Problema de Máximo Flujo

Javier Navarro
Espíndola
Escuela Nacional de
Estudios Superiores Unidad
Morelia
javojavojavojavo@gmail.com

Figura 1: Locomotora eléctrica VL80T-831 con un tren en las inmediaciones del andén de parada Wet Chaltyr por Vadim Anokhin. Licencia: CC BY-SA 3.0

Resumen

Presentación del problema de máximo flujo y diferentes opciones algorítmicas para su solución. Se presenta el procedimiento para solucionar un ejemplo particular usando el método SIMPLEX de programación lineal y otro procedimiento usando el método de Ford-Fulkerson.

Keywords

SIMPLEX, Ford-Fulkerson, Programación Lineal

Programación Lineal, 12/22, México

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Javier Navarro Espíndola. 2022. Problema de Máximo Flujo. In *Proceedings of Programación Lineal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1. Introducción

Considérese un grafo dirigido pesado con un vértice origen y otro vértice destino (fig. 2). Existien varios caminos que inician en el vértice origen y llegan al vértice destino. El peso asignado a las aristas es la capacidad máxima de flujo que pueden llevar.

En el problema de máximo flujo, se busca maximizar el flujo del vértice origen al vértice destino. [noa 2017]

Podemos modelar, por ejemplo, una red ferroviaria teniendo una estación origen y una destino y un conjunto de estaciones intermedias por las cuales se podría pasar para llegar al destino. Las aristas siendo equivalentes a las vías del ferrocarril que conectan a una

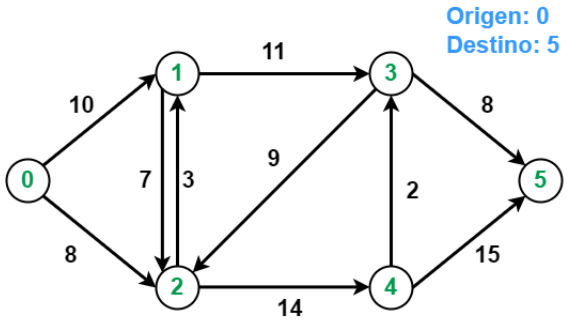

Figura 2: Ejemplo de problema de máximo flujo cuya solución manda el máximo flujo por las aristas del origen al destino sin exceder la capacidad de cada arista.

estación con otra y el peso de dichas aristas, es decir su capacidad, siendo equivalentes a la cantidad de personas o/y recursos que pueden llegar de una estación a otra en cierta unidad de tiempo. Dicha cantidad podría variar porque en ciertas rutas los trenes van más rápido o son más grandes, por ejemplo.

El problema de máximo flujo surgió a partir de la segunda guerra mundial. El documento [T. E. Harris 1995] que plantea el problema por primera vez que propone la solución fue desclasificado en 1999. Estados Unidos quería saber el lugar preciso para echar bombas en la red ferroviaria de la Unión Soviética para inhabilitarla con los menos bombardeos posibles. Para eso se necesitaba saber la capacidad de la red ferroviaria. La red ferroviaria era muy importante estratégicamente ya que transportaba personas y recursos dentro de la unión soviética.[Schrijver 2002]

2. Metodologías para resolver el problema

A continuación se presentan distintas maneras de abordar el problema de máximo flujo.

2.1. Inundando

En este algoritmo[noa 2017] podemos imaginar que estamos lidiando con un sistema de tuberías y queremos liberar la mayor cantidad de agua posible.

El algoritmo es voráz y sencillo. Se identifica el camino que permita llevar más flujo del origen al destino. Cada camino tiene como flujo el mínimo de flujo que tengan sus aristas. Es decir, si se tiene un camino donde todos los artistas tienen una capacidad de 8 pero al final el último arista tiene una capacidad de 2, el flujo que se puede mandar por ese camino es la mínima capacidad de las aristas, es decir 2. Ya que si se manda más al final sólo llegaría 2 porque el último arista funge como cuello de botella. Una vez elegido se 'llenan' las aristas que forman parte del camino, es decir se le resta la capacidad del camino a todas las aristas que forman parte de él. Posteriormente se vuelve a hacer una búsqueda de caminos con el grafo actualizado y se repite el proceso hasta que ya no haya ningún camino que pueda llevar flujo del origen al destino.

La solución es la cantidad de flujo que se mandó por cada arista (grafo actualizado) y la cantidad de flujo que llegó al destino.

Utilizar este algoritmo no garantiza encontrar la solución óptima, como es el caso del ejemplo de la figura 3.

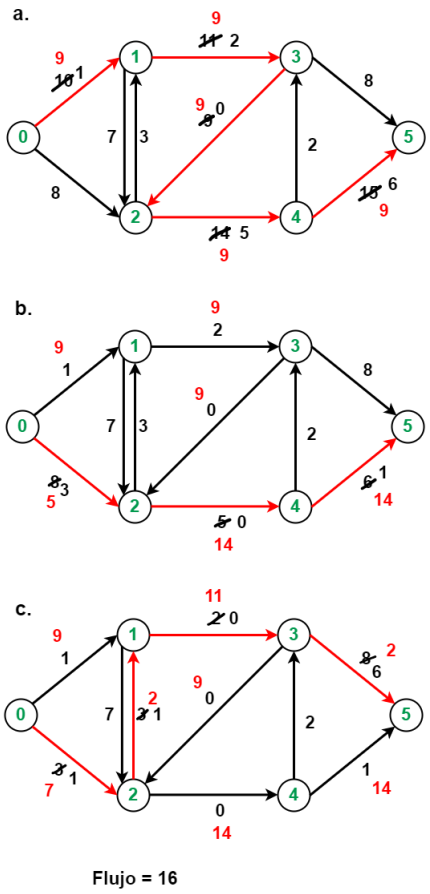

Figura 3: *Inundando* llega a una solución en 3 iteraciones, pero no es óptima. Nótese que al momento de elegir el camino (en rojo) se elige el de máxima capacidad cada vez por ser un algoritmo voraz.

2.2. Grafo residual (Ford-Fulkerson)

Este algoritmo es una extensión del anterior (*Inundando*).[noa 2017] [noa 2013] Se hace lo mismo pero ahora añadiendo un arista en sentido contrario al arista existente entre los vértices que solamente tienen una arista que los conecta. Todos los nodos que estaban conectados originalmente en un solo sentido ahora lo deben estar en ambos. Es importante destacar que la capacidad de las aristas en reversa se inicializa en 0 y cada que se resta capacidad a una arista, por que se mandó flujo, se le suma a su reversa. Posteriormente se identifica un camino, se toma y se hacen las restas y sumas a los aristas. Se repite el proceso hasta que no haya más caminos posibles.

Añadir aristas en reversa cumple la finalidad de 'deshacer'. Es decir si primero se mandó flujo por un camino pero resulta que convenía más mandarlo por otro, el flujo bien puede regresar (más

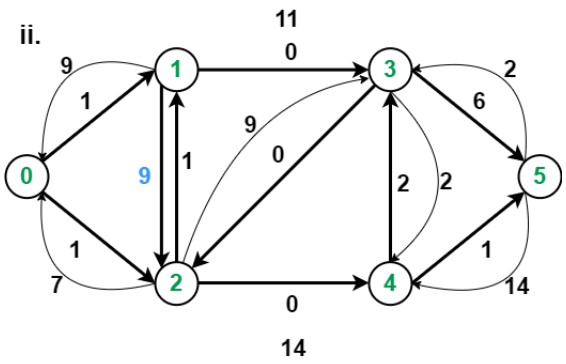

Figura 4: Planteamiento con un grafo residual a partir del estado obtenido al final de la figura 3. Con el grafo residual es posible continuar y encontrar la solución óptima. Nótese que donde ya existe una arista en reversa (entre lós vértices 1 y 2), no es necesario añadir otro.

bien no mandarse en primer lugar) desde donde se quedó para corregir su camino.

Si se retoma el ejemplo anterior solucionado con *inundando*, pero ahora se plantea con un grafo residual, podemos llegar a la solución óptima garantizada (fig. 4 y 5).

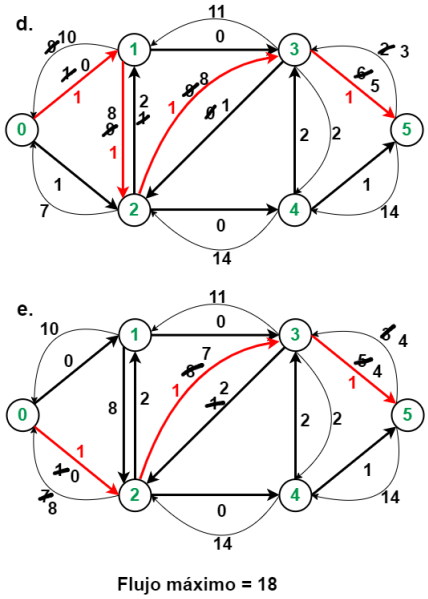

Figura 5: Con 2 iteraciones más utilizando el grafo residual se encuentra el máximo flujo.

El algoritmo de Ford-Fulkerson (al igual que el algoritmo *Inundando*) tiene una complejidad de $O(Ef)$ ya que encontrar un camino toma máximo el número de aristas que existen y como cada camino debe cargar al menos una unidad de flujo, se debe terminar máximo en f iteraciones.

2.3. Problema de programación lineal (SIMPLEX)

Es posible también plantear el problema de máximo flujo como uno de programación lineal.[Matousek and Gärtner 2007] [Adams 2020] Para ello es neceseario definir las variables de decisión, una función objetivo a minimizar y restricciones que acoten a las soluciones. Tomamos el mismo ejemplo (fig. 2).

2.3.1. *Notación*

$$C_{ij} = \textit{ capacidad de arista de } i \textit{ a } j.$$

Por ejemplo en la figura 2:

$$C_{13} = 11$$
$$C_{31} = 0$$

$$X_{ij} = \textit{ flujo de arista de } i \textit{ a } j$$

Por ejemplo en c. de la figura 3:

$$X_{35} = 2$$
$$X_{53} = 0$$

$$E \textit{ siendo el set de aristas}$$

$$n = \#\textit{ vertices}$$

2.3.2. *Problema planteado*

La siguiente expresión limita a cada nodo a recibir el mismo flujo del que dan, y a su vez al origen a dar el mismo flujo que recibe el destino. La primera suma suma todo el flujo que sale del vértice i y la segunda suma suma todo el flujo que llega al vértice i. Que ambas sumas den lo mismo significa que el flujo que entra de un nodo es el mismo que sale. El orígen y destino son excepciones, por eso tienen sus casos particulares.

$$\sum_{(i,j)\in E} X_{i,j} - \sum_{(k,i)\in E} X_{k,i} \leq \begin{cases} f \textit{ si } i = 1, \\ -f \textit{ si } i = n, \\ 0 \textit{ si } i = 2, 3, ..., n-1 \end{cases}$$

Naturalmente $C_{ij} \geq X_{ij}$ y $X_{ij} \geq 0$.

2.3.3. *Forma matricial*

$$Min\ c^T x \textit{ sujeta a } Ax \leq b, x \geq 0$$
$$\textit{donde :}$$

$$x^T = [\, f\ x01\ x02\ x12\ x13\ x21\ x24\ x32\ x35\ x43\ x45\,]$$

El vector x corresponde a las variables de decisión, y definen el flujo que pasa por cada arista. f es el flujo total, equivalente a un arista imaginaria que va del nodo destino al nodo origen.

$$c^T = [\, 0\ -1\ -1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\,]$$

El vector c corresponde al flujo en los aristas que se busca maximizar. Los aristas que cuyo valor está en -1 *se minimizarán. Están en* -1 *y no* 1 *ya que es necesario convertir el problema de maximización (flujo máximo) en un de minimización. En este caso se optó por elegir la arista* 01 *y la* 02*, pero bien pudo haberse elegido la arista imaginaria* f *la cual garantiza un corte completo en cualquier grafo.*

$$A = \left[\begin{array}{c|cccccccccc} -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Existen 3 valores diferentes en A: 0 *indica que no hay flujo,* 1 *indica que el vértice i manda flujo por el arista j y* -1 *indica que el vértice i recibe flujo por el arista j. Las restricciones en A se dividen en partes. Las primeras 6 filas son las restricciones que garantizan que el flujo que entra a un nodo es equivalente al que sale.* f *(el arista imaginario) sirve para que la restricción se cumpla también en el vértice origen y destino de los cuales solo sale y entra flujo respectivamente, es por eso que solo tiene valor en el primer y último renglón. Cada fila representa un nodo y cada columna representa un arísta (la primera columna representa al arista* f *imaginario). Nótese que en cada columna, tomando en cuenta solo las primeras 6 filas, existe un* 1 *y un* -1 *porque cada arista recibe y manda flujo una sola vez. A partir de la séptima fila se restringe el flujo de cada arista para que sea menor o igual a su capacidad.*

$$b^T = [\,0\ 0\ 0\ 0\ 0\ 0\ 10\ 8\ 7\ 11\ 3\ 14\ 9\ 8\ 2\ 15\,]$$

Los primeros 6 elementos del vector b coorresponden a las restricciones que aseguran que el flujo que entra y sale de cada nodo es igual. Los elementos restantes indican la capacidad de cada arista.

Para ejemplificar, la segunda restricción equivale a :

$$-x_{01} + x_{12} + x_{13} - x_{21} \leq 0$$

Donde el flujo que llega al vértice 1 *debe ser igual al que sale de él.*

La última restircción es:

$$x_{45} \leq 15$$

Y evita que el flujo de x_{45} *exeda su capacidad*

3. Implementación y ejemplos

Se eligieron 2 métodos para su implementación y solución del ejemplo presentado en la figura 2.

- Grafo residual (Ford-Fulkerson)
- Problema de Programación Lineal

Ambos métodos garantizan una solución óptima. El código con la implementación de ambos métodos y el cálculo de soluciones se encuentra en el siguiente repositorio: github.com/JavoJavo/max-flow-ford-fulkerson-SIMPLEX.

3.1. Grafo residual (Ford-Fulkerson)

El pseudocódigo se presenta en el algoritmo 1.

Se hace una representación del grafo para poder procesarlo.

Es necesario hacer una copia del grafo original ya que el algoritmo modifica al grafo y posteriormente se busca comparar ambos grafos para calcular el flujo que pasó por cada arista.

La búsqueda en anchura encontrará un camino posible desde el origen al destino. Se considera un camino posible a un camino donde todos los aristas que lo componen tienen al menos 1 de capacidad.

Una vez elegido un camino se actualizan las capacidades de los aristas en el grafo restando el flujo del camino a cada arista del camino.

El flujo del camino es la capacidad mínima del conjunto de capacidades de las aristas que conforman al camino.

Cuando no hay más caminos posibles, el algoritmo calcula los flujos y ha terminado.

Algoritmo 1 Ford-Fulkerson

```
Grafo_original ← copy(Grafo)
camino_posible ← busq_anchura(Origen, Destino, Grafo)
while camino_posible ≠ False do
    camino_posible ← busq_anchura(Origen, Destino, Grafo)
    actualizar_grafo(Grafo, camino_posible)
end while
flujo ← contar_flujo(Grafo, Grafo_original, Origen)
```

3.2. Problema de programación lineal (SIMPLEX)

El pseudocódigo se presenta en el algoritmo 2.

La matriz que recibe A es el problema de programación lineal en su forma canónica.

La condición de paro se alcanza cuando ya no hay números negativos en la última fila.

Los pivotes se eligen y se guardan los índices para al final indicar cuáles son las variables básicas.

Las operaciones elementales vuelven 1 el pivote y 0 los elementos de su columna, convirtiéndolo en variable básica.

Al alcanzar la condición de paro se regresa la solución.

Algoritmo 2 SIMPLEX

```
A ← matriz
while condicion_paro(A) = False do
    pivote_i, pivote_j ← sacar_pivote(A)
    operaciones_elementales(pivote_i, pivote_j, A)
end while
valores_de_variables ← A[−1, : −1]
f ← A[−1, −1]
```

4. Conclusiones

El algoritmo de Ford-Fulkerson puede seguirse fácilmente y termina en relativamente pocas iteraciones. Se garantiza que todos los

flujos de las aristas en la solución son enteros y tiene una complejidad razonable.

Usar el método SIMPLEX para resolver el problema de máximo flujo presenta una solución ingeniosa. Sin embargo, una desventaja es que la solución no necesariamente será entera por lo cual la interpretación es un poco menos sencilla. Aunque el orden del método SIMPLEX sea exponencial, en la práctica es probable que no haya grandes diferencias en el tiempo de ejecución comparándolo con el algoritmo de Ford-Fulkerson.

5. Problema dual

En el problema dual se minimizan las capacidades de cada arista. Las variables de decisión son:

- La "capacidad" de cada nodo.
- La capacidad de cada arista.

Referencias

2013. Ford-Fulkerson algorithm for maximum flow problem. https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/. Accessed: 2022-11-29.

2017. Max flow problem introduction. https://www.geeksforgeeks.org/max-flow-problem-introduction/. Accessed: 2022-11-29.

Henry Adams. 2020. Linear Programming 44: Maximum flow.

Jiri Matousek and Bernd Gärtner. 2007. *Understanding and using linear programming* (2007 ed.). Springer, Berlin, Germany.

Alexander Schrijver. 2002. On the history of the transportation and maximum flow problems. *Math. Program.* 91, 3 (2002), 437–445.

F S Ross T. E. Harris. 1995. *FUNDAMENTALS OF A METHOD FOR EVALUATING RAIL NET CAPACITIES.* Technical Report.