresar de la siguiente manera:

Transformación a instancia de problema de circulación con demanda y límites (T1)

```
Sea P los productos y C los clientes
Construir G=(V,E) red de flujo

Crear s fuente, insertarla en V
Establecer demanda de s igual a cero.

Crear t sumidero, insertalo en V
Establecer demanda de t igual a cero.

Por cada elemento c en C
        Crear un nodo c', insertarlo en V
        Establecer demanda de c igual a cero.
        Crear un eje e=(s,c'), insertarlo en E
        Establecer ci como capacidad del eje e
        Establecer c'i como límite inferior del eje e

Por cada elemento p en P
        Crear un nodo p', insertarlo en V
        Establecer demanda de c igual a cero.
        Crear un eje e'=(p',t), insertarlo en E
        Establecer pi como capacidad del eje e'
        Establecer p'i como límite inferior del eje e'

Por cada producto p que un cliente c compro
        Crear un eje e"=(c',p'), insertarlo en E
        Establecer 1 como capacidad del eje e"
        Establecer 0 como límite inferior del eje e"

Crear un eje e'"=(t,s), insertarlo en E
Sea x la suma de los ci de todos los clientes
Sea y la suma de los c'i de todos los clientes
Establecer x como capacidad del eje e'"
Establecer y como límite inferior del eje e'"

Retornar red de flujo G
```

La complejidad de la transformación está dada por O(k*n) dado que en el peor de los casos todos los clientes compraron todos los productos.

Utilizaremos como caja negra un algoritmo que resuelva el problema de circulación con demanda y límites. Tendremos como resultado la red de flujo con una función de flujo f que representa la circulación.

El análisis de la circulación y la red de flujo nos retornará la solución del problema original. En primer lugar si la circulación es inválida podremos afirmar que no se puede construir la encuesta con los parámetros solicitados. Por otro lado si la circulación es válida obtendremos de la misma la siguiente información:

- El flujo del eje e=(t,s) contendrá la cantidad de preguntas totales a realizar
- El flujo de cada eje e=(s,Ci) contendrá la cantidad de preguntas que se le debe realizar el cliente i
- El flujo de cada eje e=(Pj,t) contendrá la cantidad de preguntas a realizar sobre el producto j.
- Aquellos ejes e=(Ci,Pj) saturados (con flujo igual a 1) corresponden a una pregunta a realizar al cliente i acerca del producto j.

Expresado como pseudocódigo:

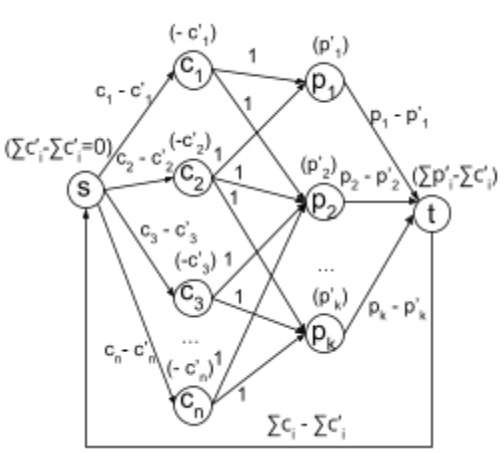
Transformación de la solución de circulación con demanda y límites a solución (T2)

```
Sea G=(V,E) red de flujo
Sea f circulación en la red

Establecer la cantidad de preguntas como f(e) con e=(t,s)
Por cada cliente i
        Establecer la cantidad de preguntas a i como f(e) con e=(s,Ci)
Por cada producto j
        Establecer la cantidad de preguntas a j como f(e) con e=(Pj,t)
Por cada cliente i
        Por cada producto j
                Si existe el eje e=(Ci,Pj) y f(e)=1
                        Preguntar al cliente i sobre el producto j
```

La complejidad de esta segunda transformación está dada por O(k*n)

Si la circulación es inválida puede ser informada por la caja negra que resuelve el problema de circulación, sin embargo si no lo hace se puede verificar. Se debe constatar que para cada nodo se cumpla la conservación de oferta/demanda y por cada eje la restricción de capacidad.

Antes de finalizar veamos cómo se transforma el problema dentro de la caja negra primero en un problema de circulación con demandas y luego en uno de flujo máximo. De esa forma se podrá comprender con más detalle cómo es que la reducción funciona para cualquier instancia del problema original.

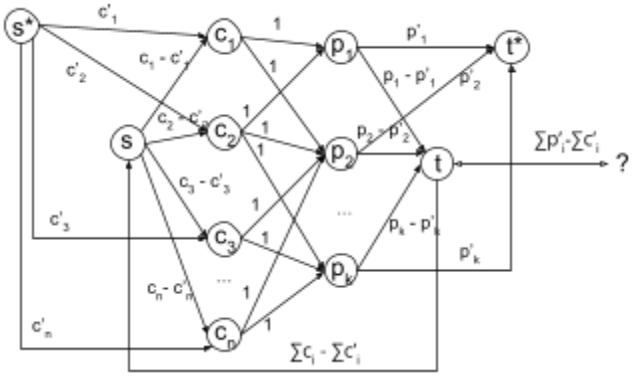


La imagen que acompaña a este párrafo indica cómo se transforma el problema a uno de circulación con demandas (sin límites). Adicionalmente se debe crear y reservar pseudo circulación para reconstruir el resultado final. Se puede observar que los nodos que representan a los clientes pasan a ser nodos productores con demanda equivalente a la cantidad mínima de preguntas que debe responder. De igual manera los nodos que representan a los productos se convierten en nodos consumidores con demanda igual a la cantidad mínima de preguntas que deben recibir. Los ejes que llegan a los nodos “clientes” tiene como capacidad el número de preguntas adicionales que puede responder el cliente sin excederse del máximo. Lo mismo ocurre con los ejes que salen de los nodos “productores”. El nodo “s” responde a uno interno (sin demanda). Por otro lado el nodo “t” podrá ser consumidor, interior o productor. Depende si la cantidad total de preguntas por productos es mayor, igual o menor que la cantidad total de preguntas a clientes (respectivamente). Esa cantidad de preguntas son las requeridas para balancear los mínimos entre el total de preguntas por productos y de clientes.

Si el nodo t se transforma en productor ocurre que para alcanzar la mínima cantidad de preguntas para los productos se requiere más que la mínima cantidad de preguntas a los clientes. Justamente la demanda de t corresponde a este valor. Para llevarla a destino utilizará algún camino que inicie por el eje e=(t,s). Cuya capacidad indica el tope de

preguntas extras en total a cliente que se pueden adicionar. El camino continua a través de alguno de los ejes que sale de s (limitados por la posibilidad de responder por encima de su valor mínimo). Si el nodo t se transforma en consumidor ocurre que para alcanzar la mínima cantidad de preguntas para los clientes se requiere más que la mínima cantidad de preguntas a los productos. El flujo deberá llegar desde alguno de los nodos productores (clientes). Finalmente si el nodo t se transforma en un nodo interno, entonces las cantidades mínimas entre los clientes y productos son iguales. Por lo tanto no se requiere generar adicionales.

Es importante remarcar que por este tipo de construcción se encontrará el mínimo de preguntas que cumpla los requisitos aun siendo posible consultar más . Esto debido a la naturaleza del problema de circulación con demandas y la manera de construir la red.



Finalmente para construir la red de flujo para resolver el problema de flujo máximo se agregan los nodos super fuente s* y super sumidero t* y se crearán los correspondientes ejes uniendolos con los productores y consumidores respectivamente. Con cuál se unirá el nodo t (y el sentido del eje) dependerá de la instancia particular del problema. La imagen que acompaña a este párrafo muestra esta última transformación. La solución de cada una de las reducciones nos llevará a la solución del problema original. La suma de las complejidades de cada proceso realizado corresponderá a la complejidad final del método de resolución propuesto.

6. Selección de proyectos

Hay problemas que a primera vista no parecen tener relación con problemas de flujo máximo. Sin embargo, utilizando una reducción de forma ingeniosa los podremos resolver de forma eficiente. Analizaremos un caso que llamaremos “selección de proyectos” que fue propuesto y resuelvo por Barrie Baker[1] en 1984. En este problema contamos con una

[1] A Network-Flow Algorithm for Project Selection, Baker B. M., 1984, The Journal of the Operational Research Society, 35(9), 847.

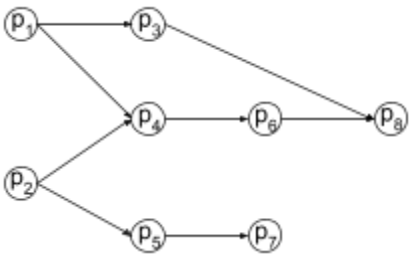
jerarquía de proyectos que podemos seleccionar para llevar adelante. Comenzar a ejecutar un determinado proyecto puede requerir completar cero, uno o más proyectos previos. Al mismo tiempo finalizar un proyecto puede ser requisito para iniciar cero, uno o más proyectos posteriores. Cada proyecto tiene asociado un valor económico. Si el mismo es positivo corresponde a una ganancia y si es negativo, pérdida.

El objetivo es seleccionar un subconjunto factible de proyectos que nos permitan maximizar la ganancia. Diremos que un subconjunto es factible si los prerrequisitos de cada proyecto que contiene también pertenecen al mismo.

Formalizaremos el problema como:

Problema de selección de proyectos
Sean un conjunto P de proyectos a realizar. Cada proyecto i produce un resultado económico p_i que puede ser positivo (ganancia) o negativo (pérdida). Además cada proyecto puede tener un subconjunto de otros proyectos de P que son prerrequisitos para poder realizarse. Deseamos seleccionar un subconjunto de proyectos en P de forma de lograr maximizar la ganancia a obtener.

Resolveremos el problema mediante una reducción a un problema de corte mínimo, utilizaremos como caja negra un algoritmo que resuelva y luego transformaremos la solución a la de nuestro problema original. La primera transformación debe construir una red de flujo. Comenzaremos analizando cómo podemos generar un grafo en base a la instancia del problema. Podemos pensar la jerarquía de los proyectos como un grafo dirigido y acíclico o DAG (Directed acyclic graph). Sería ilógico suponer que existe un ciclo formado por un conjunto de proyectos, dado que eso impediría ejecutar ninguno de ellos. Cada proyecto corresponderá a un nodo. Dos proyectos que tienen una dependencia tendrán un eje que los una en el grafo. El eje saldrá del nodo que representa al proyecto prerrequisito y se unirá al nodo que representa al proyecto siguiente.



Contamos con un conjunto de 8 proyectos posibles a realizar. Representaremos a cada uno de ellos como una tupla con su identificador y su resultado económico (p_i,g_i): (p_1, -3), (p_2,1), (p_3,4), (p_4,2), (p_5,-5), (p_6,-5), (p_7,3), (p_8,4). Además conocemos

los prerrequisitos de cada problema. Los representaremos como (p_i:lista de prerrequisitos). Estos son ($p_3 : p_1$), ($p_4 : p_1,p_2$), ($p_5 : p_2$), ($p_6 : p_4$), ($p_7 : p_5$), ($p_8 : p_3,p_6$). La imagen que acompaña el párrafo corresponde al DAG que representa las precedencias de los proyectos.

Una solución factible, conformada por un subconjunto A, nos brindará una ganancia igual a la suma de resultados económicos de los proyectos que la componen. Matemáticamente lo representamos como: $Ganancia(A) = \sum_{i \in A} g_i$. El tope de ganancia corresponde a la suma de los resultados económicos de todos los proyectos con valor positivo. Sabemos que ganar más que eso resulta imposible. Este tope $T = \sum_{i \in P \,/\, g_i > 0} g_i$ puede corresponder a un subconjunto que no sea factible. Sin embargo conocer su valor nos resultará útil más adelante.

Para la transformación al problema de corte mínimo comenzaremos creando un grafo G=(V,E). Luego:

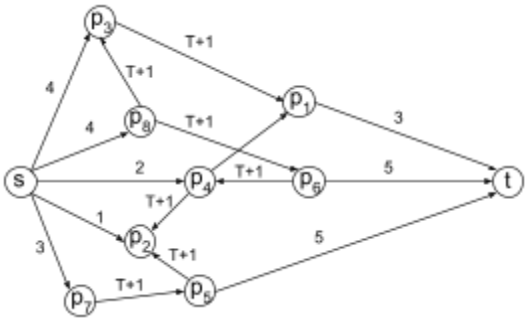
- Creamos el nodo s (fuente) en V
- Creamos el nodo t (fuente) en V
- Creamos un nodo Pi por cada proyecto i
- Por cada proyecto "i" con retorno económico g_i positivo, creamos el nodo e=(s,Pi) en E con capacidad gi.
- Por cada proyecto "i" con retorno económico g_i negativo, creamos el nodo e=(Pi,t) en E con capacidad -gi.
- Por cada proyecto "j" que precede al proyecto "i" creamos el eje e=(Pi,Pj) con capacidad igual al tope de ganancia T más uno.

Expresado en pseudocódigo:

Transformación a instancia de problema de corte mínimo (T1)
Sea P los proyectos Sea T el tope de ganancia Construir G=(V,E) red de flujo Crear s fuente, insertarla en V

```
Crear t sumidero, insertalo en V

Por cada proyecto i en P
        Crear el nodo Pi
        Si gi < 0
                Crear el eje e=(pi,t) con capacidad -gi
        Si gi > 0
                Crear el eje e=(s,pi) con capacidad gi

Por cada proyecto j predecesor de proyecto i
        Crear el eje e=(pi,pj) con capacidad T+1

Retornar G
```

La complejidad de la transformación corresponde a $O(P^2)$. Este valor se explica por la cantidad máxima de ejes dentro de un DAG. Si un DAG tiene n nodos, la máxima cantidad de ejes posibles corresponde a (n-1)(n)/2. Agregar cualquier eje adicional provocará un ciclo. Por lo tanto agregar al grafo las relaciones de precedencia de P proyectos como máximo incorpora a la complejidad $O(P^2)$. El resto de los procesos en la transformación es O(P).



En el ejemplo anterior calculamos el tope de ganancia T como la suma de los proyectos con valor positivo $p_2+p_3+p_4+p_7+p_8 = 1+4+2+3+4 = 14$. *En el armado de la red de flujo estos proyectos serán aquellos nodos que reciben un eje de la fuente. Sus capacidades corresponden a la ganancia que nos otorgan realizarlos. Los nodos de los proyectos p1, p5 y p6 tendrán un eje que sale de ellos y finaliza en el sumidero. Su capacidad, el opuesto de su retorno económico (pérdida). Los ejes de precedencia entre proyectos son los mismos que en el DAG pero con el sentido invertido. La capacidad de estos ejes corresponde al tope de ganancia más una unidad.*

Una vez obtenida la red de flujo, debemos utilizar como caja negra un algoritmo para resolver el problema de flujo máximo/corte mínimo. Obtendremos como solución una asignación de flujo y el corte mínimo. Deberemos transformar esta, en la solución del problema original.

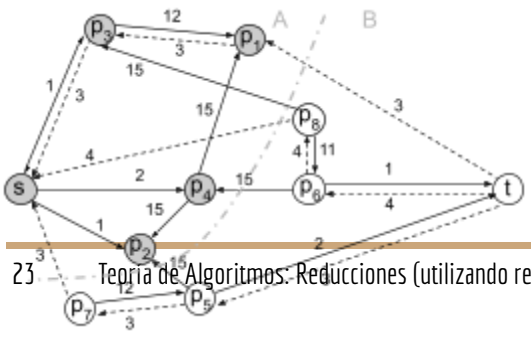
Como retorno de la caja negra tendremos el grafo con su correspondiente función de asignación de flujo f. Con este podremos reconstruir el grafo residual. Utilizando DFS o BFS podemos determinar cuales son los nodos accesibles desde la fuente. Estos serán aquellos proyectos que debemos realizar para maximizar la ganancia o el conjunto solución S. Podemos sumar los retornos económicos de estos proyectos y conocer el valor de máxima ganancia. Está relacionado con el valor c(A,B) de corte mínimo (o flujo máximo). El tope de ganancia menos el flujo máximo es equivalente a la ganancia máxima: Ganancia(S) = T - c(A,B)

Expresado como pseudocódigo:

Transformación de la solución de corte mínimo a solución (T2)
Sea G=(V,E) red de flujo Sea f flujo en la red Sea Gr grafo residual resultante de aplicar f en G Sea T el tope de ganancia Establecer como S a los nodos de Gr accesibles desde la fuente (excluyendola) en G Establecer como Ganancia(S) la diferencia entre T y el flujo máximo. Retornar S y Ganancia(S)

La complejidad de esta transformación corresponde a obtener el valor del flujo máximo - se puede obtener en O(E) - y obteniendo los nodos accesibles desde la fuente utilizando el grafo residual en O(V+E). Debemos expresar estas complejidades en función de las variables del problema original. Como la cantidad de Ejes son $O(P^2)$ y la cantidad de nodos es O(P), la expresión de complejidad final de esta segunda transformación es $O(P^2)$

La solución del problema de corte mínimo la obtenemos mediante el cálculo del flujo máximo. La imagen muestra el grafo residual Gr luego del cálculo del flujo máximo. Podemos observar que el flujo máximo es 10. Los nodos que son accesibles en el grafo residual desde la fuente corresponden a los proyectos p_1, p_2, p_3, p_4. Son en el corte mínimo C(A,Ḃ) el subconjunto A (y también la solución S del problema) Esta solución es factible puesto que todos los proyectos elegidos tienen en S



también sus predecesores. Podemos calcular su ganancia como Ganancia(S) = g_1+g_2+g_3+g_4 = -3+1+4+2=4. El corte mínimo está conformado por los ejes que pasan de de A' a B' en G. Estos son p1→t, s→p7 y s→p8. Sumando sus capacidades 3+3+4 = 10 que corresponde al corte mínimo (que de forma esperada es igual al flujo máximo de la red). Si realizamos la operación T-Ganancia(S)=14-4=10 equivale al valor del corte mínimo c(A ,B)=10. Podemos observar la relación entre el corte mínimo y la máxima ganancia posible.

Veamos a continuación cómo es que llegamos a la relación entre el corte mínimo y la ganancia máxima.

Podemos clasificar en tres conjuntos a los ejes del grafo:

- Aquellos ejes que salen de la fuente: Representan la ganancia de los proyectos con retorno positivo.
- Aquellos ejes que llegan al sumidero: Representan la pérdida de los proyectos con retorno negativo.
- Aquellos ejes que representan precedencias entre proyectos

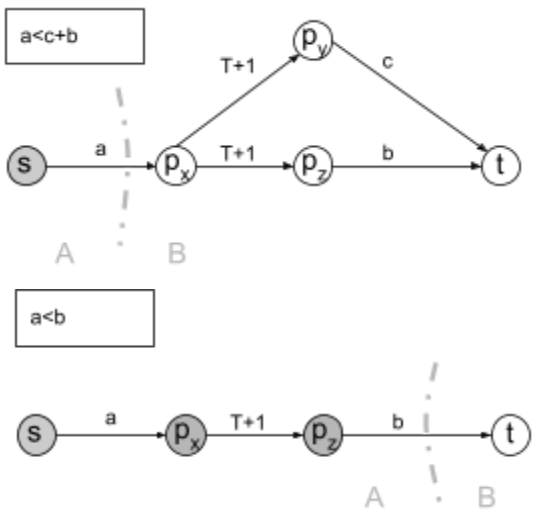
Un caso extremo corresponde a la no existencia de proyectos con ganancia negativa. En este caso el valor de ganancia tope T equivale a la ganancia máxima. $T = \sum_{i \in P \,/\, g_i > 0} g_i = \sum_{i \in P} g_i$

El grafo inicial no tiene caminos entre la fuente y el sumidero. El flujo máximo en el grafo es cero. El corte mínimo C(A,B) deja de una lado a todos los nodos excepto al sumidero. El valor del corte es igual a la suma de las capacidades de los ejes que llegan al sumidero y por lo tanto c(A,B)=0. Vemos que la ganancia máxima Ganancia(S) es igual al tope de ganancia: Ganancia(S) = T - c(A,B) = T - 0= $\sum_{i \in P} g_i$. Se cumple la relación planteada: se deben seleccionar todos los proyectos.

Un caso similar corresponde al caso extremo donde todos los proyectos tienen ganancia negativa. En ese caso nuevamente no hay camino entre fuente y sumidero. El corte mínimo en este caso deja sola a la fuente y el valor del corte es cero. El tope de ganancia en este caso es cero $T = \sum_{i \in P \,/\, g_i > 0} g_i = 0$. Y la Ganancia de la solución de mayor ganancia posibles

Ganancia(S) = T - c(A,B) = 0. También se cumple la relación planteada: no se debe seleccionar ningún proyecto.

Veamos ahora casos más generales. Para que exista flujo en el grafo un camino de aumento debe pasar por un mínimo de 2 nodos internos (además de la fuente y el sumidero): el primero uno que represente un proyecto con retorno positivo y el último un proyecto con una retorno económico negativo. Al buscar el cuello de botella del camino corresponderá o al primer eje del camino o al último. Aquellos ejes que representan precedencias no pueden saturarse bajo ninguna circunstancia dado que su capacidad es mayor a la máxima cantidad de flujo que puede salir de la fuente (la suma de las capacidades de los ejes que llegan a los proyectos con ganancia positiva equivalente a la ganancia máxima). Esto nos asegura que ningún eje de precedencia puede estar atravesando el corte mínimo de A a B. Por lo que es imposible seleccionar un proyecto sin seleccionar sus predecesores (tienen que estar del mismo lado del corte).



Los ejes que atraviesan el corte mínimo entonces son un subconjunto de aquellos que salen de la fuente o de aquellos que llegan al sumidero. Que se haya saturado un eje e=(s,Px) que sale de la fuente implica que la suma de las capacidades de los ejes e'=(Pj,t) que utilizan los caminos de aumento que pasan por e tiene un mayor valor de capacidad en conjunto. Es decir que llevar a cabo el proyecto x no es conveniente dado que el costo de sus precedentes es mayor al beneficio de llevarlo a cabo. De forma similar, que se haya saturado un eje e=(Pz,t) que llega al sumidero indica que el costo de los proyectos "i" posteriores al proyecto "z" otorgan una ganancia mayor a la pérdida de este último y por lo tanto conviene realizarlo.

En última instancia el valor del corte mínimo c(A,B) tendrá el aporte de algunos proyectos con costo positivo cuyo eje asociado atraviesa el corte partiendo de la fuente y el aporte de algunos proyectos con costo negativo cuyo eje asociado atraviesa el corte llegando al

sumidero. Los primeros contribuyen $\sum_{i \notin A \,/\, g_i > 0} g_i$ y los segundos contribuyen al corte en $\sum_{i \in A \,/\, g_i < 0} -g_i$. Unificando $c(A,B) = \sum_{i \in A \,/\, g_i < 0} -g_i + \sum_{i \notin A \,/\, g_i > 0} g_i$. Recordando la fórmula de tope de ganancia $T = \sum_{i \in P \,/\, g_i > 0} g_i = \sum_{i \notin A \,/\, g_i > 0} g_i + \sum_{i \in A \,/\, g_i > 0} g_i$ podemos reescribir el valor del corte mínimo como $c(A,B) = \sum_{i \in A \,/\, g_i < 0} -g_i + \left(T - \sum_{i \in A \,/\, g_i > 0} g_i \right)$ y unificando las sumatorios concluimos en $c(A,B) = T - \sum_{i \in A} g_i$ que generaliza los casos y nos acerca al resultado esperado. Dado que el valor tope T es constante para la instancia, al encontrar el corte mínimo, encontramos el valor de ganancia más grande. En definitiva nuestra reducción nos brindará el resultado óptimo para cualquier instancia del problema original.

7. Segmentación de imágenes.

Dentro de la edición de imagen la separación de un objeto del fondo es una funcionalidad ampliamente utilizada. Este problema tiene varias soluciones posibles, una de ellas relacionada con el problema del corte mínimo. A continuación analizaremos esta propuesta y elaboramos la reducción polinomial para resolverla. La misma fue presentada y propuesta en 1989 por Greig, Porteous y Seheult[2]

Trabajaremos la imagen como un conjunto de píxeles que forman una cuadrícula de un determinado alto y ancho. Cada pixel o punto de la grilla contiene un valor numérico que representa un color. Un subconjunto de estos píxeles corresponden a un objeto o primer plano mientras que el resto de los píxeles corresponden al fondo o segundo plano.

Con el objetivo de hacer la división contaremos con información adicional que nos llegará como parte de la instancia del problema. Como se realiza la obtención de la misma excede el alcance de la explicación. En primer lugar tendremos por cada píxel dos valores enteros positivos. Llamaremos a_i a la probabilidad de que el pixel i pertenezca al primer plano o

[2] Exact maximum a posteriori estimation for binary images, D. Greig, B. Porteous, and A. Seheult, 1980, J. Royal Statistical Society B, 51:2, pp. 271–278.

objeto. Llamaremos b_i a la probabilidad de que el pixel i pertenece al segundo plano o fondo. Si vemos a cada píxel como un elemento independiente basta con que $a_i > b_i$ para considerar que i pertenece al primer plano. Sin embargo, queremos que esta decisión también dependa de su vecindad.

Llamaremos vecindad de un píxel al conjunto de los píxeles que se encuentran a cierta distancia del mismo. Hay diferentes formas de construir la vecindad. Diremos que un píxel es vecino a otro si se encuentra en contacto con la cara del otro. Un píxel interior de la imagen tendrá 4 vecinos, en la diagonal tendrá 2 y en el borde de la cuadrícula, 3. La importancia de la vecindad radica en que habitualmente los objetos están formados por píxeles contiguos y por lo tanto es esperable que los mismos se mantengan juntos. Llamaremos penalidad de separación pij a un valor numérico entero positivo que castiga la separación de los pixeles vecinos "i" y "j" en diferente conjunto final. Es decir si el pixel i se encuentra en el fondo y el pixel j e el frente (o viceversa) se computará esta penalidad.

Al segmentar una imagen en fondo y objeto nuestro conjunto de píxeles se dividirá en dos conjuntos. Llamaremos A al conjunto de píxeles en la separación que pertenecen al primer plano y B al conjunto de píxeles que pertenecen al fondo. La intersección de los estos conjuntos es vacía y su unión es el conjunto total de píxeles de la imagen.

Existen un número exponencial de posibles segmentaciones de una imagen de n píxeles. Algunas de estas serán preferentes a otras. Queremos poder comparar numéricamente entre dos separaciones propuestas para fácilmente reconocer cuál de ellas es más conveniente. Esperamos que aquellos píxeles en A tengan en conjunto un valor de ai elevado y aquellos en B uno de bi en la misma sintonía. Además que la sumatoria de las penalidades de aquellos vecinos que se encuentran en conjuntos diferentes sea lo menor posible. Matemáticamente expresaremos la calidad de la segmentación como $q(A,B) = \sum_{i \in A} a_i + \sum_{i \in B} b_i - \sum_{i \in A, j \in B} p_{ij}$ y nuestro objetivo será encontrar aquella que maximice este valor.

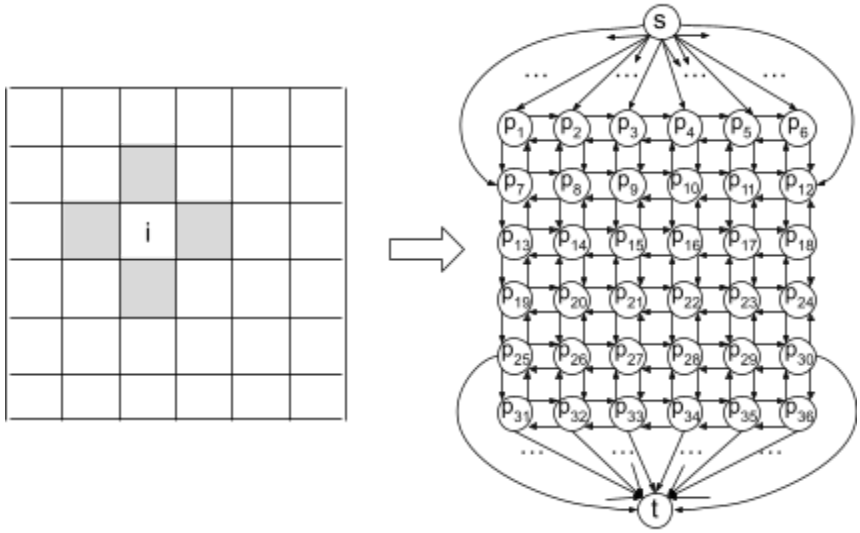
Formalizaremos entonces el problema:

Problema de segmentación de imagen

Sea una imagen representada como una cuadrícula de “n” píxeles . Donde cada pixel i tiene una probabilidad ai de pertenecer al primer plano y una probabilidad bi de pertenecer al segundo plano. Cada pixel i vecino de un pixel j tiene una penalidad pij por estar en diferente plano. Queremos separarla en primer y segundo plano de forma de maximizar la calidad de la segmentación.

A continuación elaboramos la reducción polinomial al problema del corte mínimo. La transformación de la instancia comenzará creando un grafo G=(V,E). Luego:

- Creamos el nodo fuente s en V como representación del segundo plano
- Creamos el nodo sumidero t en V como representación del primer plano
- Creamos un nodo p_i por cada píxel i de la imagen
- Un eje e=(s,pi) por cada nodo “píxel” pi con capacidad a_i.
- Un eje e=(pi,t) por cada nodo “píxel” pi con capacidad b_i.
- Un eje e=(pi,pj) por cada píxel vecino con capacidad pij (esto genera un eje ida y otro vuelta entre cada par de píxeles vecinos).



En la imagen se muestra una simplificación de la transformación de una imagen de 36 píxeles. Se muestran todos los ejes que representan las relaciones de vecindad y su penalidad de separación. Solo algunos de los ejes que representan la probabilidad de pertenecer al fondo o al objeto se muestran. Deberían aparecer en todos los nodos que representan pixeles, sin embargo para lograr una imagen que se pueda comprender se excluyen.

La transformación se puede expresar con pseudocódigo de la siguiente manera:

Transformación a instancia de problema de corte mínimo (T1)

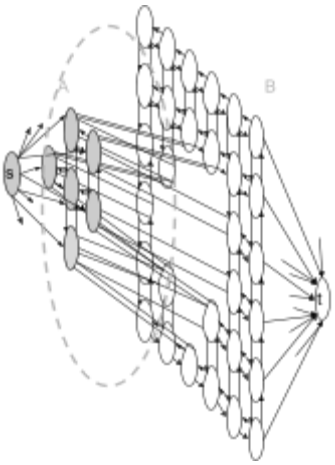
```
Sea P los proyectos
Sea T el tope de ganancia
Construir G=(V,E) red de flujo

Crear s fuente, insertarla en V
Crear t sumidero, insertalo en V

Por cada nodo i
        Crear el nodo pi
        Crear el eje e=(s,pi) con capacidad ai
        Crear el eje e=(pi,t) con capacidad bi

Por cada nodo i
        Por cada nodo j correspondiente a un píxel vecino de i
                Crear el eje e=(pi,pj) con capacidad pij
```

La complejidad de esta transformación es O(n). Se debe considerar que la cantidad de vecinos de un determinado píxel está acotado por el número 4 y por lo tanto se puede considerar una constante en la complejidad.

Una vez obtenida la red de flujo, debemos utilizar como caja negra un algoritmo para resolver el problema de flujo máximo/corte mínimo. Obtendremos como solución una asignación de flujo y el corte mínimo. Deberemos transformar esta, en la solución del problema original.

Utilizando DFS o BFS podemos determinar cuales son los nodos accesibles desde la fuente. Estos corresponden al conjunto A o aquellos pixeles que pertenecen al primer plano. El resto de los nodos corresponden a los píxeles del conjunto B o segundo plano. El valor del flujo máximo está relacionada con la calidad de la segmentación. Si llamamos $Q = \sum_{i \in E} a_i + b_i$ la suma de las probabilidades de todos los pixeles. Veremos más adelante que la calidad de la separación corresponde a la diferencia entre Q y el valor del flujo máximo.

En la imagen se muestra la red de flujo de uena posible imagen de 36 píxeles en 3 dimensiones. No se muestran todos los ejes salientes de la fuente ni todos los entrantes al sumidero. Tener en cuenta que existen ejes de estas características para todos los nodos interiores. Aquellos nodos accesibles desde la fuente luego del cálculo del flujo máximo corresponde al conjunto de píxeles A que representan el objeto en primer plano y los no accesibles corresponden al fondo. Aquellos ejes en el grafo G que parten de un nodo en A y culminan en B conformarán el corte mínimo. En el grafo residual estarán saturados.

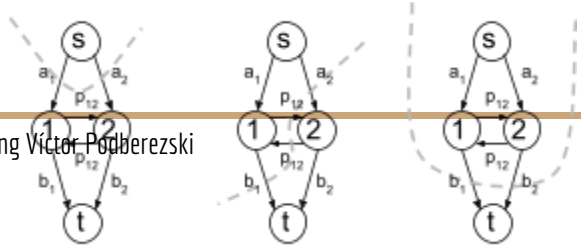


Se puede expresar la transformación utilizando pseudocódigo como:

Transformación de la solución de corte mínimo a solución (T2)
Sea G=(V,E) red de flujo Sea f flujo en la red Sea Gr grafo residual resultante de aplicar f en G Sea Q la suma de todas las probabilidades ai y bi Establecer como A (primer plano) a píxeles que representan los nodos de Gr accesibles desde la fuente (excluyendola) en G Establecer como Calidad de corte q(A,B) a Q menos al flujo máximo de la red. Retornar A y q(A,B)

La complejidad de esta transformación corresponde a la del DFS O(V+E) y la de obtener el flujo máximo sumando el flujo de los ejes que salen de la fuente O(E). Lo expresaremos en función de los parámetros del problema original. La cantidad de nodos en el grafo es O(n) y de igual manera es la cantidad de ejes O(n). Por lo que la complejidad total de la transformación es O(n).

Analizaremos a continuación el motivo por el cual el corte mínimo corresponde a la calidad de la segmentación de la imagen. Iniciaremos con casos simples y luego generalizamos. Supongamos que la segmentación se realiza en una imagen de 2 píxeles. Existen 4 posibles segmentaciones. En la imagen se muestran 3



de ellas. La primera corresponde a que ambos pixeles correspondan a la imagen en primer plano. La última a lo opuesto, que ambas pertenezcan al fondo. La situación intermedia corresponde a que cada pixel corresponde a un conjunto diferente. En esta situación existen dos posibilidades. Ambos revisten un análisis equivalente.

Si la probabilidad de ambos pixeles de pertenecer al fondo bi es mayor a la de pertenecer al primer plano ai, entonces los caminos de aumento saturaran los ejes con capacidad ai. El corte mínimo no dejará los nodos accesibles desde la fuente. Por lo tanto los pixeles pertenecerán al fondo cumpliendo lo esperado.

Similarmente, si la probabilidad de ambos pixeles de pertenecer al fondo bi es menor a la de pertenecer al primer plano ai, entonces los caminos de aumento saturaran los ejes con capacidad bi. El corte mínimo dejará los nodos accesibles desde la fuente y los píxeles pertenecerán al primer plano.

Finalmente, contemplaremos si la probabilidad de un píxel de pertenecer al fondo es mayor que la de pertenecer al primer plano y ocurre lo opuesto. En este caso los ejes de vecindad juegan un papel fundamental. Si la penalidad de separación es un valor suficientemente pequeño los pixeles quedarán en conjuntos diferentes conjuntos. Si corresponde a un valor suficientemente grande entonces permitirá derivar flujo antes de saturarse y puede ocasionar que ambos pasen a estar juntos en el segundo plano.

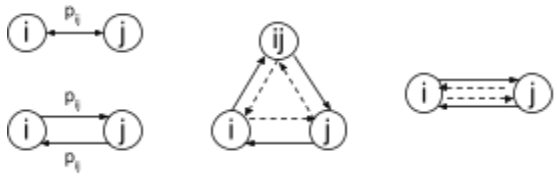
Generalizando. Podemos ver que un corte q(A,B) construido mediante el cálculo del flujo máximo tendremos 3 tipos de ejes que cruzan de A a B en el corte mínimo:

- Un conjunto de ejes e=(s,pj) con capacidad a_j de aquellos nodos j ∈ B
- Un conjunto de ejes e=(pi,t) con capacidad b_i de aquellos nodos i ∈ A
- Un conjunto de ejes e=(pi,pj) con capacidad pij y con i ∈ A y j ∈ B (ejes de vecindad)

Por cada par de ejes vecinos en diferente conjunto sólo un sentido de los ejes de vecindad atraviesan el corte. Podemos entonces expresar la capacidad del corte como $c(A,B) = \sum_{i \in A} b_i + \sum_{i \in B} a_i + \sum_{i \in A, j \in B} p_{ij}$. Recordamos la ecuación de calidad que queríamos maximizar $q(A,B) = \sum_{i \in A} a_i + \sum_{i \in B} b_i - \sum_{i \in A, j \in B} p_{ij}$. Llamamos $Q = \sum_{i \in E} a_i + b_i$ la suma de las

probabilidades de todos los pixeles. Podemos ver que $\sum_{i \in A} a_i + \sum_{i \in B} b_i = Q - \sum_{i \in A} b_i - \sum_{i \in B} a_i$. Reemplazamos las primeras dos sumatorias por la equivalencia recién encontrada y tendremos $q(A,B) = Q - \sum_{i \in A} a_i - \sum_{i \in B} b_i - \sum_{i \in A, j \in B} p_{ij}$. La podemos acomodar como $q(A,B) = Q - \left(\sum_{i \in A} b_i + \sum_{i \in B} a_i + \sum_{i \in A, j \in B} p_{ij} \right)$. Como Q es un valor constante, para maximizar q(A,B) tenemos que minimizar la expresión entre paréntesis. Observando detenidamente esta última expresión corresponde al valor del corte mínimo. Por lo tanto al encontrar el corte mínimo podemos obtener $q(A,B) = Q - c(A,B)$ qué es lo que nos había quedado pendiente demostrar.

Antes de finalizar es importante aclarar una cuestión acerca de los ejes antiparalelos que une cada uno de los nodos vecinos en el grafo construido. Al desarrollar Ford Fulkerson se trabajó con grafos dirigidos donde existía un único eje posible entre un par de nodos. En este caso vemos que tenemos dos en sentido opuesto. Realmente no hay ningún impedimento en que esto ocurra. Lo que se debe es tener cuidado al generar el grafo residual para la construcción de los ejes hacia adelante y hacia atrás. Simplemente implementado con cuidado que par de ejes se pertenecen mutuamente, se resuelve el problema. De todas formas esto impactará en la forma de implementar el almacenamiento del grafo residual que no podrá ser realizado con una matriz. Lo mismo si se usa una lista de adyacencias los pares deben estar correctamente marcados. Otra alternativa[3] trabaja con únicamente los pares de ejes sin agregar ejes adicionales. Uno será el eje hacia atrás del otro y viceversa. El flujo total y la dirección del mismo se calculará como la mitad de la diferencia de sus capacidades en el grafo residual. Por último una tercera alternativa evita los ejes antiparalelos creando un nodo ficticio y redirigiendo uno de los ejes para pasar por el nodo recién agregado. Estas modificaciones permiten también calcular el corte mínimo de un grafo no dirigido.



[3] Computing the Minimum Cut and Maximum Flow of Undirected Graphs, Schroeder, Jonatan & Guedes, André & Duarte Jr, Elias. 2004