Facultad
de
Ciencias

GESTIÓN DE RECURSOS
Resource Managemet

Trabajo de Fin de Grado
para acceder al

GRADO EN INFORMÁTICA

Autor: Luz María Álvarez Moreno

Director: Domingo Gómez Pérez

Febrero - 2016

Agradecimientos

A mis padres, Pedro y Rosi, porque desde el día que decidí comenzar la carrera de Ingeniería Informática han estado a mi lado, apoyándome y enseñando a que todo es posible en esta vida si uno no se rinde y tiene claro cuáles son sus sueños. Desde que tengo uso de razón me han enseñado que en esta vida a base de tesón y esfuerzo llegan las recompensas y aquí finalizan muchos años de estudios, de alegrías y de llantos pero sobre de apoyo y unidad familiar. No me olvido de mi hermano Pedro, que desde siempre ha estado dispuesto a ayudarme y aunque sea el pequeño de los dos, para mi es muy grande.

No me olvido de mi abuelo Ángel, que aunque hoy ya no está conmigo siempre confió en mi y en mis posibilidades y se que desde el cielo siempre me ha mandado las fuerzas que he necesitado para cumplir nuestra promesa. Estés donde estés, va por ti 'abuelito', porque siempre serás mi ángel.

A mi profesor Domingo y a Anabel por su ayuda incondicional y apoyo durante el desarrollo de este proyecto.

Resumen

Este trabajo presenta una introducción al problema del flujo máximo en grafos, que aparece de forma recurrente en diversas áreas y del que se han propuesto diversas soluciones pero que sigue atrayendo interés en el mundo de la investigación.
Las aplicaciones más usuales que nos encontramos son en logística (transporte de mercancías), en el flujo de gases y líquidos por redes de tuberías interconectadas, en el flujo de corrientes de intensidad en redes eléctricas con distintas capacidades y en el tráfico ferroviario, por citar algunas. Un problema típico sería hallar cual es el mayor flujo de corriente que se puede transmitir por una red eléctrica.
Haciendo uso de la teoría de grafos, la red se representa mediante un grafo dirigido, compuesto por una serie de aristas con capacidades asociadas. El modelo de grafo asocia a un nodo inicial el papel de fuente y a un nodo final o destino el papel de sumidero. Asignando flujos a cada arista, el objetivo es repartir el flujo por los arcos de la red de forma óptima desde el nodo fuente hasta el nodo sumidero.
En este proyecto nos centraremos en varios algoritmos de flujo máximo y su implementación. El trabajo se divide en una introducción general a los problemas que pueden ser resueltos utilizando estos algoritmos y una segunda parte que se centra en el algoritmo Ford Fulkenson y su rendimiento para grafos acíclicos, así como una revisión de otros algoritmos propuestos.

Palabras clave: flujo máximo, algoritmo Ford Fulkerson, algoritmo Edmonds Karp, algoritmo Push Relabel, teoría de grafos.

Abstract

This work presents an introduction to the maximum flow problem. This problem frequently appears in many different areas and, although many solutions have been proposed, it still attracts interest of many researchers.

The most usual applications are found in logistics (goods transportation), liquid or gas flow through interconnected pipes, electrical current through a network with different capacities and railway traffic, just to name a few. A typical question would be to calculate the maximum flow of electrical current that can be transmitted over a given electrical network.
Using graph theory, the network is represented by a directed graph, with capacities given to itsl edges. The associated graph model has two distringuished nodes: a source node and a sink node. The solution to the problem is to split the flow optimally between all the edges from the source to the sink. In this project, we focus in several maximum flow algorithms and their implementation. The work can be divided in a first part, that introduces examples and problems that can be addressed with these techniques, and a second part that focus on the Ford Fulkenson algorithm and its performance in acyclic graphs, as well as a brief revision of other proposed algorithms.

Keywords: maximum flow, Ford Fulkerson algorithm, Edmonds Karp algorithm, Push Relabel algorithm, graph theory

Índice general

Capítulo 1

Motivación, Metodología e Introducción

1.1. Motivación, Alcance y objetivo

En la sociedad actual se encuentran múltiples ejemplos en los que se puede comprobar que está regida por redes, redes de comunicación, redes de telefonía, redes de transporte, redes de energía eléctrica, etc.
La prestación de servicios a través de estas redes requiere grandes inversiones en recursos. Normalmente la asignación de estos recursos se hace mediante un modelo matemático de forma que se permita economizar al máximo ofreciendo una calidad suficiente de servicio al usuario final.
El problema de modelar un determinado servicio y asignar los mínimos recursos posibles manteniendo un nivel de calidad tiene muchas aplicaciones en diferentes áreas. Las redes de flujo son una herramienta adecuada para abordar este tipo de problemas.
Este proyecto tiene por objeto investigar en la literatura científica para desarrollar una implementación de una serie de algoritmos que permitan, mediante la parametrización de un problema real, dar una solución aproximada de forma eficiente. En las siguientes secciones se explicarán detalladamente las aplicaciones de los métodos de flujo máximo, así como los diferentes algoritmos desarrollados a lo largo de la historia con su implementación y posterior análisis.

1.2. Introducción

Las redes de flujo normalmente son usadas para modelar líquidos que fluyen a través de tubos o la corriente eléctrica que se transmite a través de redes eléctricas, en definitiva, simulan transferencias entre nodos de una red. Por esto, las aristas dirigidas de la red son las encargadas de representar un intercambio de materiales o energía. Cada arista tiene asignada una capacidad, que es representa la máxima tasa de transmisión sobre esa artista. Los vértices representan puntos de interconexión entre el nodo fuente y el nodo destino, cumpliéndose que la tasa de entrada y la tasa de salida son idénticas, no produciéndose pérdidas. Un ejemplo de esta propiedad se encuentra en la ley de nodos o la primera ley de Kirchhoff.
El problema del flujo máximo trata de encontrar la cantidad máxima de una variable objetivo que se puede transmitir entre dos vértices fuente y destino de una red. [9] [10]. El estudio de este tipo de problemas comienza con trabajos en programación lineal, estimulados por el trabajo de Ford y Fulkerson [6]. Posteriormente se han propuesto una serie de mejoras que han mejorado la eficiencia de los algoritmos de redes de flujo, siendo un tema en el que todavía se presentan nuevas mejoras y algoritmos cada vez más eficientes a pesar de que han transcurrido más de 60 años desde que se comenzaron a investigar este tipo de problemas.
Este proyecto se centra analizar la eficiencia media del algoritmo Ford-Fulkerson y estudiar otros algoritmos propuestos para resolver el problema del flujo máximo, abordando de forma secundaria la red de flujo clásica y el problema del mínimo coste de circulación. Por lo tanto, se presentan varios de los diferentes algoritmos que se han implementado a lo largo de la historia, así como los resultados obtenidos, que se explicarán a lo largo de las posteriores secciones.
Los problemas de flujo de red analizados también se pueden modelar como problemas de programación lineal, esto es orientados a resolver un sistema de inecuaciones lineales, y por lo tanto pueden ser resueltos por algoritmos propios de este área. Sin embargo, la estructura combinatoria de estos problemas hace que sea posible la obtención de los algoritmos más eficientes.
El primer algoritmo basado en programación lineal entera que podía resolver problemas de flujo de red fue el método Simplex para redes de Dantzig [4]. Se trata de una especialización del método Simplex empleando programaciones lineales y diseñado para tomar ventaja de la estructura combinatoria de problema. Las variantes del método Simplex tratan de evitar los ciclos y

reducir la complejidad de los problemas de flujo de red.
Cunningham [3] propone una estrategia anti ciclos para el método Simplex de red basada en propiedades teóricas del grafo en el problema de mínimo coste de circulación. Recientemente, Goldfarband Hao [8] diseñó una variante del método Simplex para el problema del flujo máximo que se ejecuta en tiempo polinomial.
Orlin [12] diseñó una variante del método de Doble Simplex aplicado a redes para hallar el mínimo coste en ejecución polinomial.
Aunque durante mucho tiempo, el método Simplex para redes ha sido el método más utilizado en la práctica, especialmente en el problema de mínimo coste de circulación, posteriormente se han propuesto algoritmos que son bastante mejores con respecto al tiempo de ejecución. El primer algoritmo pseudo polinomial diseñado para resolver el problema del flujo máximo es el algoritmo de Ford y Fulkerson. Tanto Dinic como Edmonds y Karp [5] son mejoras posteriores a este algoritmo.
Por último, el método Push Relabel propuesto por Goldberg y Tarjan [7] (junto a sus posteriores variantes) consigue resultados más eficientes mediante varias operaciones de empuje y reetiquetado de flujo.

1.3. Aplicaciones del Flujo Máximo

El problema del flujo máximo tiene numerosas aplicaciones en el mundo real. Los algoritmos de flujo máximo se aplican a la vida cotidiana para resolver problemas de gestión de recursos, reparto en empresas de logística, control de vuelos con escalas en aerolíneas, gestión de selección de proyectos o para calcular las intensidades máximas en un circuito eléctrico, entre otros. En las siguientes subsecciones se describen algunas de ellas.

1.3.1. Aerolíneas

Aplicando métodos de flujo máximo se pretende determinar la máxima cantidad de conexiones de vuelos que se pueden alcanzar conectando dos ciudades entre las que los aviones deben realizar escala en diferentes aeropuertos antes de llegar a su destino. Los aeropuertos intermedios tienen capacidades limitadas, por lo que si están ocupados por otros aviones que realicen otras trayectorias no admiten nuevos aviones que quieran hacer una escala.
El modelo de problema de flujo máximo implica que en este caso las ciudades

Figura 1.1: Vuelos desde Madrid a Los Ángeles con sus escalas.

son los nodos, siendo la cuidad de origen el nodo origen o fuente y la cuidad destino el nodo destino o sumidero. Las rutas que interconectan las ciudades son los arcos y la cantidad de aviones que pueden aterrizar en el aeropuerto en ese momento hace referencia a la capacidad.

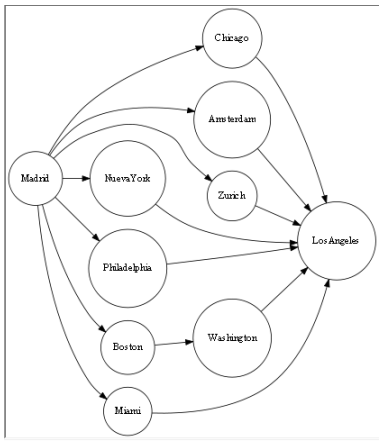

Figura 1.2: Grafo que representa los vuelos con sus escala.

En la figura 1.1 se puede observar los distintos vuelos desde el aeropuerto de Madrid, tomado como fuente, al aeropuerto de Los Ángeles que es el destino con sus diferentes escalas i. Aquí hemos reducido el problema tomando

Jardín	Inicio	Fin	Tiempo estimado
Jardín 1	1 Enero	15 Enero	2 semanas
Jardín 2	10 Enero	28 Febrero	4 semanas
Jardín 3	1 Enero	28 Enero	5 semanas
Jardín 4	18 Enero	30 Enero	2 semanas
Jardín 5	1 Enero	25 Febrero	8 semanas

Cuadro 1.1: Planificación temporal de las tareas asignadas a diferentes jardines.

solamente las opciones que ofrecen las compañias aereas sin considerar otros viajes posibles por simplicidad. Como se puede observar, las escalas se representan mediante el grafo en la figura 1.2. Las capacidades aparecen como etiquetas de las aristas y según las capacidades que tengan asignadas en sus aristas queremos obtener la máxima cantidad de conexiones de vuelos.

1.3.2. Selección de proyectos

Teniendo en cuenta una cantidad de proyectos p_i y una cantidad de equipos q_i. Cada proyecto tienen un coste cq_i y un beneficio bp_i. Cada proyecto está formado por un número de equipos distribuidos adecuadamente entre los diferentes proyectos para así maximizar las ganancias.
Se quiere resolver como asignar proyectos a diferentes equipos teniendo en cuenta el coste en tiempo que va a tener la realización de cada proyecto y así ver si es factible realizar todos los proyectos en un tiempo establecido. Se propone el siguiente ejemplo dado en el cuadro 1.1: Una empresa de jardinería dedicada al mantenimiento de varios jardines consta de cuatro equipos de trabajadores. Durante el mes de Enero han sido contratados para realizar el mantenimiento de cinco jardines en diferentes comunidades de vecinos. Los trabajos en cada jardín pueden ser iniciados en una determinada fecha pero deben ser acabado en un periodo establecido una vez se ha firmado el contrato.

Cada equipo sólo puede estar trabajando en un jardín, por lo tanto dos equipos no están a la par en el mismo jardín. Para ver si se pueden llevar a cabo las tareas a tiempo, se representan en la red en la figura 1.3.

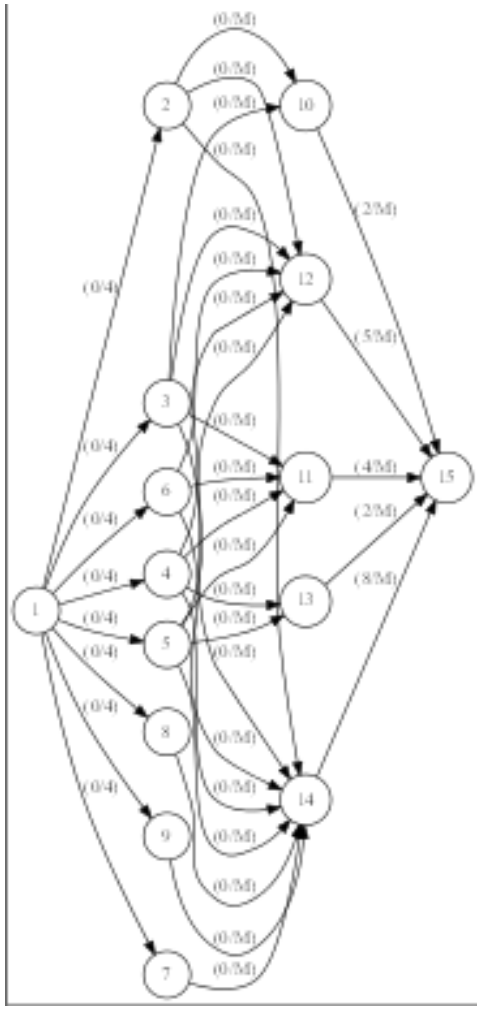

Figura 1.3: Red de trabajos

El nodo 1 es el inicio, es decir la fuente y el nodo 15, es el final o destino. Del 1 de enero al 25 de febrero pasan 8 semanas entre las cuáles se reparten los trabajos en los diferentes jardines de las comunidades, representados por los nodos del 2 al 9.
Los nodos 10 al 14 representan los trabajos en los 5 jardines contratados.
Del nodo 1 a los nodos del 2 al 8 que representan las 8 semanas del periodo contratado tienen capacidad 4, ya que la empresa consta de 4 equipos de trabajo.

El nodo 10 representa el trabajo del jardín 1, que se ha contratado en 2 semanas, nodos 3 y 4, que representan las semanas 1 y 2 de enero. Del mismo modo, se opera para los siguientes trabajos,

- Para el jardín 2, serán 4 semanas que va desde la segunda semana de enero hasta la cuarta.
- Para el jardín 3, se emplean 5 semanas desde el nodo 2 al 6.
- Para el jardín 4, se emplean 2 semanas desde el nodo 3 y 4.
- Para el jardín 5, se emplean 8 semanas desde el nodo 2 al 9.

Por lo tanto, es posible acabar todos los jardines a tiempo en 21 días $(2 + 4 + 5 + 2 + 8 = 21)$.

1.3.3. Logística

Una empresa de logística transporta m productos desde sus almacenes a los diferentes centros de demanda. Las carreteras pueden ser vistas como aristas del grafo y la capacidad representa el número de productos que pueden viajar al día por las carreteras. Las razones de la limitación pueden ser varias: carreteras muy transcurridas, limitación de tránsito de camiones, etc.
Reduciendo el problema a un problema de hallar el flujo máximo en una red de flujo, se determina que las ciudades de reparto son los nodos, el almacén principal es el nodo origen o fuente, los lugares de entrega es el nodo destino o sumidero, las rutas que interconectan las ciudades y los almacenes son los arcos y la cantidad de camiones que pueden hacer paradas en los diferentes almacenes de la compañía es la capacidad.
El objetivo es determinar la máxima cantidad de conexiones entre camiones que se puedan alcanzar conectando las diferentes ciudades y teniendo en cuenta los almacenes de la compañía de tal manera que un camión pueda entrar en uno de ellos y hacer la descarga de los paquetes que otro compañero pueda cargar y entregarlo en una ruta que le coja de camino, ahorrando así rutas entre camiones y aumentando el beneficio de la empresa.

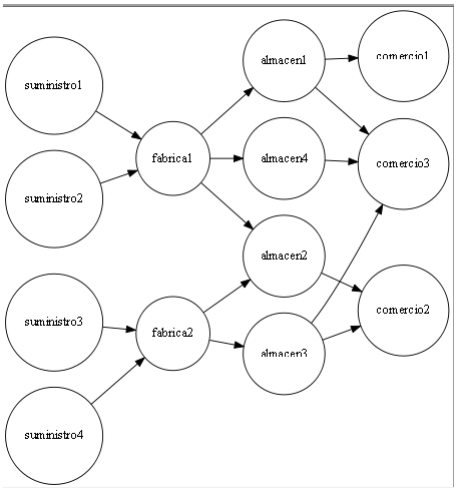

Figura 1.4: Grafo que representa el proceso realizado en una empresa de logística

En el ejemplo que se representa en la figura 1.4 se puede ver una red de suministro de materias primas que se transforman en una fábrica en productos elaborados y que se almacenan temporalmente hasta llegar a los comercios en los que han sido demandados, mediante la red de transportes de una empresa de logística.

1.3.4. Diseño de circuitos eléctricos

Las leyes de Kirchhoff son muy utilizadas en ingeniería eléctrica para obtener los valores de la corriente y voltaje en diferentes puntos de un circuito eléctrico y surgen de la aplicación de la ley de conservación de la energía y la carga. Permiten resolver circuitos planteando un conjunto de ecuaciones. Un circuito eléctrico puede ser modelado como una serie de componentes eléctricos interconexionados a través de líneas de transmisión o cables que se consideran ideales. La representación en forma de grafo implica la generación de una serie de nodos o puntos dentro del circuito donde se unen más de un terminal de un componente eléctrico.
Según el enunciado de la primera ley de Kirchhoff la corriente que entra a un nodo por un cable es igual a la suma de las corrientes que salen de él. La segunda ley de Kirchhoff enuncia que la suma de todas las tensiones genera-

das sobre un lazo cerrado tiene que ser cero.
Dado que la intensidad eléctrica tiene restricciones asociadas a la protección de los componentes y a otras consideraciones de seguridad, se plantea el problema de entregar la intensidad máxima a un determinado nodo del circuito.

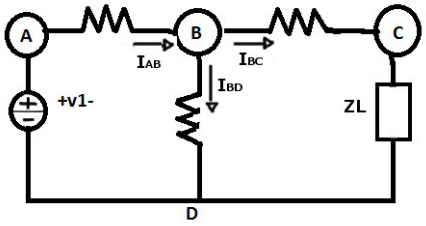

Figura 1.5: Ejemplo de un circuito eléctrico sencillo.

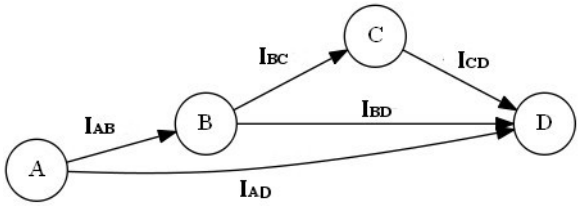

Figura 1.6: Grafo que representa las corrientes de intensidad del circuito de la figura 1.5. Por la primera ley de Kirchhoff, $I_{AD} = -I_{AB}$ y $I_{BC} = I_{CD}$

La figura 1.5 representa un circuito con dos lazos. En el circuito hay cuatro nodos de interés marcados por las letras A, B, C, D. También en el circuito hay una fuente de alimentación, tres resistencias y una carga. La elección del voltaje de la fuente de alimentación depende de las intensidades que soporten los cables y del valor de las resistencias. La representación en forma de red de flujo esta dado en la figura 1.6.

Capítulo 2

Fujo Máximo

2.1. Redes de Flujo

Se conoce como red de flujo a un grafo dirigido $G = (V, E)$ con pesos no negativos. Estos pesos representan el flujo que transcurre por una arista. Asociado también al grafo, existe la capacidad c que asigna a cada arista del grafo la capacidad máxima que tiene esa arista. Esta aplicación normalmente se extiende a todo par de vértices (u, v) dando $c(u, v) = 0$ si $(u, v) \notin E$.

En esta red hay dos nodos con propiedades especiales: la fuente, que la denotaremos por s y el sumidero, denotado por t. Estos nodos tienen una cualidad especial, **es que el flujo entrante es distinto del flujo saliente.** Esto es, la suma de los pesos de las aristas entrantes no es igual a la suma de los pesos en las aristas salientes. En el nodo fuente, la suma de los pesos en las aristas salientes es estrictamente mayor que la suma de los pesos de las aristas entrantes y en el nodo sumidero es al contrario. En el resto de los vértices del grafo se da la igualdad.[1]
Como ya hemos mencionado, los pesos de las aristas en una red de flujo no exceden las capacidades que van asociadas a cada arista.

Veamos paso por paso el modelado una red de flujo en un ejemplo concreto. Supongamos que necesitamos trasladar material, que van desde una fábrica que consideraremos como el nodo fuente, hasta un destino que consideramos el sumidero. Los materiales se producen en unidades enteras, lo que quiere decir que no podemos producir $0,23$ de material ni tampoco transportarlo. La tasa en que la fábrica produce el material la vamos a considerar

[1]Notad el paralelismo con la primera ley de Kirchhoff

estacionaria, pudiendo ser aumentada tanto como queramos. El sumidero es el encargado de consumir el material que le ha llegado por alguno de las aristas del grafo. Por lo tanto, consideramos cada arco del grafo como un conducto con cierta capacidad y, por tanto, se puede considerar como flujo del material en cualquier punto a la tasa a la cuál el material se va moviendo. Para más detalles, ver [2]. La definición formal de redes de flujo es la siguiente:

Definición 1. *Una red de flujo es un grafo dirigido $G = (V, E)$, donde V es el conjunto de vértices o nodos y $E \subset V \times V$ es el conjunto de aristas junto con dos aplicaciones más $f, c : V \times V \mapsto \mathbb{N}$ satisfaciendo las siguientes propiedades: en el cuál cada arco $(u, v) \in E$*

- *Hay dos vértices distinguidos: los vertices s, t, conocidos como fuente y sumidero.*
- *Si $(u, v) \notin E$, entonces $f(u, v) = c(u, v) = 0$.*
- *Cada vértice está en alguna ruta de la fuente al destino.*
- *Para cualquier nodo u que no sea el nodo fuente o sumidero se cumple que:*
$$\sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u).$$
- *Adicionalmente se cumple que:*
$$\sum_{v \in V} f(s, v) \geq \sum_{v \in V} f(v, s), \quad \sum_{v \in V} f(t, v) \leq \sum_{v \in V} f(v, t).$$

Veremos que en problemas de redes de flujo donde hay varios nodos fuentes y varios nodos sumideros, se pueden reducir al caso especial donde hay solamente un nodo fuente y otro nodo sumidero. En la siguiente sección enunciaremos el problema del flujo máximo.

2.2. El problema del Flujo Máximo

El problema del flujo máximo consiste en encontrar la mayor cantidad de un material que se pueda trasladar en una red de flujo desde una fuente s a

un destino t respetando las restricciones de capacidad. Dicho de otra forma, lo que hay que intentar es máximizar el valor de flujo. El valor del flujo se define como $|f| = \sum_{v \in V} f(s, v)$, denotando el valor del flujo total que sale de la fuente.

Veamos un ejemplo, que aclara muchas dudas. Este ejemplo es suficientemente sencillo para realizarlo con lápiz y papel.

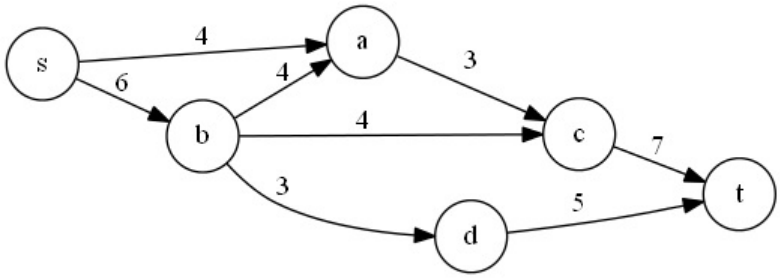

Figura 2.1: Grafo con una red dirigida y capacidades

En la red de flujo representada en la figura 2.1, vemos un grafo dirigido con 6 nodos. Las etiquetas de los nodos son $\{s, a, b, c, d, t\}$ y este orden servirá para representar la matriz de adyacencias y las capacidades. La matriz de adyacencias tiene la siguiente forma:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

donde el orden de los nodos es $\{s, a, b, c, d, t\}$. La aplicación $c : V \times V \mapsto \mathbb{N}$ se puede representar en forma de matriz siguiendo un esquema muy parecido a la matriz de adyacencia. Las filas se suponen etiquetadas con $\{s, a, b, c, d, t\}$ al igual que las columnas. Ahora, si $c(s, a) = 4$ ponemos en el elemento que esta en la fila que esta etiquetada como s y en el columna etiquetada como a 4. Repitiendo este proceso para todos los nodos, obtenemos la siguiente

matriz:

$$\begin{bmatrix} 0 & 4 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 4 & 0 & 4 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Algunas observaciones sobre este caso es que el flujo máximo que puede entrar en el nodo t es 12 y también es fácil ver que el máximo flujo que puede salir del nodo s es 10. Por lo tanto, se deduce que el flujo máximo tiene que ser menor que 10. Es más, un flujo máximo esta dado por la siguiente matriz:

$$\begin{bmatrix} 0 & 3 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.3. Una primera aproximación a la solución

Hay un algoritmo para resolver el problema del flujo máximo, pero no garantiza la solución óptima y además se trata de un algoritmo no determinista. En secciones posteriores estudiaremos otros algoritmos más eficientes y que garantizan la solución óptima. La razón por la que presentamos este algoritmo es para explicar ideas y dificultades que tiene este problema. La idea es tomar mejorar iterativamente un flujo dado. Esto es a partir de un flujo f, encontrar un flujo f' tal que $|f| \leq |f'|$.

Empezaremos eligiendo un camino p de s a t y comprobaremos las capacidades de las aristas en el camino p. Se comprueba si el flujo que define f en todas de las aristas puede ser aumentado, y en ese caso, se aumente al mínimo posible de todas las aristas. La idea es escoger un cámino al azar y comprobar si se puede aumentar el flujo por ese cámino. Un concepto que aparece naturalmente es la red residual G_f de un flujo f y es el grafo definido por las aristas que se puede aumentar el flujo. Estas aristas son aquellas que une dos vertices $u, v \in V$ y $c(u, v) > f(u, v)$.
Notad que se podrán elegir varios caminos y según el camino que se escoja, el algoritmo podrá alcanzar una solución óptima o no. La razón es que, en

algunas ocasiones, tendremos que disminuir el flujo por algún camino para aumentarlo en otro. Esto se representará en la red residual G_f añadiendo aristas entre los nodos v y u si $f(v, u) > 0$.
Para deshacer el flujo que pasa por el camino trabajaremos en el grafo residual G_f y enviaremos flujo por las aristas v, u de formar que $f(u, v) > 0$. Por esto, añadir m unidades de flujo en G_f puede significar disminuir m unidades de flujo en varias aristas de G. Esto será la base del primer algoritmo que alcanza la solución del problema del flujo máximo.

2.4. Redes con múltiples flujos y destinos

El problema de encontrar el flujo máximo se puede plantear en terminos más generales. Podríamos encontrarnos con el problema de que haya varias fuentes $s_1, s_2, \ldots, s_n$ y sumideros $t_1, t_2, \ldots, t_n$, pero será reducido a un problema original de flujo máximo que tendrá una súper-fuente y un súper-destino. Para solucionar el problema de las múltiples fuentes y destinos, se agrega un vértice fuente que conecte a las fuentes anteriores mediante aristas de capacidad infinita o muy grande y otro vértice destino del mismo modo. Una vez hecho esto, el problema se reduce a un problema de flujo máximo conocido. En la imagen 2.2 se puede ver un ejemplo de este método.

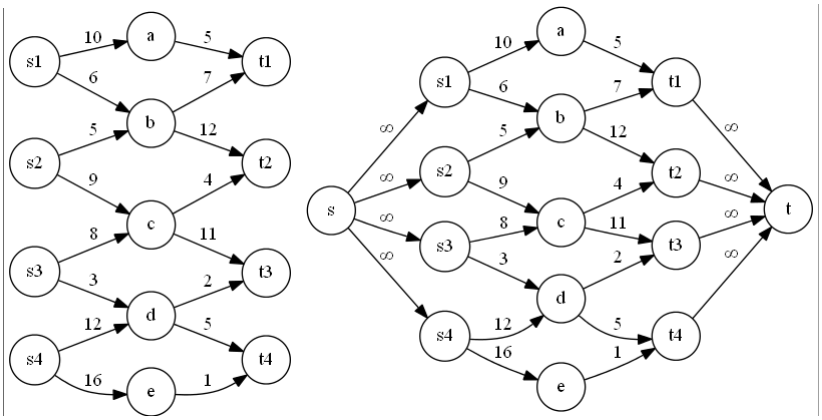

Figura 2.2: Problema de múltiples fuentes y sumideros y su reducción

Capítulo 3

Trabajo con Flujos

3.1. Conceptos básicos en el algoritmo Ford-Fulkerson

El algoritmo de Ford-Fulkerson, data de 1956 y se trata del primer algoritmo propuesto para resolver el problema de flujo máximo, basado en el concepto de grafo residual. Fue en 1962 cuando Ford y Fulkerson establecieron el teorema de flujo máximo-corte mínimo y resolvieron el problema de flujo máximo a través de algoritmos de caminos incrementales.
Se trata de un método genérico que aumenta la capacidad de los flujos incrementalmente a lo largo de los caminos que van del origen al destino y que sirve como la base de una serie de algoritmos que se explicarán más adelante.
El objetivo del algoritmo de Ford-Fulkerson es buscar caminos en los que se pueda aumentar el flujo, hasta alcanzar el flujo máximo, encontrando una ruta de penetración con un flujo positivo neto que una los nodos origen y destino.
Este algoritmo es un método que depende de tres ideas principales: red residual, aumento de camino y cortes.

- Comienza con $f(u, v) = 0$ para todo $u, v \in V$ para cada par de nodos y con un flujo inicial igual a cero.

- En cada iteración se incrementa el valor del flujo con el objetivo de encontrar un camino incremental, que va desde la fuente s, al destino t y que puede conducir mas de un flujo por el.

- El proceso se repetirá tantas veces como sea necesario hasta que no se encuentre ningún camino de aumento.

Un camino de flujo residual es el camino en el cual de la fuente al sumidero todas las aristas del camino tienen un flujo residual mayor que cero. En una red residual los arcos admiten más flujo, aparecen los caminos de flujo residual, por lo tanto es un conjunto de aristas que pueden admitir más flujo. Dada una red de flujo $G = (V, E)$ con fuente s y destino t y un flujo f, la red residual inducida por f es la siguiente,
$G_f = (V, E_f)$ con $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$.
Es aquí cuando por el teorema de max-flow min-cut muestra al final del algoritmo que el proceso lleva el flujo máximo.
El flujo residual es el flujo que queda disponible en una arista después de que se haya enviado flujo por ella. El flujo neto debe superar a la capacidad de la arista ni ser menor que cero.
El flujo residual se define como la capacidad menos el flujo actual. Para cada par de nodos, $u, v \in V$, la cantidad de flujo adicional que se podrá verter sobre u, v es la capacidad residual, en otras palabras, $c_f(u, v) = c(u, v) - f(u, v)$.

Un camino incremental p es un camino sin ciclos desde la fuente s al sumidero o destino t en el grafo residual G_f.
Cada arista o arco (u, v) incremental del camino tiene que tener flujo positivo y además siempre respete la restricción de capacidad de la arista.
En este caso, la capacidad residual es la máxima cantidad de flujo neto que se puede enviar por las aristas del camino incremental. Por lo tanto el flujo adicional máximo es, $C_f(p) = \text{mín}(c_f(u, v)/(u, v) \in p)$
En definitiva, lo que busca el método es aumentar al flujo a través de los caminos de aumento para lograr así alcanzar el flujo máximo. Para encontrar cáminos incrementales en un grafo se puede utilizar una estrategia de búsqueda en anchura (BFS).

3.1.1. Cortes de Redes de flujo

Un corte (S, T) de una red de flujo $G = (V, E)$ es un partición de V en dos conjuntos S y $T = V–S$ tal que $s \in S$ y $t \in T$, es decir la red de flujo se divide en dos redes, una contiene la fuente s y la otra el sumidero t.
Tomando f como flujo:

- $f(S,T)$ es el flujo neto a través del corte (S,T) y se define como

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u).$$

- $c(S,T)$ es la capacidad del corte de (S,T) y se define como

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v).$$

- El flujo neto de una red es el flujo neto a través de cualquier corte de la misma.

- Un corte mínimo en una red es un corte en el cual su capacidad es mínima sobre todos los cortes que hay en esa red.

La razón de estas definiciones viene por el teorema del máximo flujo mínimo corte, que dice que el flujo máximo que puede ir por una red es igual a $c(S,T)$ para algún corte S y T.

Teorema 1 (Teorema del máximo flujo mínimo corte)**.** *Sea f un flujo en una red de flujo $G = (V,E)$ con fuente s sumidero t, las siguientes condiciones son equivalentes:*

- *f es un flujo máximo en G.*

- *La red residual G_f no tiene rutas de alimentación.*

- *$|f| = c(S,T)$ para algún corte (S,T) de G.*

3.2. Algoritmo básico de Ford - Fulkerson

El pseudocódigo del algoritmo Ford-Fulkerson esta dado en el algoritmo 1.

```
Data: G, s, t, f
Result: el flujo óptimo f
foreach arista (u, v) ∈ G.E do
    f(u, v) = 0;
end
while existe un camino p de s a t en la red residual G_f do
    c_f(p) = mín{c_f(u, v) : (u, v) ∈ p};
    foreach cada arista (u, v) ∈ p do
        if (u, v) ∈ G.E then
            f(u, v) = f(u, v) + c_f(p);
        end
        else
            f(v, u) = f(v, u) − c_f(p);
        end
    end
end
```

Algorithm 1: Algoritmo básico de Ford - Fulkerson

y en el cúal se ha basado el algoritmo. Daremos un análisis somero del algoritmo en las siguientes lineas. El algoritmo tiene un coste inicial de $O(E)$ por la inicialización del flujo. Como en el ciclo for se procesa cada arco del camino de aumento, el cuerpo del bucle también tendrá valor $O(E)$.
La cota se obtiene considerando que al aumentar el flujo al menos un arco del camino de aumento va a desaparecer, el de capacidad residual mínima.
Al aumentar la distancia desde el nodo de partida se van eliminando los arcos. Este arco eliminado podría reaparecer de nuevo, pero su distancia será mayor que la fuente. Es fácil ver que el número total de caminos incrementales es $O(VE \max_{u,v} c(u, v))$. Por lo tanto el coste de algoritmo es $O(VE^2 \max_{u,v} c(u, v))$, ver [1].

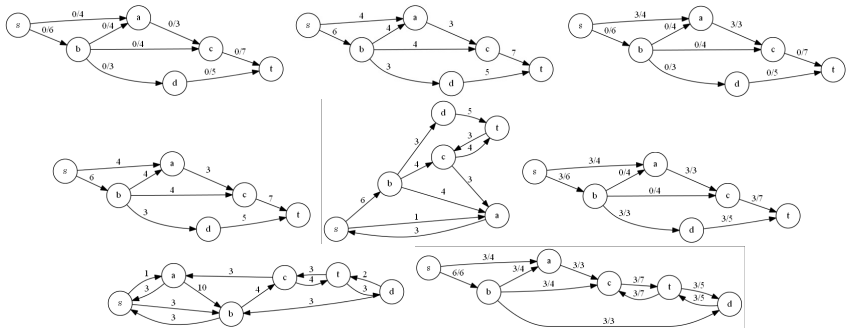

Figura 3.1: Ejemplo del Algoritmo de Ford Fulkerson en un grafo

Discutiremos ahora una ejecución del algoritmo 1, su representación gráfica se puede ver en la figura 3.1. En la primera imagen en la figura 3.1 se comienza con flujo $f = 0$, es decir aún no hay flujo asignado entre los ejes.
Seguidamente $f(s, a) = 3$, la menor capacidad encontrada entre los nodos es 3 y es la que se asigna al flujo. Por lo tanto el incremental es $s \to a \to c \to t$.
A continuación se muestran las capacidades residuales actualizadas $c_f(s, a) = 4-3 = 1$, $c_f(a, c) = 3-3 = 0$; $c_f(c, t) = 10-7 = 3$ y $c_f(a, s) = 3$, $c_f(c, a) = 3$; $c_f(c, t) = 3$
Para la siguiente iteración el camino escogido tiene capacidad 3, por lo tanto sumando al flujo anterior son 6. Ahora el camino incremental es $s \to b \to d \to t$ y por lo tanto las capacidades residuales se actualizan de nuevo a $c_f(s, b) = 6 - 3 = 3$, $c_f(b, d) = 3 - 3 = 0$; $c_f(d, t) = 5 - 3 = 2$ y $c_f(b, s) = 3$, $c_f(d, b) = 3$; $c_f(t, d) = 3$. Repitiendo el procedimiento, llegamos a la solución.

3.2.1. Grafo bipartito

Cuando se habla de un grafo bipartito se está indicando que en ese grafo hay dos subconjuntos de grafos vértices L y R, y los vértices de L solo estan unidos por aristas con los nodos de R y vice versa. El problema que se plantea es el de máximo emparejamiento, esto es encontrar el conjunto de aristas (u, v) mayor de forma que los nodos que aparezcan en una arista, no aparezcan en ninguna otra arista.

Para resolver este problema del emparejamiento en un grafo bipartito se puede utilizar el algoritmo de Ford - Fulkerson. Construimos un grafo de flujos $G' = (V', E')$ a través de un grafo bipartito $G = (V, E)$. Se agregan dos nodos ficticios s y t, $V' = V \cup \{s, t\}$. Se asigna capacidad 1

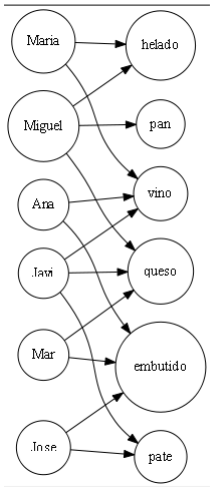

Figura 3.2: Grafo que representa el reparto de la comida entre los amigos, realizando los emparejamientos

a los arcos de E. Las aristas de E dirigidas de R a L son los arcos de G' mas las aristas que van a los nodos ficticios y las que salen de ellos. $E' = \{(s,u) \ : \ u \in L\} \cup \{(u,v) \ : \ u \in L, v \in R, (u,v) \in A) \cup ((v,t) : v \in R)\}$.
A continuación se va a detallar un ejemplo sobre la aplicación del grafo bipartito.
Se va a celebrar una cena de amigos en la que cada uno de los diferentes amigos puede llevar uno del menú, tal y cómo se detalla en la tabla adjunta. El objetivo final es que la cena tenga el máximo número de alimentos sin repetir.

Amigo	Pan	vino	paté	helado	embutido	queso
Ana		x			x	
Jose			x		x	
Miguel	x			x		x
María		x		x		
Javi		x	x			x
Mar					x	x

A continuación, se muestra en la figura 3.2 el grafo obtenido y en los casos en los que se ha podido conseguir emparejamiento directo. Por lo tanto,

con este tipo de grafos queda explicado cómo resolver un problema de emparejamiento. Además, este método se generaliza a otros problemas como el de asignación de tareas, en el que el problema consiste en asignar el mayor número de tareas para que puedan llevarse a cabo y cada persona solo puede realizar una tarea.

3.2.2. Conclusiones e Implementación

El algoritmo de Ford Fulkerson, no detalla la forma de encontrar los caminos aumentados ni el flujo que se pasa a través de cada uno de ellos, por lo que se cosidera un esquema general de un algoritmo que se mejorará con el algoritmo de Edmonds-Karp.
Dicho algoritmo corre en tiempo polinomial siempre que las capacidades no sean muy grandes.
Se genera una anomalía si existen arcos con cantidades finitas o muy grandes, ya que el algoritmo tarda mucho tiempo en en llegar a converger.
Un inconveniente de este tipo de algoritmos es que en cada iteración, va olvidando las etiquetas de los nodos que tienen información de los caminos incrementales desde la fuente a otros nodos, con lo cuál destruye información potencial. Se han realizado pruebas experimentales con el algoritmo Ford-Fulkerson para los cuales se han generado 10 grafos aleatorios para cada número de nodos entre 10 y 300. En la figura 3.3 vemos la evolución de tiempos en milisegundos para con capacidades entre 1 y 1000. El algoritmo Ford-Fulkerson se comporta de forma muy similar cuando el grafo tiene pocas aristas en termino medio, como se puede ver en la figura 3.4, pero el caso peor el tiempo crece mucho más rápido.

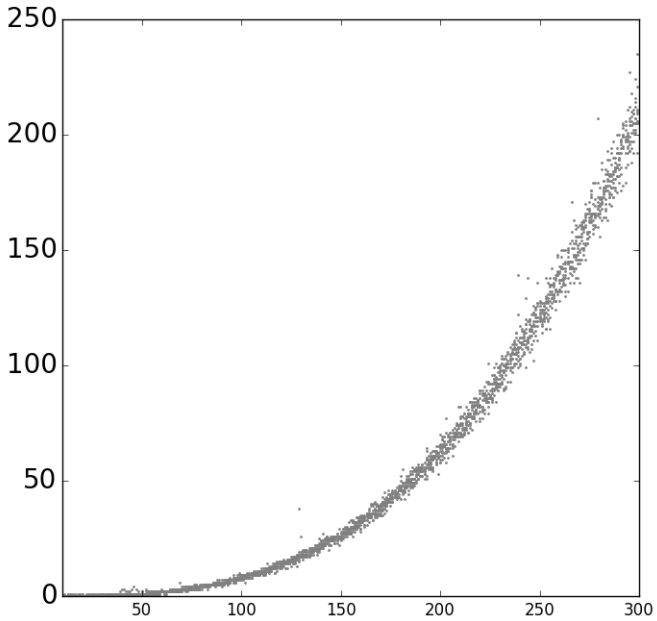

Figura 3.3: Tiempos de ejecución sobre grafos aleatorios del Algoritmo Ford-Fulkerson

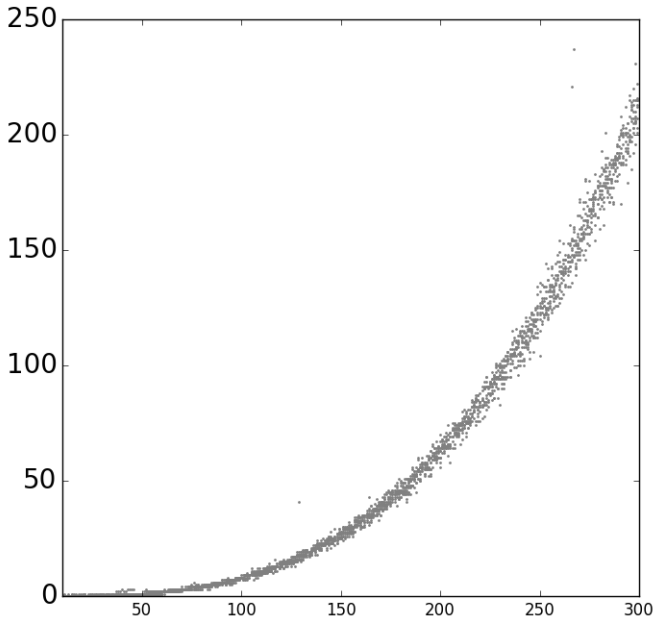

Figura 3.4: Tiempos de ejecución sobre grafos con pocas aristas del Algoritmo Ford-Fulkerson

Capítulo 4

Otros Algoritmos de Flujo Máximo

4.1. Edmonds y Karp

El algoritmo de Edmons Karp es una implementación y mejora del algoritmo y método de Ford-Fulkerson para calcular el flujo máximo de una red de flujo, el procedimiento que sigue es exactamente igual a diferencia que define el orden para buscar los caminos de aumento.

Data: G, s, t, f
Result: el flujo óptimo f
initialization;
foreach $(u, v) \in G$ **do**
    $f[u, v] \leftarrow 0$;
    $f[v, u] \leftarrow 0$;
end
while *se encuentre un camino* p *de* s *a* t *en* G_f **do**
    $c_f(p) \leftarrow \text{mín}\{c_f(u, v) \ : \ (u, v) \in p\}$;
    foreach $(u, v) \in p$ **do**
        $f[u, v] \leftarrow [u, v] + c_f(p)$;
        $f[v, u] \leftarrow [u, v] + c_f(p)$;
    end
end

Algorithm 2: Algoritmo de Edmonds - Karp

4.1.1. Algoritmo de Edmonds - Karp

El algoritmo Edmonds Karp es idéntico al algoritmo de Ford-Fulkerson, excepto que se define el orden de búsqueda para encontrar el trayecto, la trayectoria de aumento encontrar es la más corta entre la fuente y el destino, es decir, la trayectoria que tenga menor número de arcos. Si la capacidad es siempre un número entero, entonces el valor del flujo máximo será entero y existirá un flujo máximo que tenga en cada arco un valor entero. En este caso, la complejidad es $O(VE^2)$ ya que cada camino de aumento se encuentra con un complejidad $O(E)$.

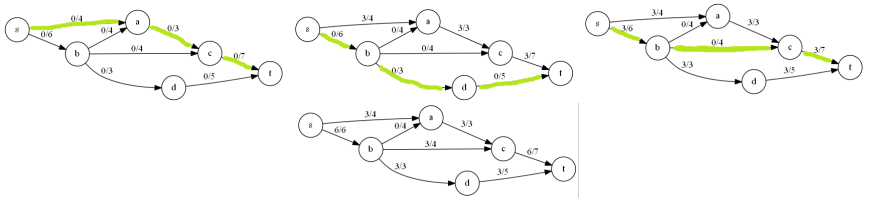

Figura 4.1: Ejemplo del algoritmo Edmonds Karp en un grafo.

En la primera imagen tomamos el camino $P_1 = s \to a \to c \to t$, por lo tanto se tiene un aumento de $c_f(P_1) = 3$, ya que la capacidad es 3.
Para el segundo camino $P_2 = s \to b \to d \to t$, por lo tanto se tiene un aumento de $c_f(P_2) = 3$, ya que la capacidad es 3.
El tercer camino $P_3 = s \to b \to c \to t$, por lo tanto se tiene un aumento de $c_f(P_3) = 3$, ya que la capacidad es 3.
Y por último, se observa que no hay mas caminos incrementales P, por lo que si $P > 0$, el flujo máximo será, $Fmax = 3 + 3 + 3 = 9$.

4.2. Algoritmo Push - Relabel

El algoritmo Push - Relabel es un algoritmo, impementado por Goldberg y Tarjan [8] como mejora al algoritmo de Ford Fulkerson, para calcular la solución al problema del flujo máximo, para ello realizará las dos operaciones de las que toma el nombre Push (empuje) y Relabel (reetiquetado) de flujo [2] [11].
A lo largo de su ejecución, se empieza con un flujo que posteriormente se va convirtiendo gradualmente en flujo máximo de forma local entre los vértices

vecinos mediante la operación Push. Posteriormente realizará la operación Relabel para reetiquetar los flujos.
Durante la ejecución del algoritmo, puede ser que no se cumpla la propiedad de conservación de flujo, es decir, la cantidad de flujo saliente en nodos que no son la fuente es positiva.
Se conoce como preflujo a cualquier función $f : V \times V \mapsto \mathbb{N}$ que cumple restricción de capacidad, por lo tanto el flujo que va a utilizar el algoritmo durante su ejecución se conocerá como preflujo. La cantidad de flujo que tiene un vértice u,será el exceso de flujo de u y lo denotaremos $e(u)$. También definiremos una función llamada altura $h : V \mapsto \mathbb{N}$ que indicara para donde se puede desplazar flujo.
Al comenzar la ejecución del algoritmo, el vértice fuente s, tiene una altura $h(s) = |V|$ y el resto de vértices tomarán como valor de altura cero. La altura de cada vértice va a determinar la dirección hacia dónde se va a enviar el flujo, que siempre será hacia los valores más bajos.
En primer lugar se envía el máximo flujo posible desde la fuente para realizar una serie de iteraciones en las que se ejecutarán las operaciones relabel, que van actualizando las alturas de los vértices, y push, que va empujando el exceso de flujo hacia los vértices que se encuentren más cercanos al destino. Una vez haya concluído el algoritmo se ha realizado una conversión de preflujo a flujo y además se trata de la solución al problema del flujo máximo.

4.2.1. Operación Push

La operación push o empuje se aplica a un vértice u si se cumplen las siguientes condiciones:

- $e(u) > 0 : u$, se trata de un vértice con exceso de flujo.
- $c_f(u, v) > 0$, se puede enviar parte del exceso de flujo a un vértice vecino v.
- $h(u) > h(v)$, la altura del vértice u, es mayor que la de su vecino v.

La cantidad del exceso empujado es el mínimo de exceso de flujo que tiene el vértice u y la cantidad máxima que se pueda envíar por el arco que le conecta con v.
La operación $Push(u, v)$ empuja el flujo de u a v. Si esta operación se aplica sobre una arista (u, v) que sale del vértice u, se aplica push sobre u. Si se

trata un empuje saturado si una arista (u, v) en la red de flujo no admite más flujo, en el caso contrario, se trata de un empuje no saturado.

4.2.2. Operación Relabel

La operación Relabel se encarga de ir actualizando la altura de los nodos cuando tienen exceso de flujo y no pueden enviárselo a ninguno de sus vecinos debido a que tienen una altura menor.
Para poder aplicar la operación deben cumplirse las siguientes condiciones:

- $e(u) > 0 : u$ es un vértice con exceso de flujo.
- Para todo $v \in \{v \ : \ (u, v) \in E_f\} \implies h(u) \leq h(v)$ la altura del vértice u será menor o igual a la de sus vecinos v.

Si se dan estas dos condiciones, entonces se puede cambiar la altura del vértice u y se situará a una altura mayor que la de algunos de sus vecinos, al menos de uno de ellos.

4.3. Análisis

Con respecto al algoritmo de Ford-Fulkerson, Push Relabel va realizando aumentos globales que envían el flujo siguiendo caminos desde la fuente al sumidero, de nodos más elevados a nodos más bajos.
Está considerado uno de los algoritmos de flujo máximo más eficiente, ya que tiene una complejidad polinomial del tiempo $O(|V|^2|E|)$, que asintóticamente es más eficiente que el algoritmo de Edmonds Karp con complejidad asintótica polinomial $O(|V||E|^2)$.

Apéndice

4.4. Implementación del algoritmo de Ford Fulkerson

```
package fordfulkerson;

import java.util.LinkedList;

// Implementacion Java del algoritmo Ford Fulkerson
public class FordFulkerson
{
    private static final int V = 6; // Numero de vertices en el grafo dado

    //Devuelve true si existe una ruta desde el nodo fuente S hasta
    //el destino T en el grafo residual.
    // Tambien rellena el vector padres[]
    //para almacenar la ruta.
    private static boolean bfs(int[][] rGraph,int s,int t,int[] padres)
    {
        // Crea un vector de visitados y marca todos
        // los vertices como no visitados
        boolean[] visitados = new boolean[V];
        for(boolean b : visitados)b = false;
        // Crea una cola, encola el vertice fuente
        // y lo marca como visitado
        LinkedList<Integer> q = new LinkedList<>();
        q.add(s);
        visitados[s] = true;
        padres[s] = -1;
        // Bucle BFS estandar
```

```
        while(!q.isEmpty())
        {
            int u = q.poll();
            for(int v = 0; v < V; v++)
                if(visitados[v] == false && rGraph[u][v] > 0)
                {
                    q.add(v);
                    padres[v] = u;
                    visitados[v] = true;
                }
        }
        // Si alcanzamos el destino en el BFS, partiendo desde
        // el nodo fuente, entonces devuelve true, sino false
        return (visitados[t] == true);
    }
    //Devuelve el maximo flujo desde S hasta T en el grafo dado
    private static int fordFulkerson(int[][] grafo,int s,int t)
    {
        int u, v;
        // Crea un grafo residual y lo rellena con las capacidades
        // dadas en el original
        //pero tomadas como capacidades residuales
        // en este nuevo grafo
        // grafoResidual[i][j] indica la capacidad residual
        // del conector desde I hasta J
        int[][] grafoResidual = new int[V][V];
        for(u = 0; u < V; u++)
            for(v = 0; v < V; v++)
                grafoResidual[u][v] = grafo[u][v];
        // Este vector es rellenado por el BFS para almacenar
        // la ruta de paso
        int[] padre = new int[V];
        // Al principio no hay flujo
        int max_flujo = 0;
        // Ira aumentando segun haya ruta desde el fuente
        // hasta el destino
        while(bfs(grafoResidual,s,t,padre))
        {
            //Encuentra la capacidad residual minima de los conectores
            // a lo largo de la ruta
```

```
            // rellenada por el BFS.
            int flujo_ruta = Integer.MAX_VALUE;
            for(v = t; v != s; v = padre[v])
            {
                u = padre[v];
                flujo_ruta = Math.min(flujo_ruta,grafoResidual[u][v]);
            }
            // actualiza las capacidades residuales de los conectores
            // e invertidos, a lo largo de la ruta
            for(v = t; v != s; v = padre[v])
            {
                u = padre[v];
                grafoResidual[u][v] -= flujo_ruta;
                grafoResidual[v][u] += flujo_ruta;
            }
            // Angade el flujo de la ruta al flujo maximo
            max_flujo += flujo_ruta;
        }
        //Devuelve el flujo maximo
        return max_flujo;
    }
    // Programa de prueba, para testear los metodos hechos
    public static void main(String[] args)
    {
        //Vamos a crear un grafo de ejemplo
        int[][] graph =
        {
            {
                0,4,6,0,0,0
            },
            {
                0,0,0,3,0,0
            },
            {
                0,3,0,4,3,0
            },
            {
                0,0,0,0,0,7
            },
            {
```

```
                0,0,0,0,0,5
            },
            {
                0,0,0,0,0,0
            }
        };
        System.out.println("El maximo flujo posible es " + fordFulkerson(
    }
}
```

4.5. Implementación del algoritmo de Edmonds Karp

```
package edmondskarp;

//Este programa Java implementa el algoritmo de Edmonds Karp,
//el cual se usa para calcular el flujo maximo entre el vertice
// fuente y un destino.
import java.util.ArrayList;
import java.util.*;

public class EdmondsKarp
{
    static class Edge
    {
        int s, t, rev, cap, f;
        public Edge(int s,int t,int rev,int cap)
        {
            this.s = s;
            this.t = t;
            this.rev = rev;
            this.cap = cap;
        }
    }

    public static List<Edge>[] crearGrafo(int nodos)
    {
        List<Arista>[] grafo = new List[nodos];
        for(int i = 0; i < nodos; i++)
```

```
            grafo[i] = new ArrayList<>();
        return grafo;
    }

    public static void anadirArista(List<Arista>[] grafo,int s,int t,int cap)
    {
        grafo[s].add(new Arista(s,t,grafo[t].size(),cap));
        grafo[t].add(new Arista(t,s,grafo[s].size() - 1,0));
    }

    public static int maxFlujo(List<Arista>[] grafo,int s,int t)
    {
        int flujo = 0;
        int[] q = new int[grafo.length];
        while(true)
        {
            int qt = 0;
            q[qt++] = s;
            Edge[] pred = new Arista[grafo.length];
            for(int qh = 0; qh < qt && pred[t] == null; qh++)
            {
                int cur = q[qh];
                for(Arista e : grafo[cur])
                    if(pred[e.t] == null && e.cap > e.f)
                    {
                        pred[e.t] = e;
                        q[qt++] = e.t;
                    }
            }
            if(pred[t] == null)
                break;
            int df = Integer.MAX_VALUE;
            for(int u = t; u != s; u = pred[u].s)
                df = Math.min(df,pred[u].cap - pred[u].f);
            for(int u = t; u != s; u = pred[u].s)
            {
                pred[u].f += df;
                grafo[pred[u].t].get(pred[u].rev).f -= df;
            }
            flujo += df;
```

```
        }
        return flujo;
    }

    // Ejemplo de uso
    public static void main(String[] args)
    {
        List<Arista>[] grafo = crearGrafo(9);
        anadirArista(grafo,0,2,10);
        anadirArista(grafo,1,2,2);
        anadirArista(grafo,1,3,4);
        anadirArista(grafo,1,4,8);
        anadirArista(grafo,2,4,9);
        anadirArista(grafo,3,5,10);
        anadirArista(grafo,4,3,6);
        anadirArista(grafo,4,5,10);
        System.out.println("El flujo maximo posible es: " + maxFlujo(grafo
    }
}
```

Bibliografía

[1] J. Aguilar. Flujo máximo: Redes de flujo y método de Ford-Fulkerson, 2006.

[2] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[3] W. H. Cunningham. A network simplex method. *Mathematical Programming*, 11(1):105–116, 1976.

[4] G. B. Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[5] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[6] L. R. Ford Jr. and D. R. Fulkerson. *Flows in networks*. Princeton university press, 2015.

[7] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[8] A.V. Goldberg. *Efficient graph algorithms for sequential and parallel computers*. Laboratory for Computer Science, Massachusetts Institute of Technology Cambridge, Massachusetts, 1987.

[9] B. Kone, L. Lovasz, H. J. Pröbmel, and A.Schrijver. *Algorithms and Combinatorics*. 1990.

[10] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 253–262. IEEE, 2013.

[11] S. Gutiérrez Mota. Aplicación de las GPUs en la solución del problema del flujo máximo, 2012.

[12] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations research*, 41(2):338–350, 1993.