FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

El problema del flujo máximo: Teoría, algoritmos y aplicaciones.

Ángela López Porta

2018/2019

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

El problema del flujo máximo: Teoría, algoritmos y aplicaciones.

Ángela López Porta

Julio, 2019

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Coñecemento: Estadística e Investigación Operativa
Título: El problema de flujo máximo: Teoría, algoritmos y aplicaciones
Breve descrición do contido
En este trabajo el alumno se familiarizará con una clase importante de problemas de programación lineal. Más concretamente una clase especial de problemas de flujo en redes con coste mínimo: el problema de flujo máximo. Se trata de un problema clásico, ampliamente estudiado en la literatura y que aparece frecuentemente en aplicaciones prácticas, ya sea de forma directa o como subproblema de problemas más complejos.

Índice general

Resumen

A lo largo de esta memoria estudiaremos en profundidad el problema de flujo máximo y veremos algunas de sus aplicaciones prácticas. Este estudio incluirá resultados teóricos que relacionan este problema con otros ya conocidos, como el teorema max-flow min-cut, y se presentará la familia de algoritmos de trayectorias aumentadas, diseñada para la resolución del problema.

Abstract

In this paper we will study the maximum flow problem and we will show some of its practical applications. We will include theorical results to relate this problem with other well known problems, like max-flow min-cut theorem. Moreover, the family of augmenting path algorithms, designed to solve this problem, will be presented.

Introducción

La Programación Lineal es la parte de la Investigación Operativa que se encarga del estudio de problemas de optimización (maximización o minimización) para funciones lineales que, además, deben cumplir una serie de restricciones también lineales. El origen de la Programación Lineal se remonta a la Segunda Guerra Mundial donde se empleaba tanto para la optimización de operaciones militares como para planificar eficientemente el uso de los recursos disponibles con el fin de reducir costes. Una vez finalizada la guerra, las empresas vieron en esta parte de las matemáticas una herramienta eficaz para ampliar sus beneficios y optimizar aspectos de su planificación diaria.

El primer trabajo de Programación Lineal data de 1939 y fue realizado por el matemático y economista ruso L. V. Kantorovich. Sin embargo, el no publicarse hasta 1959 propició que se conozca a G. B. Dantzig como el padre de esta disciplina tras sus trabajos realizados en 1947.

En esta memoria nos centraremos en el estudio de uno de los más conocidos problemas de programación lineal, el problema de flujo máximo. Con la finalidad de que el trabajo sea autocontenido, el primer capítulo se basará en un pequeño resumen donde podremos encontrar desde definiciones de elementos básicos para el manejo de problemas de optimización en redes como pueden ser un grafo o una red con flujo, hasta la formulación del problema dual, el desarrollo del problema de flujo en redes a coste mínimo o resultados relacionados, todo ello visto en la asignatura de Programación Lineal y Entera que se estudia en el Grado de Matemáticas. Una vez recordados todos estos conceptos, dedicaremos el Capítulo 2 a la formulación y desarrollo del problema de flujo máximo. Veremos algunas situaciones cotidianas en las que aparece dicho problema y estudiaremos, tras recordar en qué consiste el problema del conjunto de corte con capacidad mínima, la relación entre ambos problemas. Antes de comenzar a hablar de resolver el problema del flujo máximo, en el tercer capítulo podremos recordar cómo clasificar un algoritmo según su complejidad computacional, la formulación del problema del camino más corto u otros resultados ya conocidos que nos van a ser de utilidad para lo que sigue. Además, también aprenderemos a trabajar con redes residuales y definiremos las etiquetas de distancia asociadas a los nodos

del grafo, pues ambas cosas son empleadas posteriormente para el diseño de los algoritmos de resolución. Ya el cuarto capítulo lo dedicaremos a definir la familia de los algoritmos de trayectorias aumentadas y analizaremos en detalle el algoritmo de etiquetado, que es un caso particular de esta familia. Como veremos, este algoritmo posee tres grandes debilidades, por lo que en el último capítulo nuestro objetivo está centrado en implementar variaciones del algoritmo que sean capaces de subsanar dichos problemas, obteniendo así algoritmos más eficientes para resolver el problema de flujo máximo. Además, sobre un ejemplo particular, desarrollaremos las ideas de una familia alternativa de algoritmos, conocidos como preflow-push, que hoy en día son considerados superiores a los algoritmos de trayectorias aumentadas (desde el punto de vista de eficiencia computacional).

Capítulo 1

El Problema de Flujo en Redes con Coste Mínimo

Con este primer capítulo se pretende rememorar aquellos conceptos de la asignatura de Programación Lineal y Entera cursada en el grado que nos servirán de apoyo para poder hablar con fluidez del problema a estudiar, el problema de flujo máximo. Asimismo, veremos una serie de resultados interesantes para los cuales obviaremos su demostración por estar ya vista en dicha asignatura. Tomaremos como referencia a seguir los apuntes del profesor Julio González Díaz (2018-2019).

1.1. Programación lineal

La *programación lineal* tiene como objetivo optimizar (minimizar o maximizar) funciones lineales condicionadas por una serie de restricciones también lineales, de igualdad o de desigualdad.

Para poder formular un problema de programación matemática general necesitaremos identificar una serie de elementos en nuestro problema:

- Variables de decisión, $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Representan las distintas decisiones para las cuales estamos buscando una solución óptima.
- Función objetivo $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. Representa o bien el coste o bien el beneficio asociado a las variables de decisión. Nuestra tarea consistirá en maximizar o minimizar esta función.
- Restricciones, representan que valores para los cuales las variables $x_1, \ldots, x_n$ serían *factibles*. El conjunto de puntos verificando las restricciones se le denomina *región factible*. Pueden ser de dos tipos:

- Restricciones de desigualdad, $g_i(x) \leq 0,\ i \in \{1,\ldots,m\}$ $(g_i : \mathbb{R}^n \longrightarrow \mathbb{R})$.
- Restricciones de igualdad, $h_j(x) = 0,\ j \in \{1,\ldots,l\}$ $(h_j : \mathbb{R}^n \longrightarrow \mathbb{R})$.

Así, un problema de *programación matemática* puede ser representado como:

$$\begin{array}{lll} \text{Minimizar} & f(x) & \\ \text{sujeto a:} & g_i(x) \leq 0 & i = 1,\ldots,m \\ & h_j(x) = 0 & j = 1,\ldots,l. \end{array}$$

Si además imponemos que las tanto la función objetivo como las restricciones sean todas lineales, obtendremos la representación de un problema de programación lineal.

Más comúnmente, podemos ver representado el problema de programación lineal de forma matricial. Para ello, definimos:

- Vector de costes, $\boldsymbol{c} \in \mathbb{R}^n$.
- Matriz de restricciones $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, con elementos de la forma a_{ij}.
- Vector de lados derechos, $\boldsymbol{b} \in \mathbb{R}^m$.

Identificando estos elementos podemos escribir el problema de programación lineal como:

$$\begin{array}{ll} \text{Minimizar} & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ \text{sujeto a:} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array}$$

Recordemos que siempre podemos transformar un problema de maximización en uno de minimización, y viceversa, multiplicando todos los coeficientes de la función objetivo por -1. La solución óptima para el problema original será la solución del nuevo problema pero multiplicada por -1. Esto se debe a que se verifica la siguiente igualdad:

$$\text{máx} \sum_{j=1}^{n} c_j x_j = -\,\text{mín} \sum_{j=1}^{n} -c_j x_j.$$

En cuanto a las restricciones,también podemos transformar una desigualdad del tipo "$\geq$" en otra del tipo "$\leq$", y viceversa, multiplicando por -1 ambos lados de la desigualdad. Además, podemos transformar desigualdades en igualdades añadiendo unas nuevas variables, conocidas como *variables de holgura*. Dada una restricción $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$, se tendría $\sum_{j=1}^{n} a_{ij}x_j - x_i^s = b_i$ con $x_i^s \geq 0$. Análogamente, si tuviéramos $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, se le sumaría esta nueva variable.

Para la resolución de los problemas de programación lineal utilizamos el *método símplex*. Este consiste en ir saltando de punto extremo en punto extremo de la región factible hasta encontrar un punto extremo óptimo o hasta concluir que no existe tal óptimo.

Como veremos más adelante en la Sección 3.2, para analizar el rendimiento de un algoritmo se puede recurrir a diferentes técnicas. Entre ellas destacan el análisis del peor caso y el análisis del caso promedio. El análisis del peor caso mira el rendimiento del algoritmo para aquellas clases de problemas que le resultan más difíciles de resolver. Por otra parte, el análisis del caso promedio mira el rendimiento promedio del algoritmo sobre todos los problemas de una determinada clase. El método símplex se comporta de forma diferente dependiendo de si usamos un análisis u otro. Mientras que en el caso promedio se obtiene un rendimiento polinomial, utilizando el análisis del peor caso el rendimiento pasa a ser exponencial[1].

1.1.1. Dualidad

Dado un problema de programación lineal, al que denominaremos primal, podemos asociarle un problema de programación lineal conocido como el *problema dual*. El estudio de este nuevo problema resulta de gran interés pues nos proporciona una solución para el problema primal. Además, es muy útil en el diseño de algoritmos eficientes para resolver tanto problemas de programación lineal como algunos problemas de optimización en redes, que introduciremos más adelante.

La formulación de un problema dual a través de un primal es la siguiente:

$$\begin{array}{rl} \text{(P) Minimizar} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{sujeto a:} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array} \qquad \begin{array}{rl} \text{(D) Maximizar} & \boldsymbol{b}^\top \boldsymbol{\pi} \ (= \boldsymbol{\pi}^\top \boldsymbol{b}) \\ \text{sujeto a:} & \boldsymbol{A}^\top \boldsymbol{\pi} \leq \boldsymbol{c} \\ & \boldsymbol{\pi} \text{ no restringido.} \end{array}$$

Figura 1.1: Problemas primal y dual en su forma estándar.

$$\begin{array}{rl} \text{(P) Minimizar} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{sujeto a:} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array} \qquad \begin{array}{rl} \text{(D) Maximizar} & \boldsymbol{b}^\top \boldsymbol{\pi} \ (= \boldsymbol{\pi}^\top \boldsymbol{b}) \\ \text{sujeto a:} & \boldsymbol{A}^\top \boldsymbol{\pi} \leq \boldsymbol{c} \\ & \boldsymbol{\pi} \geq 0. \end{array}$$

Figura 1.2: Problemas primal y dual en su forma canónica.

[1] El tiempo que necesita para resolver el "peor" problema para un grafo con m aristas y n nodos, crece exponencialmente en función de estos dos parámetros.

Diremos que $\boldsymbol{\pi}$ es el vector de variables duales del problema, y, a partir de dichas formulaciones, se deduce el siguiente conjunto de reglas para transformar un problema en el otro:

Problema primal		Problema dual	
F.objetivo	Minimizar	Maximizar	F.objetivo
Restricciones	$\geq b_i$	≥ 0	Variables
	$\leq b_i$	≤ 0	
	$= b_i$	no restring.	
Variables	≥ 0	$\leq c_j$	Restricciones
	≤ 0	$\geq c_j$	
	no restring.	$= c_j$	

Figura 1.3: Correspondencias entre el problema primal y el dual.

Ejemplo 1.1. Consideremos el siguiente problema de programación lineal

$$\begin{aligned} \text{(P)} \qquad \text{Min} \quad & 4x_1 + 2x_2 + 6x_3 + x_4 \\ \text{sujeto a:} \quad & x_1 + x_2 + x_3 \geq 6 \\ & x_3 + x_4 \leq 4 \\ & x_1 + x_3 = 2 \\ & \boldsymbol{x} \geq 0. \end{aligned}$$

A partir de las reglas mencionadas en la tabla anterior, se tiene que la formulación para el problema dual asociado será:

$$\begin{aligned} \text{(D)} \qquad \text{Máx} \quad & 6\pi_1 + 4\pi_2 + 2\pi_3 \\ \text{sujeto a:} \quad & \pi_1 + \pi_3 \leq 4 \\ & \pi_1 \leq 2 \\ & \pi_1 + \pi_2 + \pi_3 \leq 6 \\ & \pi_2 \leq 1 \\ & \pi_1 \geq 0 \\ & \pi_2 \leq 0 \\ & \pi_3 \quad \text{no restrig.} \end{aligned}$$

Puesto que el problema dual no deja de ser un problema de programación lineal podemos formular el dual del mismo, lo que nos lleva al siguiente resultado:

Proposición 1.2. *El dual del dual coincide con el primal.*

Consideremos ahora un problema de programación lineal y su dual expresados en su forma canónica. Si $\boldsymbol{x}$ y $\boldsymbol{\pi}$ son dos soluciones factibles para los respectivos problemas, entonces verifican que $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$, $\boldsymbol{\pi}^\top \boldsymbol{A} \leq \boldsymbol{c}^\top$, $\boldsymbol{x} \geq \boldsymbol{0}$ y $\boldsymbol{\pi} \geq \boldsymbol{0}$. De estas desigualdades se puede seguir:

$$\boldsymbol{c}^\top \boldsymbol{x} \geq (\boldsymbol{\pi}^\top \boldsymbol{A})\boldsymbol{x} = \boldsymbol{\pi}^\top (\boldsymbol{A}\boldsymbol{x}) \geq \boldsymbol{\pi}^\top \boldsymbol{b}$$

Es decir, la función objetivo asociada a cualquier solución factible para el problema de minimización siempre será mayor o igual que la función objetivo para el problema de maximización. Esta propiedad es conocida como la propiedad de *dualidad débil*.

Presentamos, además, la siguiente relación entre las soluciones óptimas de ambos problemas.

Proposición 1.3 (Dualidad fuerte)**.** *Dados un problema de programación lineal y su dual, si uno de ellos tiene solución óptima entonces el otro también la tiene y ambos valores óptimos en la función objetivo coinciden.*

1.1.2. Programación Lineal Entera

En algunos problemas de programación lineal surge la necesidad de imponer que todas nuestras variables tomen valores únicamente enteros, puesto que en nuestro contexto no tendría sentido hablar de variables de decisión divisibles. Esta nueva restricción da lugar a un subproblema conocido como el *problema de programación (lineal) entera*.

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeto a:} & g_i(x) \leq 0 \ i = 1, \ldots, m \\ & h_j(x) = 0 \ j = 1, \ldots, l \\ & x \in \mathbb{Z}^n. \end{array}$$

Resulta casi inmediato pensar en resolver estos problemas mediante el método símplex, obviando que las variables deben ser enteras, es decir, relajando el problema, y después redondear el resultado obtenido. Sin embargo, con este procedimiento se pueden llegar a obtener resultados muy lejanos al óptimo real. Por ello, para resolver estos problemas eficazmente se utiliza el método de *ramificación y acotación*. Este método consiste en, a partir de observaciones anteriores, subdividir sucesivamente la región factible del problema de partida y resolver las versiones relajadas de los subproblemas resultantes, como ya hemos visto en la asignatura del grado. El método de ramificación y acotación es de tipo exponencial, por lo que se podría decir que es peor que el método símplex al necesitar más tiempo de ejecución.

1.2. Grafos y Redes con Flujo

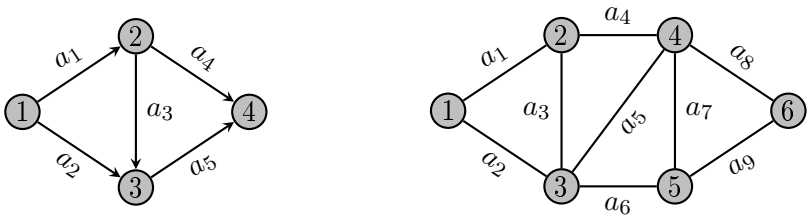
Para resolver los problemas de optimización en redes, y más particularmente el problema de flujo máximo que queremos estudiar, es muy habitual utilizar representaciones gráficas que nos permitan visualizar tales problemas con facilidad. Es por ello que debemos recordar la definición de grafo y redes con flujo.

Un *grafo* G es un par (N, A) en el cual N es el conjunto de elementos llamados *nodos* o *vértices* y A es el conjunto de elementos llamados *aristas* o *arcos*. Podemos distinguir dos tipos de grafos en función de A:

- Grafo dirigido: G se dice que es un grafo dirigido si $A \subseteq N \times N$, es decir, la arista (i, j) comienza en el nodo i y termina en el nodo j.

- Grafo no dirigido: En este caso A se compone de subconjuntos de N de dos elementos. Los conjuntos $\{i, j\}$ y $\{j, i\}$ representan la misma arista.

En lo relativo a este trabajo, **solamente utilizaremos grafos dirigidos**.



(a) Grafo dirigido. En este caso $N = \{1, 2, 3, 4\}$ y $A = \{a_1, \ldots, a_5\}$.

(b) Grafo no dirigido. En este caso $N = \{1, \ldots, 6\}$ y $A = \{a_1, \ldots, a_9\}$.

Figura 1.4: Ejemplo de un grafo dirigido y uno no dirigido.

En ocasiones, es conveniente expresar la información dada en los grafos dirigidos de forma matricial. Para ello, dado un grafo $G = (N, A)$ dirigido con conjunto de nodos $N = \{1, \ldots, n\}$ y conjunto de aristas $A = \{a_1, a_2, \ldots, a_m\}$, se define la *matriz de adyacencia* del grafo, $\overline{\boldsymbol{A}}_{\boldsymbol{n \times n}}$, como:

$$\overline{a}_{ij} = \begin{cases} 1 & \text{si } (i, j) \text{ es un arco de } G \\ 0 & \text{en otro caso.} \end{cases}$$

Otra posible forma de representar el grafo dirigido G es a través de la *matriz de inci-*

dencia, $\overline{\boldsymbol{B}}_{\boldsymbol{n\times m}}$, que se define como:

$$\overline{b}_{ik} = \begin{cases} 1 & \text{si } i \text{ es el nodo inicial de } a_k \\ -1 & \text{si } i \text{ es el nodo terminal de } a_k \\ 0 & \text{en otro caso.} \end{cases}$$

$$\overline{\boldsymbol{A}} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \overline{\boldsymbol{B}} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{pmatrix}$$

(a) Matriz de adyacencia (b) Matriz de incidencia

Figura 1.5: Matrices de adyacencia e incidencia asociadas al grafo 1.4(a).

1.2.1. Redes con Flujo

Una *red* es un grafo con uno o más números asociados a cada arista o nodo. Dichas cantidades pueden representar costes, distancias,... Denominamos *flujo* al envío de elementos de un lugar a otro dentro de la red, es decir, de un nodo a otro nodo. Un flujo podría representar el transporte de mercancías entre una fábrica y sus distintos puntos de distribución. Los modelos correspondientes a estas características los llamaremos *modelos de redes con flujo*.

Los objetos que se envían a través de la red los denominamos *unidades de flujo* o simplemente *unidades*. Las unidades de flujo podrían ser personas, objetos,...

En nuestro problema de optimización asociaremos a toda arista k un flujo f_k obtenido tras resolver el problema, es decir, los flujos f_k son las variables de decisión del problema. Dicho flujo depende de los siguientes tres parámetros:

Cota inferior $l_k \geq 0$: cantidad mínima de flujo debe de enviarse por la arista k.

Capacidad o cota superior $u_k \geq 0$: cantidad máxima de flujo que se puede enviar por la arista k.

Coste o beneficio c_k: si es positivo denota el coste de enviar una unidad de flujo por la arista k. Si es negativo, en cambio, representa el beneficio.

En aquellos casos en los que nos interese hacer referencia a los nodos incidentes, i y j, de una arista k, pasaremos a referirnos a esta arista como la arista (i, j) y tendremos los valores asociados l_{ij}, u_{ij} y c_{ij}.

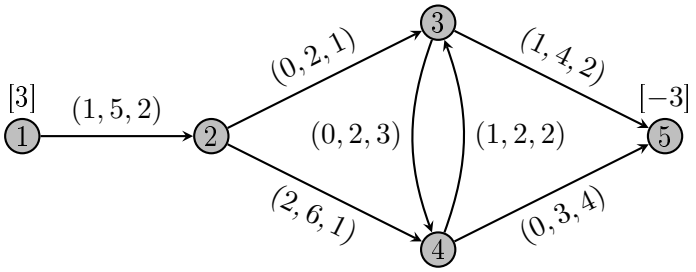


Figura 1.6: Ejemplo de una red con flujo.

En ocasiones podemos encontrar modelos de redes con flujo en los cuales, por algunos de los nodos, entre o salga flujo por la red. En ese caso diríamos que tenemos *flujos externos*. Podemos tener un *flujo externo fijo*, es decir, una cantidad fija de flujo, que será positiva si entra a la red y negativa si sale de ella; o bien una cantidad variable, el *flujo externo de holgura*. En este último caso se asocia a cada nodo i los siguientes parámetros:

Cota inferior, $\boldsymbol{l_i^e}$: Si es una cantidad positiva denota la cantidad mínima de flujo que entra a la red por el nodo i, y si es negativa denota la máxima cantidad de flujo que sale de la red por el nodo i.

Cota superior o capacidad, $\boldsymbol{u_i^e}$: Si es positivo hace referencia a la cantidad máxima de flujo que entra por el nodo i a la red, y si es negativa denota la cantidad mínima que debe salir por dicho nodo.

Coste o beneficio, $\boldsymbol{c_i^e}$: Si es positivo denota el coste por unidad de flujo que circula por el nodo i, y si es negativa el beneficio.

Dada una red con flujo R podemos expresarla como $R = ((N, A), (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}), (\boldsymbol{l^e}, \boldsymbol{u^e}, \boldsymbol{c^e}))$ donde (N, A) es el grafo asociado a la red, $(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c})$ son las capacidades y costes de los arcos, y $(\boldsymbol{l^e}, \boldsymbol{u^e}, \boldsymbol{c^e})$ son los flujos externos con sus costes.

Debemos observar que siempre podemos transformar una red con flujos externos en otra sin ellos añadiendo un nuevo nodo auxiliar. Así, si un nodo i tiene flujo externo positivo, añadimos un arco desde el nodo auxiliar al nodo i. Si, por el contrario, el nodo tiene un flujo externo negativo, el arco irá del nodo i al nodo auxiliar. Los parámetros de los nuevos arcos se determinan a través de los parámetros que teníamos para el nodo i. Ilustramos este procedimiento en la Figura 1.7.

1.3. PFCM

Para definir el problema general de flujo en redes con coste mínimo vamos a considerar la ausencia de flujos externos en la red, lo cual no es restrictivo por lo visto anteriormente.

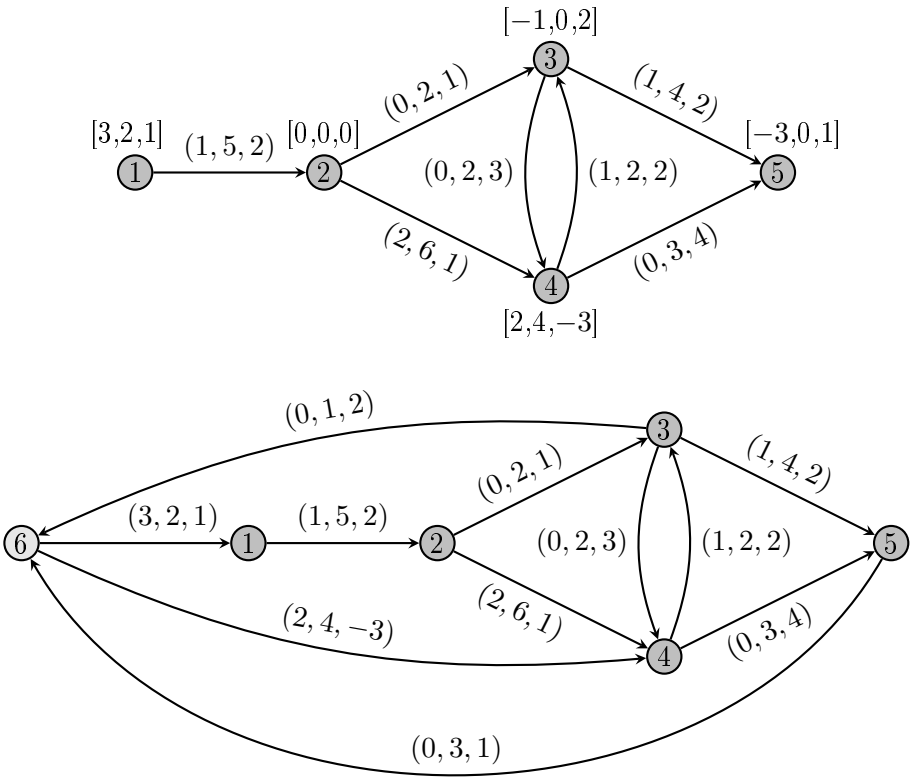


Figura 1.7: Paso a una red sin flujos externos.

Por ello, es claro imponer la restricción de *conservación de flujo*, que establece que la cantidad de flujo que entra en un nodo tiene que coincidir con la cantidad de flujo que sale de él. En otras palabras, ningún nodo puede generar o eliminar flujo.

Dada una red con flujo, el *problema de flujo en redes con coste mínimo* consiste en establecer el flujo que debe pasar por cada arco de modo que el coste sea mínimo y, a su vez, se respeten las restricciones de capacidad de los arcos y la conservación de flujo de cada nodo. Para mayor comodidad en la lectura, nos referiremos a este problema como el PFCM.

Veamos que un PFCM se puede escribir como un problema de programación lineal. Si denotamos por A_{O_i} el conjunto de arcos que salen del nodo i y por A_{T_i} el conjunto de arcos que terminan en el nodo i, obtenemos la formulación:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{k \in A} c_k f_k & \\ \text{sujeto a:} \quad & \sum_{k \in A_{O_i}} f_k - \sum_{k \in A_{T_i}} f_k = 0 & i \in N \\ & f_k \leq u_k & k \in A \\ & f_k \geq l_k & k \in A. \end{aligned}$$

Las restricciones del problema no son más que las ya mencionadas anteriormente, la conservación de flujo y las restricciones de capacidad de los arcos. La función objetivo en este caso es el coste que supone enviar el flujo por la red. Visto de este modo, está claro que este problema lo podremos resolver mediante el método símplex.

Además, podemos obtener una representación matricial del problema. Sea $\boldsymbol{A}$ la matriz de restricciones, $\overline{\boldsymbol{B}}$ la matriz de incidencia asociada al grafo y $\boldsymbol{b}$ el vector con las constantes del lado derecho. Dado un vector $\boldsymbol{v}$, que es un vector columna, denotamos su traspuesto como $\boldsymbol{v}^\intercal$. Si denotamos por $\boldsymbol{c}$, $\boldsymbol{f}$, $\boldsymbol{u}$ y $\boldsymbol{l}$ los vectores coste, flujos, cotas superiores y cotas inferiores, respectivamente, tendríamos:

$$\begin{aligned} \text{Minimizar} \quad & \boldsymbol{c}^\intercal \boldsymbol{f} \\ \text{sujeto a:} \quad & \overline{\boldsymbol{B}}_{n \times m} \boldsymbol{f} = \boldsymbol{0} \\ & \boldsymbol{I}_{m \times m} \boldsymbol{f} \leq \boldsymbol{u} \\ & \boldsymbol{I}_{m \times m} \boldsymbol{f} \geq \boldsymbol{l}. \end{aligned}$$

1.4. PFCM con variables enteras: Unimodularidad

En ocasiones, necesitaremos imponer en nuestro PFCM que las variables tomen valores enteros con lo que obtendríamos un problema de programación lineal entera, que, por lo

general, es más costoso de resolver. Sin embargo, gracias a la estructura del PFCM, veremos que podemos resolverlo mediante el método símplex. Para ello, trabajaremos con un nuevo concepto, la unimodularidad.

Definición 1.4. Una matriz cuadrada $\boldsymbol{A} \in \mathbb{Z}^{p\times p}$ se dice que es *unimodular* si tiene por determinante 1 o -1. Además, $\boldsymbol{A} \in \mathbb{Z}^{p\times q}$ se dice *totalmente unimodular* si cualquier submatriz cuadrada es singular o unimodular.

Teniendo en cuenta que todos los elementos de una matriz pueden ser considerados como submatrices 1×1, se deduce que una matriz totalmente unimodular está compuesta únicamente de los números 0, 1 y -1.

Proposición 1.5. *Dada una matriz totalmente unimodular $\boldsymbol{A} \in \mathbb{Z}^{p\times q}$ y un vector $\boldsymbol{b} \in \mathbb{Z}^{p}$, tenemos que cualquier solución básica factible definida por las restricciones $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$, tiene todas sus componentes enteras.*

Por tanto, si tenemos un vector de lados derechos con componentes enteras y una matriz de restricciones totalmente unimodular, la proposición anterior nos asegura que todas las soluciones básicas factibles que encontremos para el problema $\boldsymbol{Ax} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}$, tendrán todos sus elementos enteros. Particularmente, si existen óptimos finitos, alguno de ellos debe tomar valores enteros. Esto nos permite obviar la condición de integralidad de las variables y resolver el problema de programación lineal con el método símplex.

Proposición 1.6. *La matriz de incidencia $\overline{\boldsymbol{B}}$ de un grafo dirigido es totalmente unimodular.*

Recordando la formulación matricial que teníamos para el problema de flujo en redes a coste mínimo

$$\begin{aligned} \text{Minimizar} \quad & \boldsymbol{c}^{\top}\boldsymbol{f} \\ \text{sujeto a:} \quad & \overline{\boldsymbol{B}}_{n\times m}\boldsymbol{f} = \boldsymbol{0} \\ & \boldsymbol{I}_{m\times m}\boldsymbol{f} \leq \boldsymbol{u} \\ & \boldsymbol{I}_{m\times m}\boldsymbol{f} \geq \boldsymbol{l}, \end{aligned}$$

veamos que podemos transformar este problema en otro problema equivalente de la forma $\boldsymbol{Af} = \boldsymbol{b}, \boldsymbol{f} \geq \boldsymbol{0}$, con $\boldsymbol{b}$ entero y $\boldsymbol{A}$ totalmente unimodular.
Por una parte, como $\boldsymbol{l} \geq \boldsymbol{0}$, la condición $\boldsymbol{f} \geq \boldsymbol{0}$ es trivial. Para obtener $\boldsymbol{Af} = \boldsymbol{b}$ tomamos como vector de lados derechos el vector $\boldsymbol{b} = (0, \ldots, 0, u_1, \ldots, u_m, l_1, \ldots, l_m)$ y como matriz de restricciones la matriz

$$\boldsymbol{A} = \begin{pmatrix} \overline{\boldsymbol{B}}_{n\times m} & \boldsymbol{0}_{n\times m} & \boldsymbol{0}_{n\times m} \\ \boldsymbol{I}_{m\times m} & \boldsymbol{I}_{m\times m} & \boldsymbol{0}_{m\times m} \\ \boldsymbol{I}_{m\times m} & \boldsymbol{0}_{m\times m} & -\boldsymbol{I}_{m\times m} \end{pmatrix}.$$

Proposición 1.7. *La matriz $\boldsymbol{A}$ definida anteriormente es totalmente unimodular si, y solo si, $\overline{\boldsymbol{B}}$ es totalmente unimodular.*

Concluimos, basándonos en los resultados previos, que para el caso de un PFCM no es restrictivo imponer que las variables sean enteras, puesto que al resolver el propio problema de programación lineal relajado las soluciones factibles obtenidas siempre tendrán sus componentes todas enteras.

Capítulo 2

El problema de flujo máximo

Para el desarrollo de este capítulo hemos tomado como referencia el libro Ahuja et al. (1993), más concretamente, el Capítulo 6.

2.1. Introducción

El *problema de flujo máximo* consiste en maximizar la cantidad de flujo que se puede enviar entre dos nodos prefijados de la red, respetando las capacidades de los arcos. De nuevo, por comodidad, denotaremos este problema como PFM. El flujo máximo que estamos buscando lo denotaremos como F, y, sin pérdida de generalidad, supondremos que queremos pasar el flujo del nodo 1 al nodo n. Dichos nodos se llamarán *fuente* y *sumidero*, respectivamente, y los denotaremos por s y t.

Sobre una red $R = (G, (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}))$ cuyas capacidades de los arcos son no negativas, $u_{ij} \geq 0$, definimos $U := \text{máx}\{u_{ij} : u_{ij} < \infty \ y \ (i,j) \in A\}$. Asimismo, definimos la lista de adyacencia del nodo i como $A(i) = \{(i,k) : (i,k) \in A\}$, es decir, el conjunto de todas las aristas que salen del nodo i.

Podemos escribir el problema de flujo máximo como un problema de programación lineal como sigue:

$$\begin{aligned} &\text{Maximizar} \quad F \\ &\text{sujeto a:} \quad \sum_{\{j:(i,j)\in A\}} f_{ij} - \sum_{\{j:(j,i)\in A\}} f_{ji} = \begin{cases} F & \text{si } i = s \\ 0 & \text{si } i \in N - \{s,t\} \\ -F & \text{si } i = t \end{cases} \\ &\qquad\qquad 0 \leq f_{ij} \leq u_{ij} \quad \forall (i,j) \in A. \end{aligned}$$

El problema de flujo máximo puede ser representado fácilmente como un PFCM en el cual no existen los costes asociados a cada unidad de flujo. Para pasar de un problema a

otro simplemente añadimos un arco "de vuelta" desde el nodo sumidero al nodo fuente. A este arco le asignamos cualquier coste negativo, que es en realidad beneficio, y no ponemos limitaciones a su capacidad. El resto de los arcos tendrán asociado un coste 0. Así vemos claramente que el objetivo radicará en mandar todo el flujo que sea posible por este arco de vuelta.

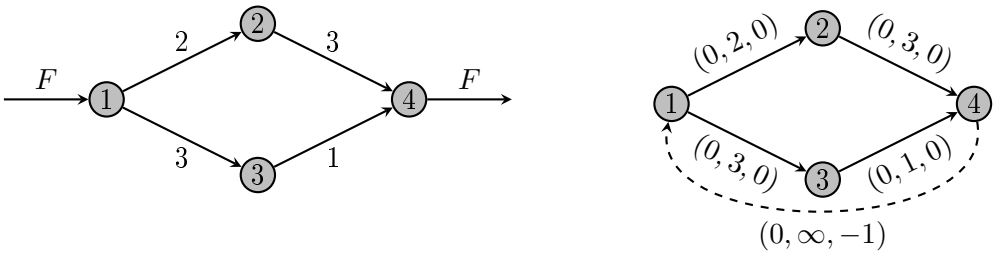


Figura 2.1: Transformación de un PFM a un PFCM.

A lo largo de toda la memoria tendremos en consideración lo siguiente:

- El grafo G es siempre dirigido.
- Todas las capacidades son enteras no negativas.
- Dado un arco $(i, j) \in A$, suponemos que $(j, i) \in A$ sin pérdida de generalidad, pues podemos imponer $u_{ji} = 0$.
- No existen dos o más arcos desde el nodo i al nodo j, para cualesquiera i y j en la red.

2.2. Aplicaciones

Hay que tener en consideración que el problema de flujo máximo surge naturalmente al modelizar otra serie de problemas que, en primera instancia, nada tienen que ver con el problema de flujo máximo. Este hecho nos permite utilizar los algoritmos de resolución del PFM, que veremos más adelante, para hallar las soluciones de los problemas.

2.2.1. Problema de flujo factible

El *problema de flujo factible* consta de identificar un flujo f en una red $R = (G, (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}))$ satisfaciendo:

$$\sum_{\{j:(i,j)\in A\}} f_{ij} - \sum_{\{j:(i,j)\in A\}} f_{ji} = b(i) \quad \forall i \in N$$
$$0 \leq f_{ij} \leq u_{ij} \qquad \forall (i,j) \in A.$$

Además, asumimos que $\sum_{i\in N} b(i) = 0$, es decir, se compensan los flujos externos.

Un ejemplo de cómo surge este problema en la práctica podría ser el siguiente:
Cierta empresa con almacenes en varios puertos necesita distribuir la mercancía de unos a otros para intentar satisfacer todas las demandas. Se conoce en que puertos hay stock, en cuales otros es necesario reponer unidades, y en qué cantidades respectivamente. Además, entre cada par de puertos, existe un número máximo de unidades que pueden ser enviadas. Queremos averiguar si es posible satisfacer todas las demandas con el stock disponible.

Lo modelizamos como un problema de flujo máximo definiendo una nueva red de flujo con las siguientes características: introducimos dos nuevos nodos, la fuente y el sumidero. Para cada nodo con $b(i) > 0$, introducimos un arco (s, i) con capacidad máxima $b(i)$, y por cada nodo con $b(i) < 0$, introducimos un arco (i, t) con capacidad $-b(i)$. Esta nueva red que acabamos de crear se le denomina *red transformada*.

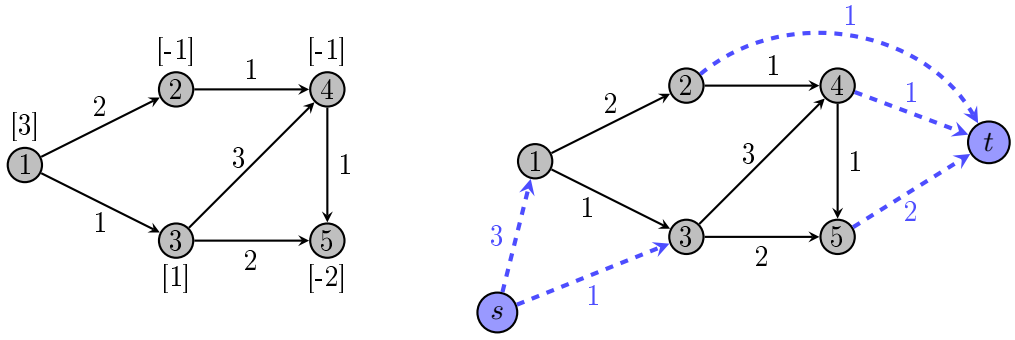


Figura 2.2: Transformación del problema del flujo factible en un PFM.

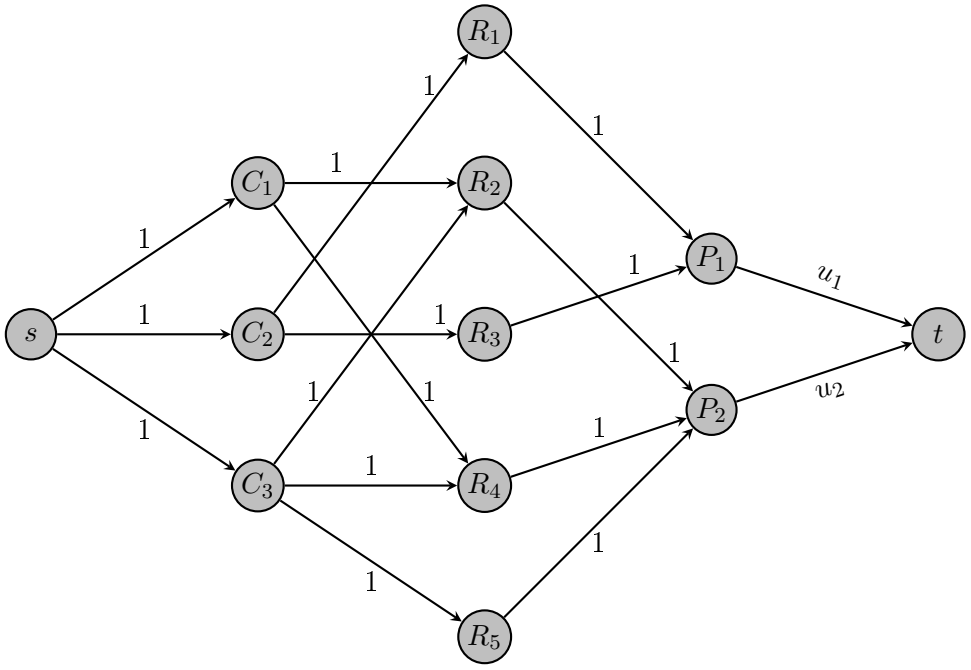
Resolviendo el problema de flujo máximo entre los nodos s y t, obtenemos que si los flujos asociados al flujo máximo saturan todos los arcos que salen de la fuente y todos los arcos que llegan al sumidero, el problema de flujo factible posee una solución factible.

2.2.2. Problema de los representantes

Supongamos que en una ciudad existen q clubes, $C_1, \ldots, C_q$, r residentes, $R_1, \ldots, R_r$ y p partidos políticos, $P_1, \ldots, P_p$. Cada club debe escoger a uno de sus miembros para representarlo en el consejo de gobierno. Además, cada residente de la ciudad pertenece a exactamente un partido político y, por lo menos, a un club. Por último, se decide que, a lo sumo, puede haber u_k afiliados del partido P_k. Se quiere determinar si existe un consejo de gobierno equilibrado satisfaciendo estas condiciones.

Veamos como modelizamos este problema como un problema de flujo máximo: Para comenzar creamos la red de flujo partiendo de un nodo fuente s y un nodo sumidero t. Añadimos todos los arcos (s, C_i), donde C_i está denotando a los distintos clubes. Además, solo si el residente R_j pertenece al club C_i, añadimos el arco (C_i, R_j), y si pertenece al partido P_k añadimos el arco (R_j, P_k). Todos estos arcos se definen con capacidad unitaria. Completamos la red añadiendo los arcos (P_k, t), con capacidades u_k, respectivamente.

Para un caso particular se tendría:



Podemos deducir que solo se obtiene un consejo equilibrado si el valor del flujo máximo sobre la red es q. Si obtenemos un flujo q se saturarían todas las aristas (s, C_i), pues tienen capacidad unitaria, lo que significa que todos los clubes pueden escoger a un representante de modo que se satisfagan las condiciones prefijadas respecto al número de integrantes de cada partido político.

2.2.3. Problema de redondeo de matrices

En este problema queremos hacer un redondeo consistente de los elementos de una matriz. Esto puede surgir en numerosos ámbitos, como por ejemplo para una mejor comprensión de datos se suelen presentar las cifras redondeadas. Dada una matriz $p \times q$ de números reales $D = \{d_{ij}\}$, podemos redondear sus elementos a la baja $\lfloor d_{ij} \rfloor$ o al alza $\lceil d_{ij} \rceil$, según nuestro criterio. Al redondear los elementos de la matriz, debe verificarse que la suma por filas de los elementos redondeados es igual al redondeo de la suma por filas y lo mismo para las sumas por columnas.

				Sumas por filas
	1.26	2.08	3.15	6.49
	5.4	7.16	1.45	14.01
	8.21	6.2	3.67	18.08
Sumas por columnas	14.87	15.44	8.27	

Figura 2.3: Problema de redondeo de matrices.

Veamos como se transforma este problema en un problema de flujo factible, con cotas inferiores y superiores para los flujos, para resolverlo posteriormente como un PFM. Construimos la red de modo que exista un nodo i para cada fila, y un nodo j' para cada columna j. Añadimos todos los arcos (s, i) que representarán las sumas por filas, y todos los arcos (j', t) que representarán las sumas por columnas. Además, cada elemento d_{ij} vendrá representado por el arco (i, j'). Los límites superiores e inferiores de cada arco serán el redondeo a la baja y al alza del elemento que representan, respectivamente. Para el ejemplo particular que presentamos en la Figura 2.3 el grafo obtenido sería el representado en la Figura 2.4.

Así formulado vemos que se trata de un problema de flujo factible que podemos resolver como ya hemos visto a través de los problemas de flujo máximo. Cualquier solución a este problema será, por lo tanto, un redondeo consistente para la matriz dada.

2.3. Conjunto de corte de capacidad mínima

Dado un grafo G, un *corte* es una partición del conjunto de nodos N en dos partes, S y $\overline{S} = N - S$. Cada corte tiene asociado un subconjunto de aristas para las cuales los vértices iniciales pertenecen al conjunto S y los vértices finales pertenecen a su complementario, $\overline{S}$, es decir, a cada corte se le asocia el conjunto $(S, \overline{S}) = \{(i, j) \in A : i \in S,\ j \in \overline{S}\}$. Si los

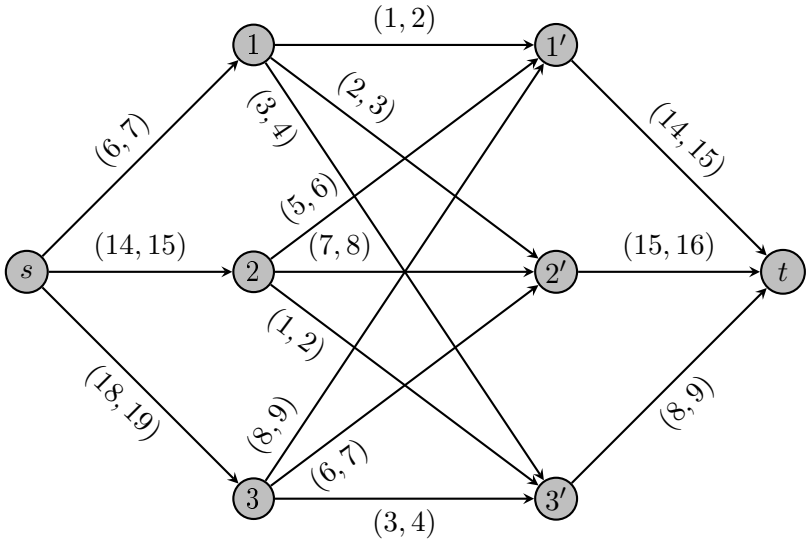
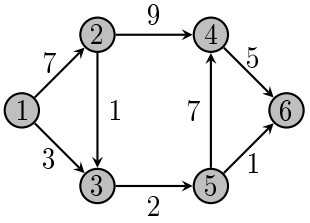


Figura 2.4: Ejemplo del problema de redondeo de matrices.

conjuntos S y $\overline{S}$ son tales que, para dos puntos distinguidos, por ejemplo 1 y n, $1 \in S$ y $n \in \overline{S}$ se dice que el corte es un *corte* $1 - n$.

Si tenemos un corte $(S, \overline{S})$ sobre el grafo G asociado a una red R, se define la capacidad del conjunto de corte como $U(S, \overline{S}) = \sum_{(i,j)\in(S,\overline{S})} u_{ij}$.



Tenemos muchas alternativas para formar conjuntos de corte $1 - 6$ para este grafo, entre ellas están los siguientes ejemplos, con sus respectivas capacidades:

- $S = \{1\}$, $\overline{S} = \{2, 3, 4, 5, 6\}$, $(S, \overline{S}) = \{(1, 2), (1, 3)\}$, $U(S, \overline{S}) = 10$.
- $S = \{1, 2, 3\}$, $\overline{S} = \{4, 5, 6\}$, $(S, \overline{S}) = \{(2, 4), (3, 5)\}$, $U(S, \overline{S}) = 11$.
- $S = \{1, 2, 3, 4, 5\}$, $\overline{S} = \{6\}$, $(S, \overline{S}) = \{(4, 6), (5, 6)\}$, $U(S, \overline{S}) = 6$.
- $S = \{1, 4, 5\}$, $\overline{S} = \{2, 3, 6\}$, $(S, \overline{S}) = \{(1, 2), (1, 3), (4, 6), (5, 6)\}$, $U(S, \overline{S}) = 16$.

Consideremos ahora un flujo factible $\boldsymbol{f}$ para el problema de flujo máximo y un conjunto de corte $(S, \overline{S})$ de modo que $s \in S$ y $t \in \overline{S}$. Por ser $\boldsymbol{f}$ un flujo factible debe verificar las

restricciones del problema, con lo que podemos escribir

$$F = \sum_{i \in S} \left[\sum_{\{j:(i,j) \in A\}} f_{ij} - \sum_{\{j:(j,i) \in A\}} f_{ji} \right].$$

Además, dados dos nodos p y q en S tales que $(p,q) \in A$, la variable x_{pq} que surge en el primer sumando cuando hacemos $i = p$ se cancela con la variable $-x_{pq}$ que surge en el segundo tras hacer $i = q$. Si p y q pertenecieran a $\overline{S}$ y $(p,q) \in A$, notar que la variable x_{pq} no aparecería en la expresión. Teniendo en cuenta estos argumentos, podemos simplificar y reescribir lo anterior de la siguiente forma:

$$F = \sum_{(i,j) \in (S,\overline{S})} f_{ij} - \sum_{(i,j) \in (\overline{S},S)} f_{ij}.$$

Teniendo en cuenta que todo flujo factible verifica $0 \leq f_{ij} \leq u_{ij}$, se deduce el siguiente resultado:

Lema 2.1. *Cualquier flujo factible del nodo* 1 *al nodo* n *es menor o igual que la capacidad de cualquier conjunto de corte* $1 - n$.

Este resultado implica que si tenemos un flujo factible cuyo valor es igual a la capacidad de un conjunto de corte, entonces es un flujo máximo y el conjunto de corte tiene capacidad mínima.

Definición 2.2. Dada una red R con capacidades limitadas en los arcos, el *problema del conjunto de corte de capacidad mínima* consiste en encontrar un conjunto de corte $(S, \overline{S})$ cuya capacidad sea la mínima entre todos los conjuntos de corte que se pueden definir en la red.

2.4. Teorema max-flow min-cut

Para finalizar este capítulo estudiaremos la formulación dual del problema de flujo máximo, la relacionaremos con los conjuntos de corte y, a partir de esta relación, deduciremos el teorema *max-flow min-cut*.
Recordemos la formulación del problema de flujo máximo:

$$\begin{aligned} &\text{Maximizar} \quad F \\ &\text{sujeto a:} \quad \sum_{\{j:(i,j) \in A\}} f_{ij} - \sum_{\{j:(j,i) \in A\}} f_{ji} = \begin{cases} F & \text{si } i = s \\ 0 & \text{si } i \in N - \{s,t\} \\ -F & \text{si } i = t \end{cases} \\ &\qquad\qquad 0 \leq f_{ij} \leq u_{ij} \quad \forall (i,j) \in A. \end{aligned}$$

Equivalentemente, teniendo en cuenta que denotamos por $\vec{e_i} \in \mathbb{R}^n$ el vector i-ésimo de la base canónica de $\mathbb{R}^n$ y que, dada la matriz de incidencia $\overline{\boldsymbol{B}}$, se tiene que $\bar{b}_{ik} = 1$ si i es el nodo inicial de la arista a_k, $\bar{b}_{ik} = -1$ si i es el nodo final del arco a_k y $\bar{b}_{ik} = 0$ en otro caso.

$$\begin{array}{lll} \text{Maximizar} & F & \\ \text{sujeto a:} & (\vec{e_n} - \vec{e_1})F + \overline{\boldsymbol{B}}\boldsymbol{f} = \boldsymbol{0} & n \text{ restricciones} \\ & \boldsymbol{f} \leq \boldsymbol{u} & m \text{ restricciones} \\ & \boldsymbol{f} \geq \boldsymbol{0} & m \text{ restricciones.} \end{array}$$

Nótese que para cada fila i de $\overline{\boldsymbol{B}}$, $\overline{\boldsymbol{B}}\boldsymbol{f}$ me da $\sum f_{ij}$ para los arcos que empiezan en i y $\sum -f_{ji}$ para los arcos que terminan en i.

Por tanto, la matriz de restricciones del problema viene dada por:

$$\boldsymbol{A} = \begin{pmatrix} -1 & \\ 0 & \\ \vdots & \overline{\boldsymbol{B}}_{n\times m} \\ 1 & \\ 0 & \\ \vdots & \boldsymbol{I}_{m\times m} \\ 0 & \end{pmatrix}_{(n+m)\times(m+1).}$$

La primera columna de $\boldsymbol{A}$ hace referencia al arco de vuelta, que va desde el nodo n al nodo 1, mientras que el resto de columnas son las equivalentes a los arcos de la red original. Cada columna de la matriz implicará una restricción en el problema dual, luego tendremos $\pi_1, \ldots, \pi_n$ variables duales asociadas a las restricciones de conservación de flujo y variables δ_{ij} asociadas a las restricciones de capacidad de los arcos.

Así, teniendo en cuenta las equivalencias dadas en la Figura 1.3 y que nuestro problema primal es el problema de maximización, el dual resulta:

$$\begin{array}{lll} \text{Minimizar} & \displaystyle\sum_{i=1}^{n} \pi_i 0 + \sum_{i=1}^{n}\sum_{j=1}^{n} u_{ij}\delta_{ij} & \\ \text{sujeto a:} & \pi_n - \pi_1 = 1 & \\ & \pi_i - \pi_j + \delta ij \geq 0 & \forall (i,j) \in A \\ & \pi \in \mathbb{R}^n & \text{(no restringido)} \\ & \delta_{ij} \geq 0. & \end{array}$$

Interpretación informal del dual:

- Nuestro objetivo es separar los nodos 1 y n "rompiendo" la red lo mínimo posible.

- δ_{ij}: ¿Qué porcentaje elimino del arco (i, j)?

Por el teorema de dualidad débil, sabemos que el valor de la función objetivo sobre cualquier solución factible para el problema de minimización será mayor o igual que el valor de la función objetivo del problema de maximización.

Consideramos ahora un corte $(S, \overline{S})$ separando el nodo 1 y el nodo n. Si definimos

$$\pi_i = \begin{cases} 0 & i \in S \\ 1 & i \in \overline{S} \end{cases} \quad \text{y} \quad \delta_{ij} = \begin{cases} 1 & (i,j) \in (S, \overline{S}) \\ 0 & \text{en otro caso,} \end{cases}$$

se puede ver que esta solución siempre es factible para el dual y la función objetivo del mismo coincide con la capacidad del conjunto de corte $(S, \overline{S})$.

Por tanto, dado que todo conjunto de corte separando la fuente y el sumidero se corresponde con una solución factible para el problema dual, el siguiente resultado es una consecuencia inmediata del teorema de dualidad débil:

Proposición 2.3. *El valor de cualquier flujo factible entre el nodo fuente y el sumidero es menor o igual a la capacidad de cualquier conjunto de corte separando esos dos nodos.*

Además, el algoritmo de trayectorias aumentadas que veremos en el Capítulo 4 construye un flujo factible y un conjunto de corte tales que el valor del primero coincide con la capacidad del conjunto de corte, lo que implica, por la proposición anterior, que ambas soluciones son óptimas. Este resultado se conoce como el Teorema max-flow min-cut que presentamos a continuación.

Teorema 2.4 (Teorema max-flow min-cut)**.** *La capacidad del conjunto de corte de capacidad mínima coincide con el flujo máximo en la red.*

Como consecuencia del resultado anterior tenemos que, esencialmente, resolver el problema del conjunto de corte de capacidad mínima es equivalente a resolver el dual del problema de flujo máximo.

Capítulo 3

Conceptos adicionales de optimización en redes

Este capítulo nos servirá de transición antes de comenzar a tratar algunos algoritmos de resolución para el problema de flujo máximo. En él, hablaremos de conceptos ya vistos en la asignatura de Programación Lineal y Entera, por lo que seguiremos los apuntes de Julio González Díaz (2018-2019). Entre estos conceptos podemos encontrar una clasificación para los algoritmos (Sección 3.1), el conocido problema del camino más corto (Sección 3.4), o una pequeña introducción a la teoría de complejidad computacional (Sección 3.2) la cual nos permitirá comparar los algoritmos que desarrollaremos para el PFM según su eficiencia. El resto del capítulo lo utilizaremos para desarrollar conceptos nuevos y los algoritmos de búsqueda, ambos obtenidos de Ahuja et al. (1993), que serán utilizados continuamente para facilitar el diseño de nuestros algoritmos.

3.1. Tipos de algoritmos

Un *algoritmo* es un procedimiento, paso a paso, diseñado para resolver un problema dado. De entre todos los algoritmos posibles para un mismo tipo de problema, debemos identificar aquel que será el más "eficiente".

Dadas las diferentes clasificaciones que existen para los algoritmos, pasamos a mencionar solamente un par de ellas.

Un algoritmo se dice *determinístico* si es "predecible", es decir, si dado un problema todas las ejecuciones del algoritmo producen el mismo resultado final y los mismos resultados intermedios. En los algoritmos *no determinísticos*, sin embargo, se introduce algún tipo de aleatoriedad en el proceso de búsqueda de la solución. Por ejemplo, un algoritmo de tipo no determinístico sería, dentro de un problema de optimización, generar un conjunto

de puntos aleatoriamente y escoger, de entre los factibles, el mejor. Aunque es posible que coincida la solución final, el conjunto de puntos generados no tiene por qué ser el mismo, lo que haría no coincidir los resultados intermedios.

Haciendo referencia a la "precisión" del algoritmo, podemos clasificarlo en estos tres grandes grupos:

- Algoritmos exactos: En este grupo se incluyen todos aquellos algoritmos que siempre devuelven la solución exacta del problema.
- Algoritmos aproximados: En este otro grupo se encuentran aquellos algoritmos cuyas soluciones estarán dentro de un determinado porcentaje del óptimo.
  Un algoritmo λ-aproximado devolverá una solución x tal que[1]:

$$\begin{cases} \mathrm{OPT} \leq c(x) \leq \lambda \cdot \mathrm{OPT} & \text{si } \lambda > 1, \\ \qquad\qquad \text{y} \\ \lambda \cdot \mathrm{OPT} \leq c(x) \leq \mathrm{OPT} & \text{si } \lambda < 1. \end{cases}$$

- Algoritmos heurísticos: Este tipo de algoritmos devuelven soluciones sin ninguna garantía de optimalidad. Pese a esto, los tiempos de ejecución para este grupo de algoritmos es mucho menor.

El interés general, claramente, estará en utilizar algoritmos exactos para la resolución de los problemas. Sin embargo, estos algoritmos pueden llegar a ser muy lentos, lo que nos lleva a echar mano de algoritmos aproximados, o, incluso, de algoritmos heurísticos.

3.2. Complejidad computacional

La teoría de complejidad computacional es la encargada de estudiar la efectividad de los diferentes algoritmos. Para ello, se centra en acotar el número de operaciones que necesitará el algoritmo en función de unos datos arbitrarios que especificarían el problema.

3.2.1. Diferentes medidas de complejidad

El número de pasos requeridos por un algoritmo es la suma de todos sus pasos intermedios, entre los cuales se encuentran los pasos de asignación, pasos de cálculos aritméticos y pasos de evaluaciones lógicas.

Habitualmente el rendimiento de un algoritmo se suele medir mediante alguno de los siguientes enfoques:

[1] Se escogerá $\lambda > 1$ o $\lambda < 1$ en función de si es un problema de minimización o de maximización, respectivamente.

1. El análisis empírico: El objetivo consiste en estimar como se comporta el algoritmo en la práctica ejecutándolo con distintos ejemplos del problema.

   Desventajas de este enfoque: El rendimiento del algoritmo depende del lenguaje de programación empleado, del compilador, del ordenador utilizado para la ejecución, así como de la habilidad del propio programador. Es muy costoso realizar un análisis de este tipo, además, a menudo los resultados son inconclusos puesto que el resultado puede depender del problema elegido para los test.

2. Análisis del caso promedio: El objetivo se basa en estimar el número medio de pasos que necesitará el algoritmo para devolver una solución. Se suele escoger una distribución de probabilidad sobre los posibles problemas y, mediante el uso de técnicas estadísticas, se derivan los tiempos de ejecución asintóticos del algoritmo.

   Desventajas de este enfoque: El análisis se ve condicionado por la distribución de probabilidad seleccionada. Habitualmente es complicado determinar la distribución que mejor se adapta a los problemas que surgen en la práctica. Además, se trata de un análisis muy complicado de llevar a cabo para algoritmos complejos.

3. Análisis del peor caso: Este enfoque proporciona cotas superiores para el número de operaciones que va a necesitar un algoritmo al resolver cualquier problema dentro de la clase en estudio.

   Ventajas de este enfoque: El análisis es independiente del entorno computacional, es más sencillo de realizar y proporciona un tiempo de ejecución máximo para el algoritmo. Además, es capaz de comparar de forma inequívoca dos algoritmos dados.

   Desventajas de este enfoque: Puede llegar a clasificar como malo un algoritmo que solo funciona mal en ejemplos patológicos, a pesar de que estos aparecen raramente en la práctica. Esto es lo que ocurría con el método símplex, que tiene un tiempo exponencial según el peor caso, pero en los casos prácticos es polinomial.

3.2.2. Notación O

Se dice que un algoritmo tiene una velocidad $O(f(n))$ si existen dos constantes k y n_0 tales que, para un problema de tamaño $n \geq n_0$, el número de operaciones efectuadas por el algoritmo para resolverlo es menor o igual que $kf(n)$. Un algoritmo se dice *polinomial* si tiene una velocidad $O(f(n))$ donde f es un polinomio.
Por ejemplo, si tenemos un algoritmo cuyo tiempo de ejecución es $1000n^2+0{,}01n^3$, tomando $n \geq 10^4$, n^3 es mayor que $1000n^2 + 0{,}01n^3$, con lo que la complejidad del algoritmo será $O(n^3)$, y, por tanto, el algoritmo es polinomial.

En la práctica nos interesará trabajar con algoritmos, a lo sumo, de tiempo polinomial puesto que los algoritmos no polinomiales pueden llegar a ser muy poco útiles en el estudio de problemas grandes.

3.3. Algoritmos de búsqueda

Los algoritmos de búsqueda están diseñados para encontrar entre todos los nodos de una red aquellos que cumplan una cierta propiedad impuesta. Este tipo de algoritmos son utilizados habitualmente en el diseño de algoritmos más complejos, como pueden ser los algoritmos de resolución para el problema de flujo máximo.

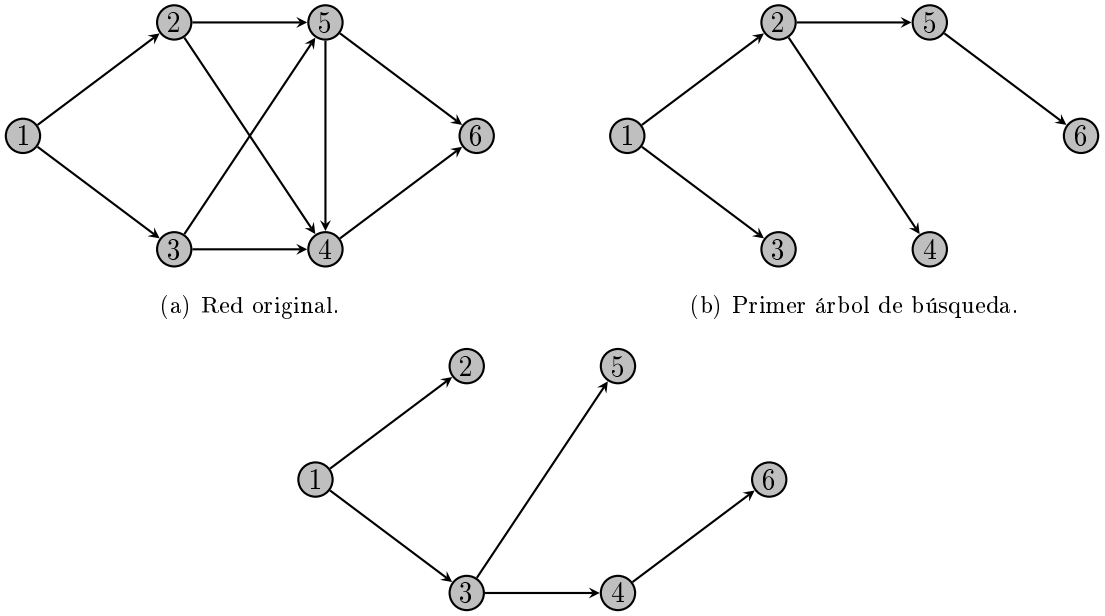
Entre las aplicaciones más comunes de un algoritmo de búsqueda se incluyen el encontrar todos los nodos de la red a los que se puede llegar a través de un camino dirigido desde un nodo prefijado, identificar todos los nodos que contienen un camino dirigido hasta un nodo t prefijado,...

Supongamos que queremos identificar todos los nodos que se pueden alcanzar partiendo de un nodo fuente, s, en una red dirigida $R = (G, (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}))$. Para ello, se definen dos conjuntos, el conjunto de nodos *marcados* y el de nodos *no marcados*. El conjunto de nodos marcados estará formado por aquellos nodos que son alcanzados desde la fuente, y, por tanto, será nuestra solución al problema. Por otra parte, se dirá que un arco (i, j) es *admisible* siempre y cuando el nodo i pertenezca al conjunto de los nodos marcados y el nodo j no; en otro caso, se dirá que el arco es *inadmisible*. Además, cuando el arco (i, j) es admisible, el nodo j se puede introducir en el conjunto de los nodos marcados, pues tenemos un camino dirigido de s a j formado por el camino de s a i (que existe por ser i un nodo marcado) más el arco (i, j). En este caso se dice que j tiene como *predecesor* al nodo i, pred(j)=i.

Para analizar los arcos admisibles de la red, el algoritmo parte de un nodo marcado i y examina todos los arcos que salen de i para ver si son admisibles o no. Para ello, hace uso de la lista de adyacencia del nodo, $A(i)$. Dados dos nodos j, k, con $j < k$, y $(i, j), (i, k) \in A(i)$, se impone que el algoritmo analice primero el arco (i, j) y después el arco (i, k). Una vez que se hayan analizado todos los arcos, decimos que i ya no posee arcos admisibles y, entonces, el algoritmo salta a otro de los nodos marcados para analizar, de la misma forma, su lista de adyacencia.

Una vez que el algoritmo ha finalizado, es decir, una vez que ya no queden nodos marcados sin analizar sus listas de adyacencia, haciendo uso de los predecesores se puede crear un árbol conocido como el *árbol de búsqueda*. Tomando como ejemplo la red dada en la Figura 3.1(a), podemos obtener los árboles de búsqueda 3.1(b) y 3.1(c). Como vemos,

existen múltiples árboles de búsqueda para una misma red dependiendo del orden en el cual vayamos seleccionando los nodos para analizar. Pese a ello, notar que el conjunto de nodos marcados siempre será el mismo.



(a) Red original.

(b) Primer árbol de búsqueda.

(c) Segundo árbol de búsqueda.

Figura 3.1: Ejemplos de árboles de búsqueda para una red

Se puede ver fácilmente que el algoritmo de búsqueda proporciona una solución en un tiempo máximo de $O(m)$, donde $m = |A|$ para la red $R = (G, (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}))$. Esto es así pues el algoritmo analiza cada arco, a lo sumo, una única vez.

3.4. Problema del camino más corto

No podemos continuar con el desarrollo del trabajo sin mencionar el problema del camino más corto. Este problema resulta de gran interés a la hora de estudiar problemas de optimización en redes pues, al igual que ocurría con los algoritmos de búsqueda, aparece frecuentemente como subproblema de otros más complejos.

El *problema del camino más corto* en una red dirigida $R = (G, (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}))$ con vector de costes $\boldsymbol{c} \in \mathbb{R}^n$ consiste en encontrar el camino entre los nodos 1 y n de la red cuyo coste sea mínimo. El coste de un camino no es más que la suma de los costes de los arcos que lo forman.

Se puede ver fácilmente que el problema del camino más corto no deja de ser un caso

particular del PFCM. Para ello, estableceremos cotas inferiores y superiores en los arcos de 0 y 1, respectivamente. Además, se tienen dos flujos externos fijos, $e_1 = 1$ y $e_n = -1$, los cuales se pueden eliminar, como ya hemos visto en el Capítulo 1, introduciendo un arco de vuelta con coste 0. Ilustramos un ejemplo en la Figura 3.2.

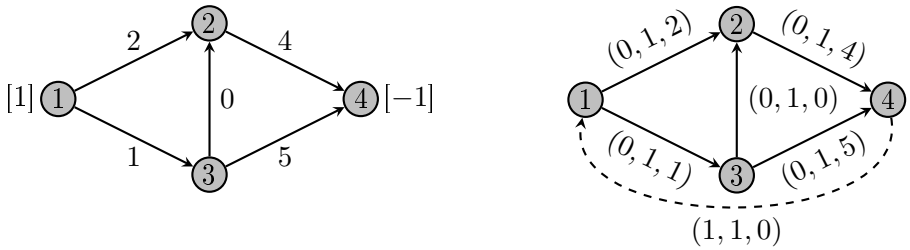


Figura 3.2: Problema del camino más corto expresado como PFCM.

Es importante observar que el problema del camino más corto es un problema entero, puesto que no podemos "dividirnos" cuando nos trasladamos de un nodo a otro. Sin embargo, esta restricción podremos obviarla al tratarse de un PFCM.

Observación 3.1. El problema del camino más corto solo es posible representarlo como un PFCM cuando no existen ciclos de longitud negativa. Esto se debe a que, a pesar de que la solución resultante puede no ser un camino, si se estaría verificando la condición de conservación de flujo. Para la red representada en la Figura 3.3 se obtendrían unos flujos, resultados de resolver el PFCM, $f_{12} = 0$, $f_{13} = 1$, $f_{24} = 1$, $f_{31} = 1$, $f_{42} = 1$, con un coste asociado de -2. Sin embargo, observamos que el camino más corto entre los nodos 1 y 3 vendría dado por el arco $(1, 3)$ y tendría un coste 2.

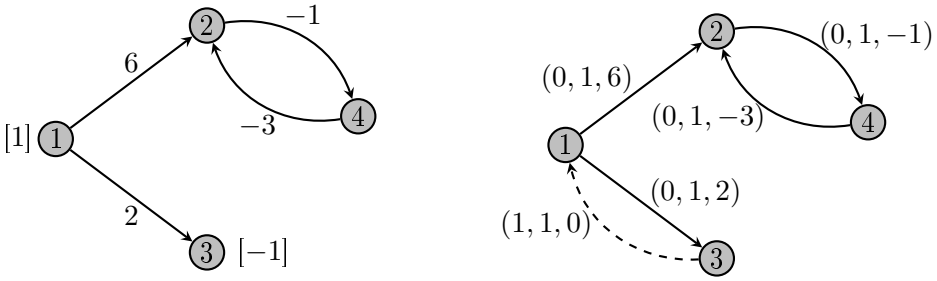


Figura 3.3: PFCM con costes negativos.

3.5. Redes residuales

El concepto de *red residual* se basa en una idea bastante intuitiva. Supongamos que tenemos una red dirigida $R = (G, (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}))$ en la cual el arco (i, j), con capacidad u_{ij}, lleva f^o_{ij} unidades de flujo. Está claro que todavía se pueden enviar $u_{ij} - f^o_{ij}$ unidades de flujo adicional desde el nodo i al nodo j por dicho arco. Asimismo, podemos enviar f^o_{ij} unidades de flujo del nodo j al nodo i por el arco (i, j), lo que cancelaría el flujo existente en el arco. Mandar una unidad de flujo del nodo i al nodo j a través del arco (i, j) aumenta en c_{ij} unidades el coste del problema, mientras que si, utilizando el mismo arco, se envía una unidad de flujo de j a i, el coste disminuye en c_{ij} unidades puesto que estamos cancelando el envío de esa unidad de flujo.

Basándonos en lo anterior, se define la red residual respecto a un flujo $\boldsymbol{f}^o$, $R(\boldsymbol{f}^o)$, como sigue. Cada arco (i, j) en la red original se sustituye por dos arcos, (i, j) y (j, i). El arco (i, j) tendrá un coste c_{ij} y una capacidad residual $r_{ij} = u_{ij} - f^o_{ij}$, y el arco (j, i) tendrá asociados un coste $-c_{ij}$ y una capacidad residual $r_{ji} = f^o_{ij}$ (ver Figura 3.4).

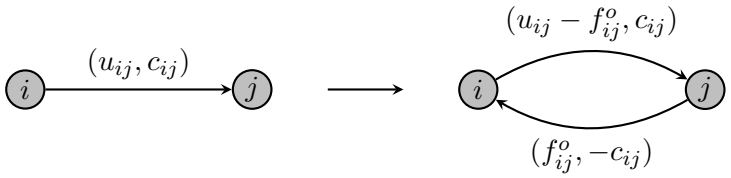


Figura 3.4: Construyendo la red residual $R(\boldsymbol{f}^o)$.

Nótese que si la red original R posee también el arco (j, i), en la red residual pueden aparecer dos arcos paralelos del nodo i al nodo j con costes y capacidades residuales diferentes, y/o dos arcos paralelos del nodo j al nodo i en las mismas condiciones. En estas circunstancias la referencia al arco (i, j), o al arco (j, i), será ambigua y no definirá un único coste ni una única capacidad residual. Particularmente, para el problema del flujo máximo esto no supone ningún problema puesto que todos los arcos paralelos en la red residual tendrán coste cero, por lo que podremos fusionarlos en un solo arco cuya capacidad residual será igual a la suma de las capacidades residuales de los otros dos arcos.

Veremos que cada flujo $\boldsymbol{f}$ en la red original R se corresponde con un flujo $\boldsymbol{f}'$ en la red residual asociada al flujo $\boldsymbol{f}^o$. Tomamos un flujo $\boldsymbol{f}' \geq 0$ que debe verificar

$$f'_{ij} - f'_{ji} = f_{ij} - f^o_{ij} \tag{3.1}$$

y

$$f'_{ij} f'_{ji} = 0. \tag{3.2}$$

En el caso de que $f_{ij} \geq f^o_{ij}$, se toman $f'_{ij} = f_{ij} - f^o_{ij}$ y $f'_{ji} = 0$. Nótese que si $f_{ij} \leq u_{ij}$, entonces f'_{ij} satisface las condiciones de capacidad residuales, pues $f'_{ij} \leq u_{ij} - f^o_{ij} = r_{ij}$. Si, por el contrario, $f_{ij} < f^o_{ij}$, tomamos $f'_{ij} = 0$ y $f'_{ji} = f^o_{ij} - f_{ij}$, con lo que se tiene que $0 \leq f'_{ji} \leq f^o_{ij} = r_{ji}$, con lo que también se estarían satisfaciendo las condiciones de capacidades residuales.

3.6. Etiquetas de distancia

Sea $R = (G, (\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{c}))$ una red con flujo con dos nodos distinguidos, la fuente, s, y el sumidero, t. Se puede definir una aplicación, denominada la *función distancia*, sobre el conjunto de nodos de forma que tome valores sobre los enteros no negativos, es decir, $d : N \longrightarrow \mathbb{Z}^+ \cup \{0\}$. Decimos que la función es *válida* con respecto a un flujo $\boldsymbol{f}$ si verifica las siguientes condiciones:

$$\begin{aligned} &d(t) = 0; \\ &d(i) \leq d(j) + 1 \quad \text{para todo } (i,j) \text{ en la red residual } R(\boldsymbol{f}). \end{aligned}$$

Estas condiciones son conocidas como las *condiciones de validación* y $d(i)$ se dice que es la *etiqueta de distancia* del nodo i.

Proposición 3.2. *Si las etiquetas de distancia son válidas, la etiqueta $d(i)$ es una cota inferior para la longitud del camino más corto, suponiendo que todos los costes son unitarios, entre el nodo i y el nodo t en la red residual.*

Demostración. Consideramos un camino cualquiera de longitud k entre el nodo i y el nodo t, $i_1 - i_2 - i_3 - \ldots - i_k - i_{k+1}$, donde $i_1 = i$ y $i_{k+1} = t$. Como las etiquetas de distancia son válidas se verifican las condiciones de validación, con lo que

$$\begin{aligned} d(i_k) &\leq d(i_{k+1}) + 1 = d(t) + 1 = 1 \\ d(i_{k-1}) &\leq d(i_k) + 1 \leq 2 \\ d(i_{k-2}) &\leq d(i_{k-1}) + 1 \leq 3 \\ &\vdots \\ d(i_2) &\leq d(i_3) + 1 \leq k - 1 \\ d(i) = d(i_1) &\leq d(i_2) + 1 \leq k. \end{aligned}$$

Puesto que el camino escogido es arbitrario, $d(i)$ es una cota inferior para las longitudes de todos los caminos entre el nodo i y el nodo t, en particular, es una cota inferior para el camino de longitud mínima entre dichos nodos. □

Proposición 3.3. *Si $d(s) \geq n = |N|$, y las etiquetas son válidas, la red residual no contiene ningún camino dirigido entre la fuente y el sumidero.*

Demostración. Todo camino dirigido entre la fuente y el sumidero, a lo sumo, recorre todos los nodos antes de alcanzar el sumidero, con lo que no puede contener más de $(n-1)$ arcos. Por la proposición anterior, $d(s)$ es una cota inferior de todos los caminos dirigidos entre s y t, luego debe ser menor o igual que $n-1$. Entonces, si $d(s) \geq n$, es imposible que exista un camino dirigido con estas características. □

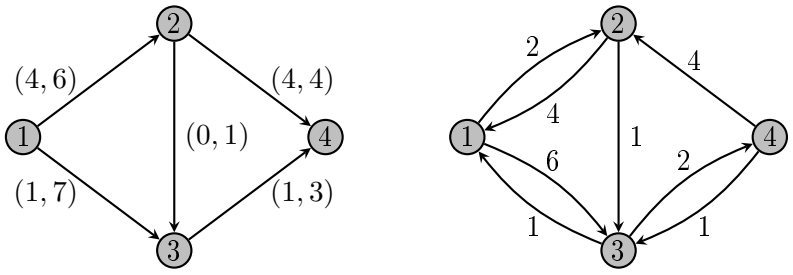
Diremos que un arco (i, j) en la red residual es *admisible* dadas unas etiquetas de distancia válidas, si satisface $d(i) = d(j) + 1$. Además, a aquellos caminos de la fuente al sumidero formados por este tipo de arcos, los denominaremos *caminos admisibles*.

Capítulo 4

Algoritmo de trayectorias aumentadas

Dedicaremos este capítulo a desarrollar uno de los algoritmos más simples e intuitivos para resolver el problema del flujo máximo, el *algoritmo de trayectorias aumentadas*. Para ello, seguimos tomando como referencia el Capítulo 6 del libro Ahuja et al. (1993).

Consideramos una red R con un flujo $\boldsymbol{f}$ y su red residual asociada $R(\boldsymbol{f})$, por ejemplo, las representadas en la Figura 4.1. Dada un red residual $R(\boldsymbol{f})$, todo camino dirigido entre la fuente y el sumidero diremos que es una *trayectoria aumentada*. Dicha trayectoria tendrá una capacidad residual asociada, resultado de obtener el mínimo de las capacidades residuales entre todos los arcos que la componen.



(a) Red con flujo. (b) Red residual asociada.

Figura 4.1: Ejemplo de una red con flujo y su red residual asociada.

Para la red residual representada en la Figura 4.1(b), podemos seleccionar dos trayectorias aumentadas distintas desde el nodo 1 hasta el nodo 4, el camino $1-2-3-4$ y el camino $1-3-4$. En el caso de la primera trayectoria la capacidad residual sería $\delta_1 = \text{mín}\{2, 1, 2\} = 1$, y para la segunda, se tendría $\delta_2 = \text{mín}\{6, 2\} = 2$.

Hay que tener en cuenta que, como las capacidades residuales de los arcos de la red residual son siempre positivas, la capacidad residual de una trayectoria aumentada también será positiva. Es por ello que si existe una camino dirigido de la fuente al sumidero, podremos incrementar la cantidad de flujo que se puede enviar entre la fuente y el sumidero en δ unidades.

El trabajo del algoritmo de trayectorias aumentadas consiste en ir identificando los caminos dirigidos de s a t existentes en la red, con sus respectivas capacidades residuales, hasta que no existan más trayectorias de este tipo.

Algoritmo 1: Algoritmo de trayectorias aumentadas.

inicio
  $\boldsymbol{f} := \mathbf{0}$;
  mientras $R(\boldsymbol{f})$ *contenga un camino dirigido de* s *a* t **hacer**
    identificar una trayectoria aumentada P de s a t;
    $\delta := \text{mín}\{r_{ij} : (i,j) \in P\}$;
    aumentar δ unidades de flujo a lo largo de P y actualizar $R(\boldsymbol{f})$.
  fin
fin

El algoritmo de trayectorias aumentadas descrito de esta forma no queda totalmente determinado, por lo que su eficiencia dependerá, por ejemplo, de la forma en la que se identifiquen y se elijan las diferentes trayectorias aumentadas, como veremos más adelante.

4.1. Ejemplo de resolución

Procedemos a ilustrar, paso a paso, cómo se obtiene el flujo máximo del nodo 1 al nodo 7 para la siguiente red con flujos empleando el algoritmo de trayectorias aumentadas.

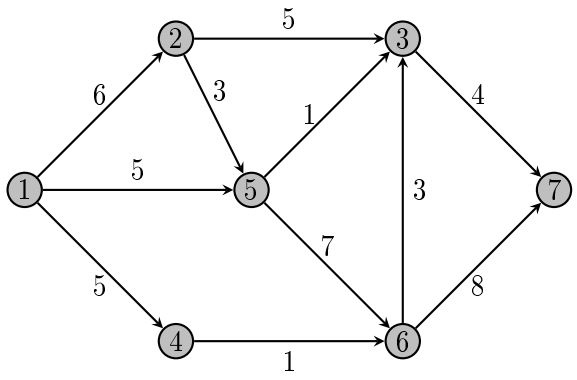


Figura 4.2: Ilustración del algoritmo para el PFM.

PASO 1: Inicializamos el vector de flujo $\boldsymbol{f} = \mathbf{0}$, con flujo asociado $F = 0$.

PASO 2 (Iteración 1): Con el vector de flujos $\boldsymbol{f} = \mathbf{0}$, $R = R(\boldsymbol{f})$, con lo que $u_{ij} = r_{ij}$. Seleccionamos un camino dirigido de s a t, por ejemplo, el camino $P_1 = \{(1,2),(2,3),(3,7)\}$. Dicha trayectoria tiene una capacidad residual de $\delta = \text{mín}\{6,5,4\} = 4$ unidades. Enviamos, por tanto, δ unidades de flujo a lo largo de P_1, con lo cual $F = 4$, y actualizamos $R(\boldsymbol{f})$:

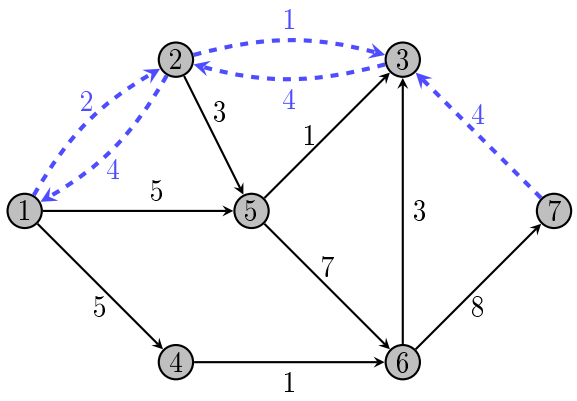


Figura 4.3: Iteración 1.

PASO 3 (Iteración 2): Puesto que todavía existen caminos dirigidos entre la fuente y el sumidero, el algoritmo continúa. Escogemos otra de las trayectorias aumentadas, en este caso, $P_2 = \{(1,2),(2,5),(5,6),(6,7)\}$ y $\delta = \text{mín}\{2,3,7,8\} = 2$. Actualizamos el valor del flujo $F = 4 + 2 = 6$ y la red residual resultante tras enviar δ unidades de flujo a lo largo de P_2:

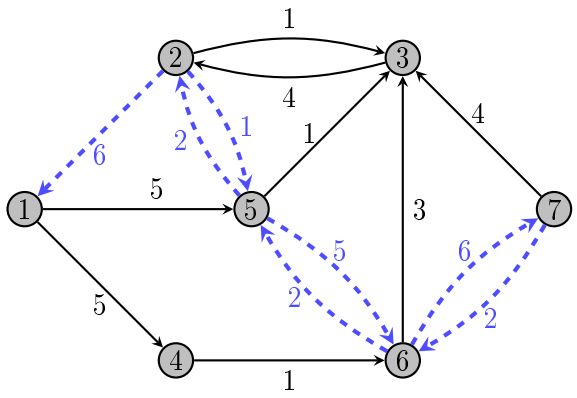


Figura 4.4: Iteración 2.

PASO 4 (Iteración 3): Observando la nueva red residual vemos que todavía existen trayectorias aumentadas, luego el algoritmo actúa de forma análoga a la iteración anterior. Identificamos un camino dirigido desde el nodo 1 al nodo 7, por ejemplo $P_3 = \{(1,4),(4,6),(6,7)\}$, cuyo valor residual es $\delta = \text{mín}\{5,1,6\} = 1$. Actualizamos nuestra red residual para los nuevos flujos y aumentamos el valor de $F = 6 + 1 = 7$:

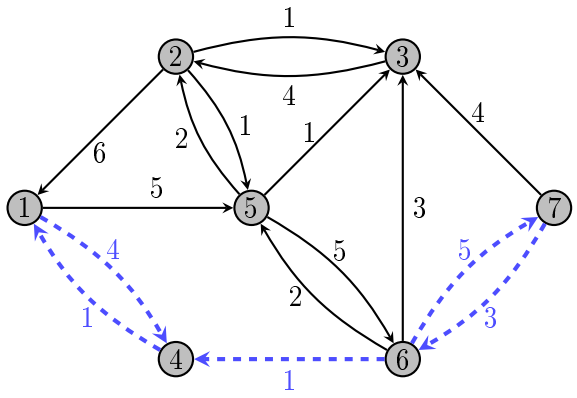


Figura 4.5: Iteración 3.

PASO 5 (Iteración 4): Observamos que $P_4 = \{(1,5),(5,6),(6,7)\}$ es una trayectoria aumentada con capacidad residual $\delta = \text{mín}\{5,5,5\} = 5$. Tenemos, por tanto, una actualización en el flujo $F = 7 + 5 = 12$ y una nueva red residual:

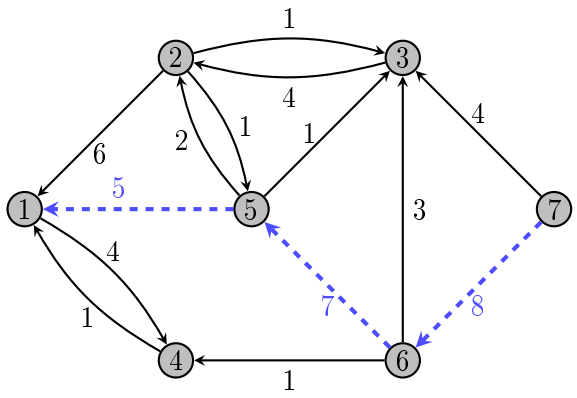


Figura 4.6: Iteración 4

PASO 6 (Iteración 5): No existe ningún camino dirigido entre el nodo 1 y el nodo 7 en la red residual, por lo que el algoritmo finaliza y el valor del flujo máximo de la red es $F = 12$. Dicho valor tiene asociadas las variables de flujo $f_{12} = 6,\ f_{15} = 5,\ f_{14} = 1,\ f_{23} = 4,\ f_{25} = 2,\ f_{37} = 4,\ f_{46} = 1,\ f_{56} = 7,\ f_{67} = 8$ y, en cualquier otro caso, $f_{ij} = 0$.

4.2. Relaciones entre la red original y la red residual

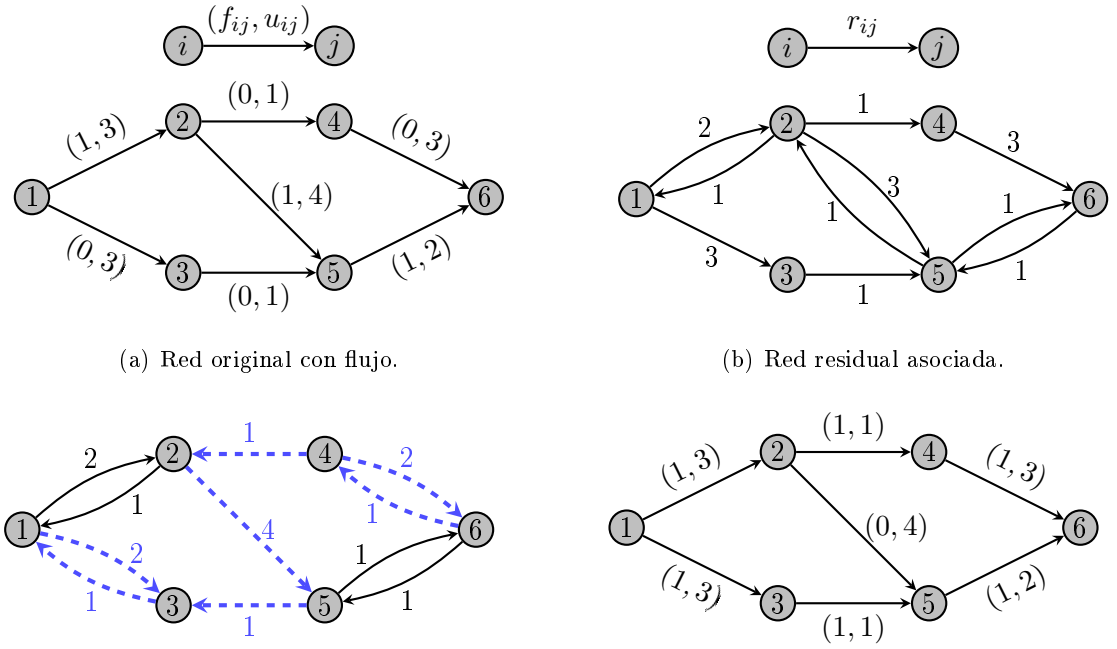
En el ejemplo anterior hemos visto que el algoritmo nos devuelve el valor del flujo máximo para la red. Sin embargo, para dar una solución completa, debemos proporcionar también las variables de flujo correspondientes a cada arco. Hay que tener en cuenta que cuando definimos el algoritmo de trayectorias aumentadas estamos siempre trabajando sobre la red residual, con lo que obtenemos los resultados en función de las capacidades residuales de caminos entre la fuente y el sumidero, de donde obtenemos posteriormente las variables de flujo. Sin embargo esta elección no es obligatoria, podríamos utilizar la red original y obtener directamente los flujos asociados a las aristas. Para ver cómo podemos pasar de una alternativa a otra, debemos entender las relaciones que existen entre los flujos asociados a las aristas en la red original y las capacidades residuales de los caminos en la red residual.

Una *trayectoria aumentada en la red original* es un camino entre la fuente y el sumidero donde la arista (i, j) se puede recorrer en el sentido natural si, para el flujo $\boldsymbol{f}$ actual, se tiene $f_{ij} < u_{ij}$, y en sentido inverso si $f_{ij} > 0$. Si la red original R contiene una trayectoria aumentada con respecto a un flujo $\boldsymbol{f}$, existe un camino dirigido de la fuente al sumidero en $R(\boldsymbol{f})$. De igual modo, si existe un camino dirigido en la red residual, entonces existe una trayectoria aumentada en la red original.

Supongamos que hemos actualizado las capacidades residuales. ¿Cómo afecta dicho

aumento a las variables de flujo? Sabemos que las capacidades residuales están definidas como $r_{ij} = u_{ij} - f_{ij} + f_{ji}$, luego si aumentamos δ unidades en el arco (i, j), podrían ocurrir cualquiera de los siguientes casos:

1. Un aumento de δ unidades en f_{ij} en la red original.
2. Una disminución de δ unidades en f_{ji} en la red original.
3. Una combinación lineal de ambas.



(a) Red original con flujo.

(b) Red residual asociada.

(c) Red residual actualizada después de mandar una unidad de flujo por el camino $1 - 3 - 5 - 2 - 4 - 6$.

(d) Vuelta a la red original.

Figura 4.7: Ilustración de cómo cambian las variables de flujo.

Para la red proporcionada en la Figura 4.7 si comparamos la red inicial (4.7(a)) con la solución obtenida tras enviar una unidad de flujo por el camino $1 - 3 - 5 - 2 - 4 - 6$ (4.7(d)), entonces observamos que los arcos $(1, 3), (3, 5), (2, 4)$ y $(4, 6)$ aumentaron en una unidad su flujo, mientras que el arco $(2, 5)$ disminuyó esta misma unidad.

Nos queda por ver cómo se transforman las capacidades residuales r_{ij} nuevamente en flujos f_{ij}. Tenemos que $r_{ij} = u_{ij} - f_{ij} + f_{ji}$, o, lo que es lo mismo, $f_{ij} - f_{ji} = u_{ij} - r_{ij}$. Obviamente, existen diferentes combinaciones de f_{ij} y f_{ji} que proporcionan un mismo resultado. Por ello, si tenemos $u_{ij} \geq r_{ij}$ tomaremos $f_{ij} = u_{ij} - r_{ij}$ y $f_{ji} = 0$. En otro caso, $f_{ij} = 0$ y $f_{ji} = r_{ij} - u_{ij}$.

Vamos a detallar este cálculo para obtener el flujo de los arcos después de enviar una unidad de flujo por el camino $1-3-5-2-4-6$, es decir, vamos a ver cómo transformamos la red 4.7(c) en la red 4.7(d). Primeramente, sacamos las capacidades de los arcos de la red original y las capacidades residuales de 4.7(c) no nulas:

$$\begin{array}{llllllll} u_{12}=3 & r_{12}=2 & u_{25}=5 & r_{25}=4 & u_{46}=3 & r_{46}=2 & r_{56}=1 \\ u_{13}=3 & r_{13}=2 & & r_{31}=1 & & r_{53}=1 & \\ u_{24}=1 & & u_{35}=1 & & u_{56}=2 & r_{56}=1 & \\ & r_{21}=1 & & r_{42}=1 & & r_{64}=1 & \end{array}$$

Tenemos que $u_{12} \geq r_{12}$, $u_{13} \geq r_{13}$, $u_{24} \geq r_{14} = 0$, $u_{25} \geq r_{25}$, $u_{35} \geq r_{35} = 0$, $u_{46} \geq r_{46}$ y $u_{56} \geq r_{56}$, luego, en estos casos, se tiene que $f_{ij} = u_{ij} - r_i$ y $f_{ji} = 0$, es decir,

$$\begin{array}{llllll} f_{12}=3-2=1; & f_{13}=3-2=1; & f_{24}=1-0=1; & f_{25}=5-4=1; \\ f_{35}=1-0=1; & f_{46}=3-2=1; & f_{56}=2-1=1; & \end{array}$$

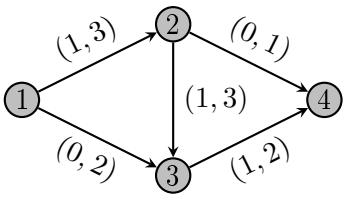
En otro caso, $u_{ij} < r_{ij}$, tendremos que $f_{ij} = 0$ y $f_{ji} = r_{ij} - u_{ij}$:

$$\begin{array}{lll} f_{12}=1-0=1; & f_{24}=1-0=1; & f_{56}=1-0=1; \\ f_{13}=1-0=1; & f_{46}=1-0=1; & \end{array}$$

Observamos que hay variables de flujo que se repiten en los dos casos, pero su valor siempre va a coincidir.

4.3. Efectos del aumento en la descomposición de flujo

Dedicaremos esta sección a mejorar la interpretación que se tiene del algoritmo de trayectorias aumentadas. Para ello, ilustraremos el efecto que supone un aumento en la descomposición de flujo sobre la siguiente red con flujos:



Estos flujos iniciales pueden descomponerse como se representa en la Figura 4.8(a). Ahora bien, si consideramos el camino $1-3-2-4$ sobre la red residual asociada y enviamos una unidad de flujo a través del mismo, obtenemos la descomposición de flujo dada en la

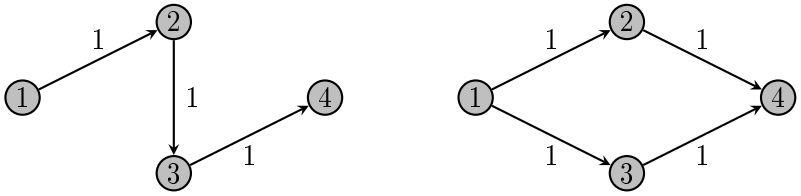


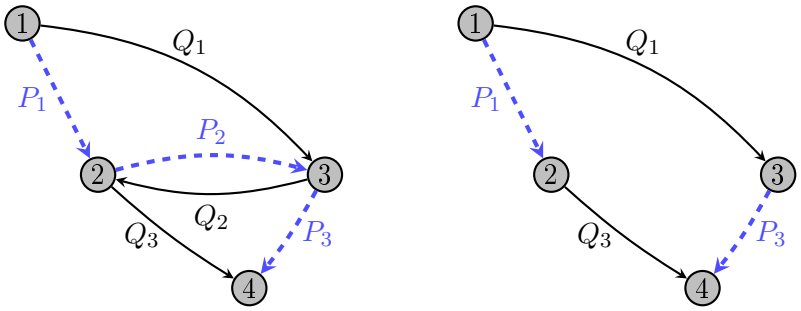
Figura 4.8: Descomposiciones de flujo individuales.

Figura 4.8(b). Vemos que, a pesar de estar enviando flujo por el camino $1-3-2-4$, dicho camino no aparece en la nueva descomposición de flujo.

La argumentación de por qué pasa esto es bastante sencilla de ilustrar. Primeramente, el camino $1-2-3-4$ podemos descomponerlo en tres segmentos: el primero sería el camino hasta el nodo 2, el segundo sería el arco $(2,3)$ recorrido en sentido natural, y el camino hasta el nodo 4. De igual forma, el camino $1-3-2-4$ se descompone como el camino hasta el nodo 3, el arco $(2,3)$ recorrido en sentido inverso, y el camino hasta el nodo 4 (ver Figura 4.9(a)).

El aumento por el camino $1-3-2-4$ podemos verlo como la unión del primer segmento del camino $1-2-3-4$ con el último segmento del aumento, la unión del último segmento del camino $1-2-3-4$ con el primer segmento del aumento, y la cancelación de flujo en los arcos $(2,3)$ y $(3,2)$, pues la unión de ambos se anula. Estas uniones dan lugar a una descomposición neta del flujo en la cual no aparece el arco $(2,3)$ (ver Figura 4.9(b)).

En general, cada aumento no es más que "pegar" segmentos de las descomposiciones de flujo para obtener una nueva descomposición de flujo unificada.



(a) Representación de los dos aumentos.

(b) Efecto neto de los aumentos.

Figura 4.9: Descomposición de flujo unificada.

4.4. Algoritmo de etiquetado

En la sección anterior no hemos mencionado ningún procedimiento concreto para identificar las trayectorias aumentadas ni se ha comprobado que, cuando el algoritmo termina en un número finito de operaciones, la solución obtenida es efectivamente un flujo máximo. Por ello, esta sección está pensada para abordar dichas cuestiones sobre el *algoritmo de etiquetado*, que no es más que una implementación específica del algoritmo genérico de trayectorias aumentadas que acabamos de ver.

El algoritmo de etiquetado se apoya en las técnicas de búsqueda vistas en la Sección 3.3 para identificar, en la red residual, las diferentes trayectorias aumentadas.

La idea general del algoritmo consiste en, partiendo del nodo fuente, s, desplazarse a través de caminos dirigidos para identificar aquellos nodos que pueden ser alcanzados desde la fuente. Es por ello que define dos grupos: el conjunto de nodos *etiquetados* y el conjunto de nodos *no etiquetados*. Los nodos etiquetados son aquellos que se han alcanzado a través de la búsqueda realizada por el algoritmo, es decir, aquellos para los que existe un camino dirigido desde la fuente en la red residual. Por contra, el conjunto de nodos no etiquetados está formado por aquellos nodos a los que el algoritmo no ha llegado todavía. El proceso realizado por el algoritmo es muy simple: de manera iterativa, el algoritmo selecciona un nodo etiquetado y analiza su lista de adyacencia en la red residual con el fin de alcanzar, y por tanto etiquetar, nuevos nodos. Cuando el algoritmo llega a etiquetar el sumidero, se envía la mayor cantidad de flujo posible a lo largo del camino empleado para llegar al nodo t, el cual se reconstruye gracias al uso de los predecesores, y se actualizan las capacidades residuales. Seguidamente, desetiqueta todos los nodos y vuelve a empezar el proceso para la nueva red residual. El algoritmo termina cuando ya se han analizado todos los nodos etiquetados y el sumidero se mantiene sin etiquetar.

Ilustración del algoritmo

Antes de continuar vamos a ver como el algoritmo de etiquetado resolvería, paso a paso, el problema del flujo máximo para un ejemplo particular.

Consideramos la red residual $R(\boldsymbol{f})$ de la Figura 4.10 asociada al flujo $\boldsymbol{f} = \mathbf{0}$.

Algoritmo 2: Algoritmo de etiquetado.

inicio
- Etiquetar el nodo t;
- **mientras** *t esté etiquetado* **hacer**
  - **inicio**
    - Desetiquetar todos los nodos;
    - Establecer pred(j)=0 para todo $j \in N$;
    - Etiquetar s y fijar LISTA:=$\{s\}$;
    - **mientras** *LISTA$\neq \emptyset$ y t no esté etiquetado* **hacer**
      - **inicio**
        - Seleccionar y retirar el nodo i de LISTA con índice más pequeño;
        - **para** *cada arco (i, j) en $R(\boldsymbol{f})$ con i como nodo de inicio* **hacer**
          - **si** *el nodo j no está etiquetado* **entonces**
            - pred(j)=i;
            - Etiquetar el nodo j;
            - Añadir j a LISTA;
          - **fin**
        - **fin**
      - **fin**
    - **si** *t está etiquetado* **entonces**
      - Ejecutar *aumento (ver Algoritmo 3)*
    - **fin**
  - **fin**
- **fin**

fin

Algoritmo 3: Aumento.

inicio
- Haciendo uso de los predecesores reconstruir, desde el sumidero a la fuente, la trayectoria aumentada P;
- $\delta = \text{mín}\{r_{ij} : (i, j) \in P\}$;
- Aumentar δ unidades de flujo a lo largo de P y actualizar las capacidades residuales;

fin

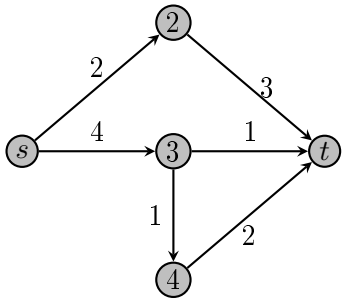


Figura 4.10: Ilustración algoritmo de etiquetado.

PRIMERA ITERACIÓN:

- Etiquetamos el nodo t;

- Desetiquetamos todos los nodos;
  pred(j)=0 $\forall j \in N$;
  Nodos etiquetados=$\{s\}$;
  LISTA=$\{s\}$;

- Eliminamos un nodo de LISTA, en este caso s por ser el único. LISTA=$\emptyset$;
  Los arcos $(s,2),(s,3) \in A(s)$ y los nodos 2 y 3 no están etiquetados;
  Nodos etiquetados=$\{s, 2, 3\}$;
  pred(2)=pred(3)=s;
  LISTA=$\{2, 3\}$;

- Eliminamos el nodo 2 de LISTA por ser el de índice menor. LISTA=$\{3\}$;
  El arco $(2,t) \in A(2)$ y t no está etiquetado;
  Nodos etiquetados=$\{s, 2, 3, t\}$;
  pred(t)=2;
  LISTA=$\{3, t\}$

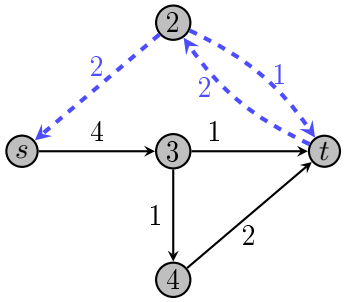
- Como t está etiquetado, ejecutamos *aumento*:

  Tenemos pred(t)=2 y pred(2)=s;

  $P = \{(s,2),(2,t)\}$, cuya capacidad residual es $\delta = \text{mín}\{2,3\} = 2$;

  Aumentamos 2 unidades de flujo a lo largo de P, con lo que $F = \delta = 2$.

  Actualizamos la red residual.



SEGUNDA ITERACIÓN:

- Nodos etiquetados=$\emptyset$;
  pred(j)=0 $\forall j \in N$;
  Nodos etiquetados=$\{s\}$;
  LISTA=$\{s\}$;

- Eliminamos s de LISTA. LISTA=$\emptyset$;
  El arco $(s,3) \in A(s)$ y el nodo 3 no está etiquetado;
  Nodos etiquetados=$\{s, 3\}$;
  pred(3)=s;
  LISTA=$\{3\}$;

- Eliminamos el nodo 3 de LISTA. LISTA=$\emptyset$;
  Los arcos $(3,t),(3,4) \in A(3)$ y los nodos 4 y t no están etiquetados;
  Nodos etiquetados=$\{s, 3, 4, t\}$;
  pred(4)=pred(t)=3;
  LISTA=$\{4, t\}$;

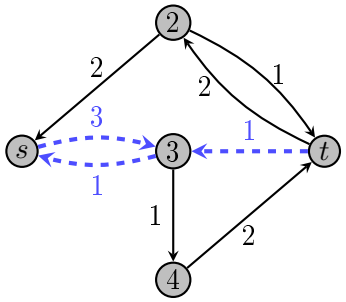
- Como t está etiquetado, ejecutamos *aumento*:

  Tenemos pred(t)=3 y pred(3)=s;

  $P = \{(s,3),(3,t)\}$, cuya capacidad residual el $\delta = \text{mín}\{4,1\} = 1$;

  Aumentamos 1 unidad de flujo a lo largo de P, con lo que $F = 2+1 = 3$;

  Actualizamos la red residual.



TERCERA ITERACIÓN:

- Nodos etiquetados=$\emptyset$;
  pred(j)=0 $\forall j \in N$;
  Nodos etiquetados=$\{s\}$;
  LISTA=$\{s\}$;

- Eliminamos s de LISTA. LISTA=$\emptyset$;
  El arco $(s,3) \in A(s)$ y el nodo 3 no está etiquetado;
  Nodos etiquetados=$\{s, 3\}$;
  pred(3)=s;
  LISTA=$\{3\}$;

- Eliminamos el nodo 3 de LISTA. LISTA=$\emptyset$;
  El arco $(3,4) \in A(3)$ y el nodo 4 no está etiquetado;
  Nodos etiquetados=$\{s, 3, 4\}$;
  pred(4)=3;
  LISTA=$\{4\}$;

- Eliminamos el nodo 4 de LISTA. LISTA=$\emptyset$;
  El arco $(4,t) \in A(4)$ y el nodo t no está etiquetado;
  Nodos etiquetados=$\{s, 3, 4, t\}$;
  pred(t)=4;
  LISTA=$\{t\}$;

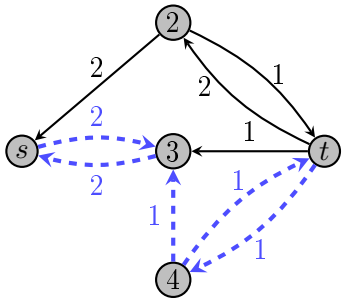
- Como t está etiquetado, ejecutamos *aumento*:

  Tenemos pred(t)=4, pred(3)=3 y pred(3)=s;

  $P = \{(s,3),(3,4),(4,t)\}$, cuya capacidad residual el $\delta = \text{mín}\{3,1,2\} = 1$;

  Aumentamos 1 unidad de flujo a lo largo de P, con lo que $F = 3 + 1 = 4$;

  Actualizamos la red residual.



CUARTA ITERACIÓN:

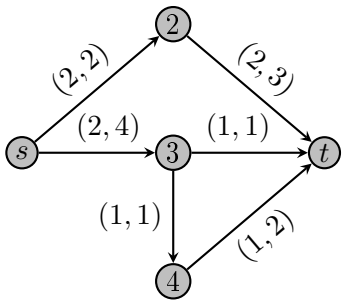
- Nodos etiquetados=$\emptyset$;
  pred(j)=0 $\forall j \in N$;
  Nodos etiquetados=$\{s\}$;
  LISTA=$\{s\}$;

- Eliminamos s de LISTA. LISTA=$\emptyset$;
  El arco $(s,3) \in A(s)$ y el nodo 3 no está etiquetado;
  Nodos etiquetados=$\{s, 3\}$;
  pred(3)=s;
  LISTA=$\{3\}$;

- Eliminamos el nodo 3 de LISTA. LISTA=$\emptyset$;
  No existen arcos que salgan del nodo 3, por tanto no hay actualizaciones;

- Como LISTA=$\emptyset$ y t no está etiquetado, el algoritmo termina.
  El flujo máximo de la red es $F = 4$.

Utilizando lo visto en la Sección 4.2, obtenemos que las variables de flujo asociadas a dicho valor de flujo máximo son $f_{s2} = 2,\ f_{s3} = 2,\ f_{2t} = 2,\ f_{3t} = 1,\ f_{34} = 1$ y $f_{4t} = 1$, es decir, obtendríamos la siguiente red con flujo.



4.4.1. Exactitud del algoritmo y resultados relacionados

Como acabamos de ver, para cada iteración del algoritmo caben dos posibilidades: o bien se realiza un aumento o bien el algoritmo termina pues no llega a etiquetar el sumidero. En el último caso, el algoritmo nos devuelve el valor asociado a un flujo $\boldsymbol{f}$ afirmando que es el valor del flujo máximo. Veámoslo.

Supongamos que el algoritmo ha terminado. Denotemos por S el conjunto de nodos etiquetados y $\overline{S} = N - S$. Claramente $s \in S$ y $t \in \overline{S}$, luego $(S, \overline{S})$ define un corte $s-t$. Además, $r_{ij} = 0\ \forall (i,j) \in (S, \overline{S})$, pues el algoritmo no puede etiquetar ningún nodo de $\overline{S}$ desde un ningún nodo de S. Teniendo en cuenta que las capacidades residuales vienen dadas por $r_{ij} = (u_{ij} - f_{ij}) + f_{ji}$ con $f_{ij} \leq u_{ij}$ y $f_{ji} \geq 0$, se deduce que $u_{ij} = f_{ij}\ \forall (i,j) \in (S, \overline{S})$ y $f_{ij} = 0\ \forall (i,j) \in (\overline{S}, S)$. Para estos valores de flujo se tiene:

$$F = \sum_{(i,j)\in(S,\overline{S})} f_{ij} - \sum_{(i,j)\in(\overline{S},S)} f_{ij} = \sum_{(i,j)\in(S,\overline{S})} u_{ij} = U(S, \overline{S}).$$

Es decir, el valor del flujo coincide con la capacidad del conjunto de corte $(S, \overline{S})$. Pero, como consecuencia del Lema 2.1, se deduce que $\boldsymbol{f}$ es un flujo máximo para la red y $(S, \overline{S})$ es el conjunto de corte con capacidad mínima.

Observemos que, tal y como comentamos en la Sección 2.4, el algoritmo de etiquetado nos proporciona la demostración para el Teorema max-flow min-cut al construir un flujo factible y un conjunto de corte tales que el valor del flujo coincide con la capacidad del conjunto de corte.

Finalmente, enunciaremos y probaremos un par de resultados importantes ayudados de las conclusiones obtenidas del algoritmo de etiquetado.

Teorema 4.1 (Teorema de trayectorias aumentadas)**.** *Dado un flujo $\boldsymbol{f}^*$, $\boldsymbol{f}^*$ es un flujo máximo si, y solo si, la red residual $R(\boldsymbol{f}^*)$ no contiene ninguna trayectoria aumentada.*

Demostración. Supongamos que $\boldsymbol{f}^*$ es un flujo máximo. Si $R(\boldsymbol{f}^*)$ contiene una trayectoria aumentada, P, podemos enviar $\delta = \text{mín}\{r_{ij} : (i,j) \in P\} > 0$ unidades de flujo adicional a lo largo de P, con lo que $\boldsymbol{f}^*$ no sería un flujo máximo.
Por otra parte, si $R(\boldsymbol{f}^*)$ no contiene trayectorias aumentadas, el conjunto de nodos etiquetados y el conjunto de nodos no etiquetados generan un corte $s-t$ cuya capacidad coincide con el valor del flujo $\boldsymbol{f}^*$, lo cual implica que el flujo es un flujo máximo. □

Teorema 4.2 (Teorema de integralidad)**.** *Si todas las capacidades de los arcos son enteras, entonces el problema de flujo máximo tiene un flujo máximo entero.*

Demostración. El resultado se deduce fácilmente aplicando inducción al número de aumentos: Teniendo que en cuenta que el algoritmo de etiquetado comienza con la red residual

para el flujo $\boldsymbol{f} = \mathbf{0}$, se tiene que $r_{ij} = u_{ij}$, luego todas las capacidades residuales iniciales son enteras. Si identificamos una trayectoria aumentada, las unidades de flujo que se envían a través de ella es el mínimo de las capacidades residuales de los arcos que la forman, las cuales, por hipótesis de inducción, son enteras. Esto implica que las nuevas capacidades residuales vayan a ser también enteras.
Continuando con este proceso tendríamos que las capacidades residuales, además de las capacidades de los arcos, son siempre enteras. Entonces, al sustraer los valores para los arcos de flujo estos serán también siempre enteros. Es decir, hemos visto que se tiene un flujo entero que además es máximo por ser resultado del algoritmo de etiquetado.

Por otra parte, como las capacidades residuales son enteras no nulas, cada aumento contribuye, como mínimo, con una unidad adicional al valor del flujo. Por el Lema 2.1, el valor del flujo máximo no puede exceder la capacidad de ningún conjunto de corte luego el algoritmo terminará en un número finito de iteraciones. □

4.4.2. Complejidad del algoritmo

Vamos a analizar la complejidad del algoritmo mediante el análisis del peor caso: Por el Lema 2.1, sabemos que el valor del flujo máximo para la red siempre estará acotado por la capacidad de cualquier corte $s-t$. En particular, si consideramos los conjuntos $S = \{s\}$ y $\overline{S} = N - \{s\}$ estamos definiendo un corte $s-t$, con lo que:

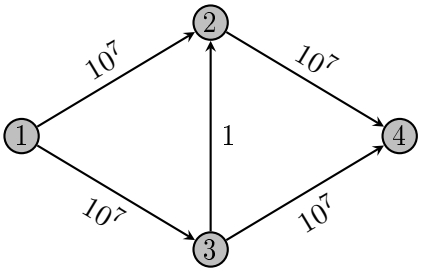
$$F \leq \sum_{(i,j)\in(s,N-\{s\})} u_{ij}$$

Si todas las capacidades de los arcos son enteras y existe un número finito U que sea una cota superior para todas ellas, entonces el valor del flujo máximo estará acotado por nU ($F \leq nU$). Por otra parte hemos visto que el algoritmo de etiquetado, cada vez que realiza un aumento, incrementa en una unidad, como mínimo, el valor del flujo, por lo que el algoritmo habrá finalizado después de nU aumentos. ¿Qué coste proporciona, por lo tanto, cada aumento? Cada vez que se lleva a cabo un aumento se realiza una pequeña implementación del algoritmo de búsqueda para analizar las listas de adyacencia de los nodos que van formando la trayectoria aumentada. Hemos visto en la Sección 3.3 que el algoritmo de búsqueda posee un tiempo $O(m)$. Tenemos entonces que el algoritmo de etiquetado resolverá el problema de flujo máximo en un tiempo de, como máximo, $O(nmU)$.

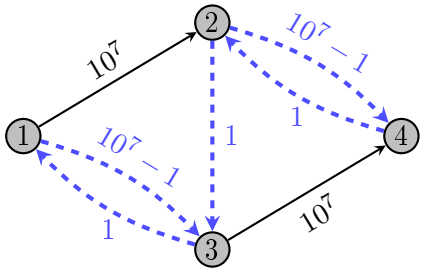
A pesar de que parece que el algoritmo es polinomial con respecto a las variables n, m y U, debemos tener en cuenta que para almacenar el valor U el ordenador empleará código binario, por tanto, el espacio requerido será $b = \log_2 U$ bits. Por tanto, como función de la entrada del problema, $O(nmU) = O(nm2^b)$ que claramente no es polinomial en n, m y b.

4.4.3. Desventajas del algoritmo

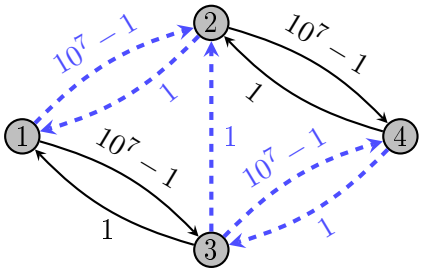
Aunque se ha observado que el algoritmo de etiquetado es un algoritmo que se desarrolla bastante bien, no deja de tener varios inconvenientes. En primer lugar supongamos que la cota seleccionada para las capacidades de los arcos es extremadamente grande, entonces es posible que el algoritmo llegue a realizar todas esas iteraciones, lo cual supone un coste computacional muy grande. Veamos un ejemplo:



(a) Red residual para el flujo cero.



(b) Red residual después de aumentar una unidad a lo largo del camino $1-3-2-4$.



(c) Red residual después de aumentar una unidad a lo largo del camino $1-2-3-4$.

Figura 4.11: Ejemplo patológico del algoritmo de etiquetado.

El algoritmo de etiquetado podría estar mandando una unidad de flujo en cada aumento alternando las trayectorias vistas en la Figura 4.11, con lo que realizaría 10^7 iteraciones. Obtenemos, por tanto, que el algoritmo necesitó 10^7 iteraciones para conseguir el flujo máximo, mientras que si hubiera seleccionado en la primera iteración el camino $1-2-4$ o el camino $1-3-4$, el algoritmo proporcionaría el mismo resultado en 2 iteraciones.

Un segundo inconveniente del algoritmo de etiquetado surge al considerar capacidades irracionales en los arcos, pues el algoritmo podría no finalizar. En el caso de algunos ejemplos patológicos para el problema de flujo máximo, el algoritmo de etiquetado no

termina, sin embargo, se obtienen una sucesión de valores del flujo que resulta convergente. Así a todo, siempre converge a un valor estrictamente menor al valor del flujo máximo. Notar, sin embargo, que el Teorema max-flow min-cut se verifica incluso si las capacidades son irracionales.

Por último, un tercer inconveniente del algoritmo de etiquetado viene dado por el hecho de que cuando se finaliza una iteración las etiquetas generadas son eliminadas, perdiéndose así una gran cantidad de información que podría ser reciclada para las siguientes iteraciones. Por tanto, lo ideal sería mantener dichas etiquetas para ahorrarnos el coste computacional que supone volver a empezar la búsqueda de los nodos etiquetados.

En el siguiente capítulo presentamos variantes del algoritmo de trayectorias aumentadas que tratan de subsanar las debilidades del algoritmo de etiquetado que acabamos de comentar.

Capítulo 5

Mejoras del algoritmo

Este último capítulo se centra en ver cómo aplicando pequeñas variantes al algoritmo de trayectorias aumentadas, utilizando recursos definidos a lo largo de esta memoria, se consiguen subsanar las principales debilidades del algoritmo de etiquetado además de ir mejorando con cada uno de ellos el tiempo de resolución empleado. Para el desarrollo de estos, hemos tomado como referencia la Sección 7.3 y la Sección 7.4 del libro Ahuja et al. (1993). Como finalización, ilustraremos brevemente una familia de algoritmos con una filosofía distinta a la construcción de trayectorias aumentadas y que no sufre algunas de las debilidades de estos últimos. Para esto tomaremos como referencia la Sección 7.6 del libro antes mencionado.

5.1. Algoritmo de trayectorias aumentadas de máxima capacidad

A continuación presentamos informalmente el algoritmo de trayectorias aumentadas de máxima capacidad, en cuya idea se apoya el algoritmo de escalado de capacidades que veremos en la Sección 5.2 y que puede verse como una implementación eficiente de la misma.

El *algoritmo de trayectorias aumentadas de máxima capacidad* proporciona un método para escoger de una forma más eficiente los caminos dirigidos entre la fuente y el sumidero, mejorando así los tiempos de ejecución para el algoritmo de etiquetado.

Partiendo de una red residual, el algoritmo identifica todas las trayectorias aumentadas entre la fuente y el sumidero y calcula sus respectivas capacidades residuales. Una vez tiene todos los valores, realizará el envío de flujo sobre aquel camino que posea la mayor capacidad residual. Observemos que utilizando este algoritmo resolveríamos el ejemplo

dado en la Figura 4.11 en 2 iteraciones pues se escogería o el camino $1-2-4$ o el camino $1-3-4$ en primer lugar.

Si F es el valor del flujo máximo, dado un flujo $\boldsymbol{f}^*$ con valor F^*, se pueden encontrar como mucho m caminos dirigidos en $R(\boldsymbol{f}^*)$ desde la fuente al sumidero de modo que la suma de todas las capacidades residuales sea $(F - F^*)$. Esto es así puesto que cada vez que enviamos flujo a lo largo de una trayectoria estamos saturando al menos una arista de la red residual, con lo que después de realizar, como mucho, m trayectorias deberíamos haber sido capaces de enviar el flujo máximo entre la fuente y el sumidero[1]. Pero entonces, como aún se pueden enviar $(F - F^*)$ unidades de flujo a lo largo de la red, la trayectoria de máxima capacidad debe tener una capacidad residual de al menos $\frac{(F-F^*)}{m}$ unidades.

Supongamos ahora que tenemos $2m$ aumentos consecutivos de máxima capacidad en la red $R(f^*)$. Si para todos los aumentos se envían al menos $\frac{(F-F^*)}{2m}$ unidades de flujo, habremos alcanzado el valor del flujo máximo al finalizarse dichos aumentos, y puede que incluso antes. Por otro lado, si tuviésemos un aumento con menos de $\frac{(F-F^*)}{2m}$ unidades de flujo, esto supondría que se ha reducido la capacidad residual de la trayectoria de máxima capacidad por un factor de al menos 2. Sabemos que para reducir un valor U a una unidad mediante el uso de un factor 2 necesitaremos $\log_2 U$ unidades. Como la capacidad residual de cualquier trayectoria aumentada siempre está entre 1 y $2U$ unidades (recordemos que U es la cota para las capacidades de los arcos), después de $O(m \log_2 U)$ iteraciones el flujo obtenido debe ser máximo.

Hemos conseguido, por tanto, reducir el tiempo del algoritmo de etiquetado a un tiempo $O(m \log_2 U)$. Pese a esto, debemos tener en cuenta que el algoritmo de trayectorias aumentadas de máxima capacidad necesita un trabajo adicional en cada ejecución, pues debe identificar la trayectoria aumentada con mayor capacidad. A continuación presentamos un algoritmo que se apoya en esta idea para conseguir mejorar la eficiencia computacional del algoritmo de etiquetado.

5.2. Algoritmo de escalado de capacidades

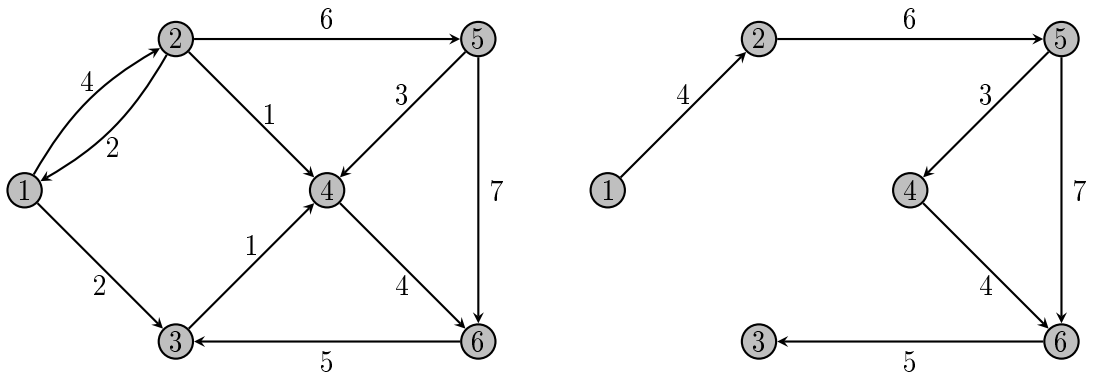
El *algoritmo de escalado de capacidades* es una mejora computacional del algoritmo de trayectorias aumentadas de máxima capacidad para eliminar el coste que supone el identificar todas las trayectorias aumentadas en la red residual.

La idea del algoritmo se basa en que solamente se enviará flujo por aquellas trayectorias

[1] Para demostrar formalmente esta idea habría que tener algo de cuidado con el hecho de que una trayectoria puede liberar capacidad en un arco saturado en alguna trayectoria anterior. La demostración formal que nos garantiza que con m trayectorias siempre será suficiente se puede ver en el Teorema 3.5 del libro Ahuja et al. (1993).

aumentadas cuyas capacidades residuales sean lo "suficientemente grandes", es decir, dado un parámetro Δ, solo trabajaremos con aquellas trayectorias que tengan una capacidad residual igual o superior a Δ. Este parámetro se podrá actualizar, y por tanto disminuir, cuando no exista ninguna trayectoria que verifique estas condiciones.

Partiendo de una red residual $R(\boldsymbol{f})$ definimos la *red* Δ*-residual*, $R(\boldsymbol{f}, \Delta)$, como la red resultante tras seleccionar aquellos arcos de $R(\boldsymbol{f})$ cuya capacidad residual es mayor o igual que Δ. Nótese que en el caso $\Delta = 1$ la red residual coincide con la red Δ-residual. Estas nuevas redes serán las que sustituirán a la red residual habitual en el algoritmo de escalado de capacidades.



(a) Red residual para un flujo $\boldsymbol{f}$.

(b) Red Δ-residual para $\Delta = 3$.

Figura 5.1: Ejemplo de una red Δ-residual.

El conjunto de iteraciones del algoritmo en las que Δ permanece constante forman lo que se conoce como una *fase de escala*. Si además queremos especificar en qué valor de Δ nos encontramos diremos que es una *fase de* Δ*-escala*. Observemos que en una fase de Δ-escala el algoritmo estará trabajando sobre la red $R(\boldsymbol{f}, \Delta)$, por lo que si existe una trayectoria aumentada, su capacidad residual será de al menos Δ unidades.

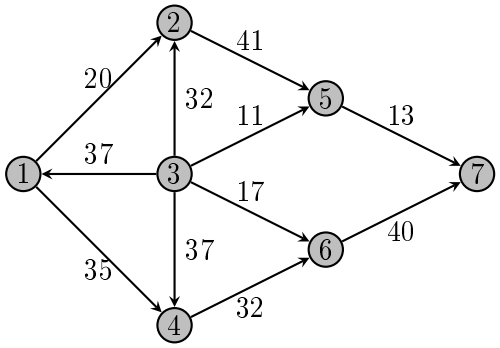
El algoritmo comienza inicializando $\Delta = 2^{\lfloor \log_2 U \rfloor}$ y, tras finalizar una fase de escala, actualiza este valor a $\frac{\Delta}{2}$ hasta llegar a $\Delta = 1$, que será la última iteración del algoritmo. Se realizarán, por lo tanto, $1 + \lfloor \log_2 U \rfloor = O(\log_2 U)$ fases de escala. Como la última fase de escala se hace sobre la red $R(\boldsymbol{f})$, el algoritmo de etiquetado nos asegura que el resultado será un flujo máximo.

Algoritmo 4: Algoritmo de escalado de capacidades.

inicio
  $\boldsymbol{f} = \mathbf{0}$;
  $\Delta = 2^{\lfloor \log_2 U \rfloor}$;
  mientras $\Delta \geq 1$ **hacer**
    mientras $R(\boldsymbol{f}, \Delta)$ *contenga una trayectoria aumentada* **hacer**
      Identificar la trayectoria aumentada P en $R(\boldsymbol{f}, \Delta)$;
      $\delta = \text{mín}\{r_{ij} : (i,j) \in P\}$;
      Aumentar δ unidades de flujo a lo largo de P y actualizar $R(\boldsymbol{f}, \Delta)$;
    fin
    $\Delta = \frac{\Delta}{2}$;
  fin
fin

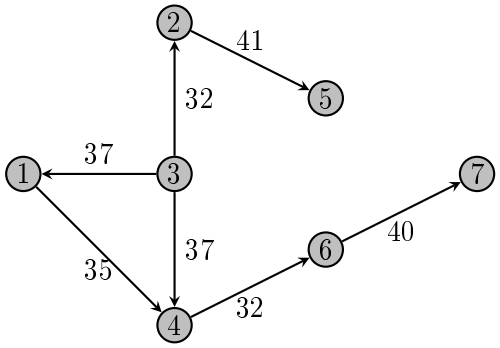
Implementación del algoritmo

Dada la siguiente red, resolveremos el problema del flujo máximo mediante el algoritmo de escalado de capacidades:

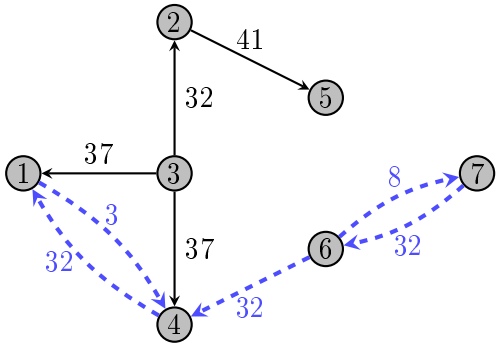


Tomando $U = 41$ tenemos una cota superior para todas las capacidades de las aristas.

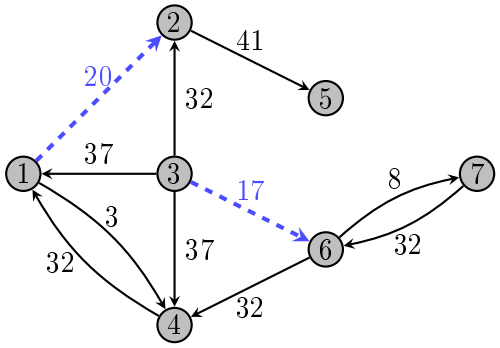
PASO 1: Inicializamos $\boldsymbol{f} = \mathbf{0}$. Además, $\Delta = 2^{\lfloor \log_2 U \rfloor} = 2^5 = 32$. Obtenemos $R(\boldsymbol{f}, \Delta)$:



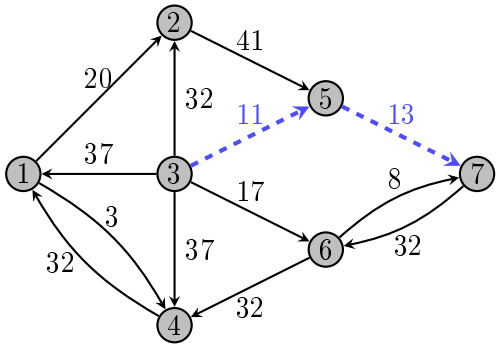
PASO 2: Como $\Delta \geq 1$, y existe un camino dirigido entre el nodo 1 y el nodo n: Identificamos la trayectoria aumentada $P = \{(1,4),(4,6),(6,7)\}$ cuya capacidad residual es $\delta = \text{mín}\{35, 32, 40\} = 32$. Aumentamos δ unidades de flujo a lo largo de P, con lo que $F = 32$, y actualizamos la red Δ-residual.



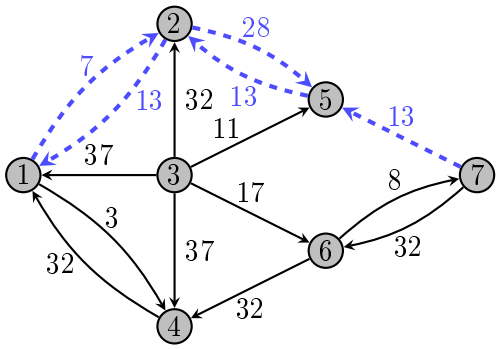
PASO 3: Como no existen trayectorias aumentadas de la fuente al sumidero, actualizamos $\Delta = \frac{\Delta}{2} = 16$. Obtenemos la red Δ-residual:



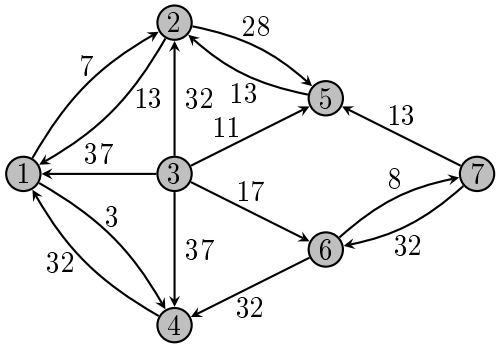
PASO 4: Como no existen trayectorias aumentadas de la fuente al sumidero, volvemos a actualizar $\Delta = \frac{\Delta}{2} = 8$. Obtenemos la red Δ-residual:



PASO 5: Tenemos $\Delta \geq 1$ y un camino dirigido entre la fuente y el sumidero: Identificamos $P = \{(1,2),(2,5),(5,7)\}$ cuya capacidad residual es $\delta = \text{mín}\{20, 41, 13\} = 13$. Aumentamos δ unidades de flujo a lo largo de P, así $F = 32 + 13 = 45$, y actualizamos la red Δ-residual:



PASO 6: Como no existen caminos dirigidos desde la fuente al sumidero, actualizamos $\Delta = \frac{\Delta}{2} = 4$ y obtenemos la red Δ-residual:



PASO 6: Como todavía no existen caminos dirigidos desde la fuente al sumidero, actualizamos $\Delta = \frac{\Delta}{2} = 2$ y obtenemos la red Δ-residual $R(\boldsymbol{f}, 2) = R(\boldsymbol{f}, 4)$.

PASO 7: Nuevamente, como no se ha añadido ningún arco en la iteración anterior, sigue sin existir una trayectoria aumentada, con lo que actualizamos $\Delta = \frac{\Delta}{2} = 1$. Esta

actualización no nos proporciona ningún arco nuevo tampoco, luego $R(\boldsymbol{f}, 1) = R(\boldsymbol{f}, 2)$.

PASO 8: Obviamente, sigue sin haber trayectorias aumentadas de la fuente al sumidero. Actualizamos $\Delta = \frac{\Delta}{2} = \frac{1}{2}$. Como $\Delta < 1$, el algoritmo finaliza. Devuelve que el valor del flujo máximo es $F = 45$ asociado a $f_{14} = f_{46} = f_{47} = 32$ y $f_{12} = f_{25} = f_{57} = 13$.

5.2.1. Complejidad del algoritmo

Para analizar la eficiencia del algoritmo, nos queda por determinar cuántos aumentos se realizarán por cada fase de escala. Veamos que, como mucho, se realizarán $2m$ aumentos: Supongamos que obtenemos un flujo $\boldsymbol{f'}$ al finalizar una fase de Δ-escala cuyo valor es F'. Sea S el conjunto formado por todos los nodos que son alcanzables desde la fuente en $R(\boldsymbol{f'}, \Delta)$ y $\overline{S} = N - S$. Puesto que la fase ha acabado, no existen trayectorias aumentadas desde la fuente al sumidero en dicha red, luego $t \in \overline{S}$ y $(S, \overline{S})$ define un corte $s - t$. Por definición de S sabemos que la capacidad residual de los arcos en $(S, \overline{S})$ es menor estrictamente que Δ, por lo que la capacidad del conjunto de corte no superará el valor $m\Delta$. Si suponemos que F es el valor del flujo máximo, por el Lema 2.1 tenemos que $F - F' \leq m\Delta = U(S, \overline{S})$. Por otra parte, la siguiente fase de escala será la fase de $\frac{\Delta}{2}$-escala, luego cada trayectoria aumentada en $R(\boldsymbol{f'}, \frac{\Delta}{2})$ tendrá como mínimo una capacidad residual de $\frac{\Delta}{2}$ unidades, lo que supone que no se podrán realizar más de $2m$ aumentos. Como estamos escogiendo un Δ arbitrario, podemos concluir que para toda fase de Δ-escala se realizarán como mucho $2m$ aumentos.

Hemos visto en el algoritmo de etiquetado que se requiere un tiempo de $O(m)$ para identificar una trayectoria aumentada, luego para cada actualización de la red Δ-residual se requiere también un tiempo $O(m)$. Podemos deducir, por lo tanto, el siguiente resultado:

Proposición 5.1. *El algoritmo de escalado de capacidades resuelve el problema de flujo máximo después de $O(m \log_2 U)$ aumentos en un tiempo $O(m^2 \log_2 U)$.*

5.3. Algoritmo de trayectorias aumentadas más cortas

El *algoritmo de trayectorias aumentadas más cortas* se basa en enviar flujo a lo largo de aquellas trayectorias aumentadas que tengan longitud mínima, es decir, a lo largo de aquellos caminos entre la fuente y el sumidero formados por la menor cantidad de arcos posibles. Para identificar los caminos de longitud mínima el algoritmo realiza un primer análisis de la red residual para asociar, posteriormente, una etiqueta de distancia válida a cada nodo de la red. A raíz de estos valores se crearán caminos admisibles entre la fuente y el sumidero a través de los cuales incrementaremos el envío de flujo. Recordemos que un

camino se dice que es admisible cuando todos los arcos que lo componen son admisibles, es decir, si se verifica $d(i) = d(j) + 1$ para todo arco (i, j) perteneciente al camino.

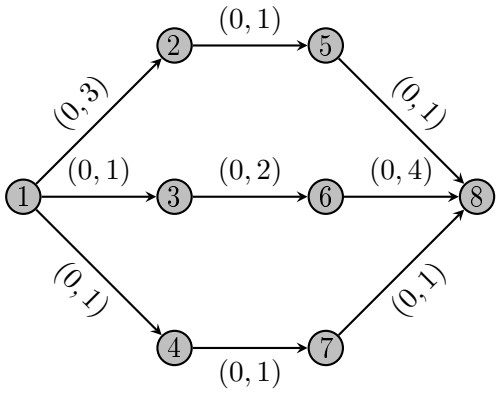
Con el fin de alcanzar el sumidero a través de un camino admisible, el algoritmo irá creando un *camino admisible parcial* desde la fuente a un nodo i, denominado *nodo actual*, $i \neq t$, al que, iterativamente, le aplica operaciones o bien de *avance* o bien de *retroceso*. En el caso de que exista un arco (i, j) admisible realiza una operación de avance añadiendo el arco (i, j) a nuestro camino parcial; en caso contrario, realiza un retroceso eliminando el arco (k, i) que pertenece al camino y un *reetiquetado* que consistirá en aumentar el valor de la etiqueta de distancia asociada al nodo i. Este proceso se repetirá hasta que alcancemos el sumidero que es cuando haremos el aumento de flujo.

Puesto que, como sabemos, la distancia mínima entre un nodo arbitrario de la red y el sumidero es monótona creciente tras realizar un aumento, explotando esta propiedad se consigue reducir el tiempo promedio de cada aumento de $O(m)$ a $O(n)$.

Ilustración del algoritmo

Antes de ilustrar cómo trabaja el algoritmo notemos que, por convención, si tenemos dos arcos admisibles (i, j) y (i, k), siempre seleccionaremos aquel cuyo índice sea más pequeño.

Procedemos, por tanto, a resolver el problema de flujo máximo para un ejemplo particular mediante el algoritmo de trayectorias aumentadas más cortas descrito en el Algoritmo 5. Consideremos la siguiente red con flujo donde el nodo 1 será la fuente y el nodo 8 el sumidero:



PRIMERA ITERACIÓN:

- $\boldsymbol{f} = \mathbf{0}$;
- Se obtienen las etiquetas de distancia válidas $d(i)$ para la red $R(\boldsymbol{f})$; (ver Figura 5.2)

- $i = s$;
- Como $d(s) < 8$ y s tiene arcos admisibles en $R(\boldsymbol{f})$, ejecutamos *avance(s)*:

  Escogemos el arco admisible $(1, 2) \in A(i)$;

  pred(2)=1;

  i=2;

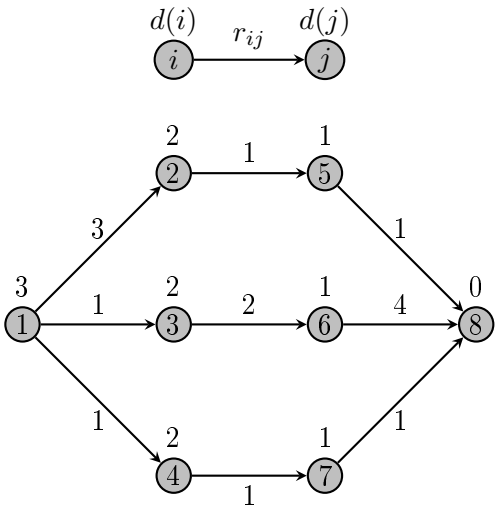


Figura 5.2: Red residual con etiquetas de distancia.

SEGUNDA ITERACIÓN:

- Como $d(s) < 8$ puesto que no se ha actualizado, y el nodo 2 tiene un arco admisible, ejecutamos *avance(2)*:

  El arco $(2, 5) \in A(2)$ es el único arco admisible;

  pred(5)=2;

  i=5;

TERCERA ITERACIÓN:

- Como $d(s) < 8$, y el nodo 5 tiene un arco admisible, ejecutamos *avance(5)*:

  El arco $(5, 8) \in A(5)$ es el único arco admisible;

  pred(8)=5;

  i=8;

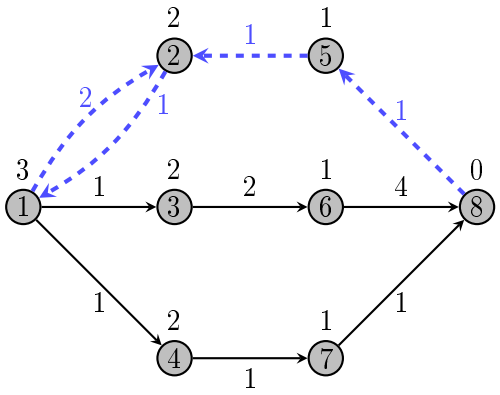
- Puesto que $i = t = 8$, se ejecuta *aumento*:

  Identificamos la trayectoria aumentada P con el uso de los predecesores: $P = \{(1,2),(2,5),(5,8)\}$

  $\delta = \text{mín}\{3,1,1\} = 1$

  Aumentamos δ unidades de flujo a lo largo de P, con lo que $F = \delta = 1$, y actualizamos la red residual.

- $i = s$;



CUARTA ITERACIÓN:

- Seguimos con $d(s) < 8$ y s tiene arcos admisibles, por tanto realizamos *avance(s)*:

  Escogemos el arco admisible $(1,2) \in A(i)$;

  pred(2)=1;

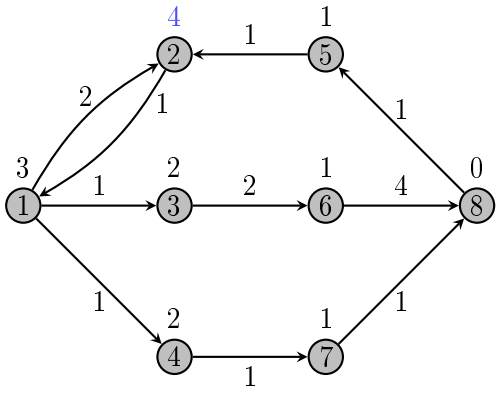
  i=2;

QUINTA ITERACIÓN:

- $d(s) < 8$, sin embargo, el nodo 2 no posee ningún arco admisible, con lo que ejecutamos *retroceso(2)*:

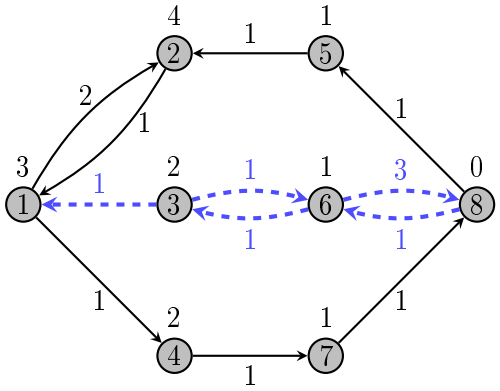
  $d(2) = \text{mín}\{d(j) + 1 : (2,j) \in A(2) \ y \ r_{2j} > 0\} = \text{mín}\{d(1) + 1\} = 4$;

  Como $i = 2 \neq s$, hacemos i=pred(2)=1;



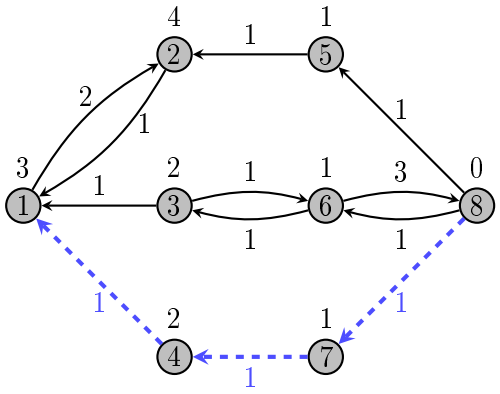
SIGUIENTES ITERACIONES:

- El algoritmo encontrará, mediante pasos análogos a los explicados anteriormente, el camino admisible $1-3-6-8$ cuya capacidad residual es $\delta = \text{mín}\{1,2,4\} = 1$, con lo que $F = 1+1 = 2$.



SIGUIENTES ITERACIONES:

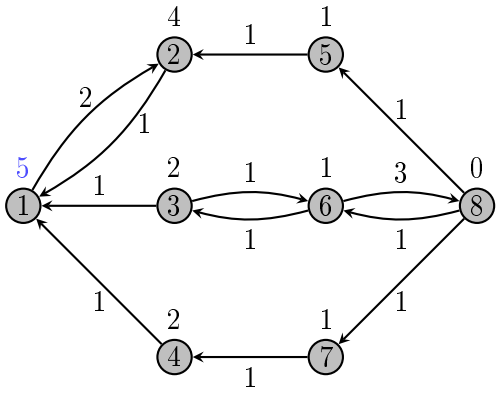
- De igual modo, identifica el camino admisible $1-4-7-8$ con capacidad residual $\delta = 1$. Se aumentan tales unidades de flujo a lo largo del camino, con lo que $F = 2+1 = 3$, y tenemos la siguiente red residual:



SIGUIENTE ITERACIÓN:

- Como acabamos de realizar un aumento, tenemos $i = s$, pues queremos identificar otro camino admisible. Sin embargo, el nodo s no tiene arcos admisibles en la red residual, luego ejecuta *retroceso(s)*:

  $d(s) = \text{mín}\{d(j) + 1 : (s, j) \in A(s) \ y \ r_{sj} > 0\} = \text{mín}\{d(2) + 1\} = 5;$



SIGUIENTE ITERACIÓN:

- Seguimos con $d(s) < 8$ y s tiene arcos admisibles, por tanto realizamos *avance(s)*:

  Escogemos el arco admisible $(1, 2) \in A(i)$;

  pred(2)=1;

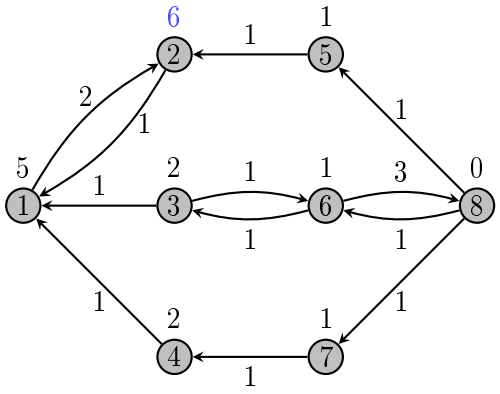
  i=2;

SIGUIENTE ITERACIÓN:

- $d(s) < 8$, sin embargo, el nodo 2 no posee ningún arco admisible, con lo que ejecutamos *retroceso(2)*:

$d(2) = \text{mín}\{d(j) + 1 : (2, j) \in A(2) \text{ y } r_{2j} > 0\} = \text{mín}\{d(1) + 1\} = 6;$

Como $i = 2 \neq s$, hacemos i=pred(2)=1;



SIGUIENTES ITERACIONES:

- Se repetirán estas dos últimas iteraciones hasta que $d(s) \geq 8$ puesto que no existen más caminos admisibles posibles en la red residual. Cuando $d(s) \geq 8$, el algoritmo finalizará y devolverá el valor del flujo máximo, $F = 3$, que estará asociado a las variables de flujo $f_{12} = f_{25} = f_{58} = f_{13} = f_{36} = f_{68} = f_{14} = f_{47} = f_{78} = 1$.

5.3.1. Exactitud del algoritmo

En este apartado vamos a probar que, efectivamente, la solución proporcionada por el algoritmo de trayectorias aumentadas más cortas es una solución al problema de flujo máximo.

Lema 5.2. *El algoritmo de trayectorias aumentadas más cortas mantiene etiquetas de distancia válidas en cada paso. Además, cada reetiquetado aumenta estrictamente el valor de la etiqueta de distancia asociada al nodo.*

Demostración. Nótese que cuando realizamos un avance no se modifican las capacidades residuales de la red ni las etiquetas de distancia, por lo que esta operación no afecta a la validez de las etiquetas de distancia. Probaremos el resultado aplicando el método de inducción al número de aumentos y reetiquetados realizados. Como el algoritmo construye inicialmente unas etiquetas de distancia válidas, tenemos que demostrar que tras un aumento, y tras un reetiquetado, las etiquetas siguen siendo válidas.

Algoritmo 5: Algoritmo de trayectorias aumentadas más cortas.

inicio
- $\boldsymbol{f} = \mathbf{0}$;
- Obtener las etiquetas de distancia $d(i)$;
- $i := s$;
- **mientras** $d(i) < n$ **hacer**
  - **inicio**
    - **si** *i tiene un arco admisible* **entonces**
      - Ejecutar *avance(i) (ver Algoritmo 6)*;
      - **si** $i = t$ **entonces**
        - Ejecutar *aumento (ver Algoritmo 3)*;
        - i=s;
      - **fin**
    - **en otro caso**
      - Ejecutar *retroceso(i) (ver Algoritmo 7)*;
    - **fin**
  - **fin**
- **fin**

fin

Algoritmo 6: Avance(i).

inicio
- Sea (i, j) un arco admisible en $A(i)$;
- pred(j)=i;
- $i = j$;

fin

Algoritmo 7: Retroceso(i).

inicio
- $d(i) = \text{mín}\{d(j) + 1 : (i, j) \in A(i)\ y\ r_{ij} > 0\}$;
- **si** $i \neq j$ **entonces**
  - i=pred(j);
- **fin**

fin

(a) Supongamos que acabamos de realizar un aumento. Como esta operación no implica ninguna modificación en las etiquetas de distancia, para todos los arcos existentes antes del aumento se siguen cumpliendo las condiciones de validación. Sin embargo, puede suceder que tras enviar flujo por un arco (i, j) se cree el arco inverso (j, i) con capacidad residual positiva, dando lugar a una nueva desigualdad, $d(j) \leq d(i) + 1$, que deben satisfacer las etiquetas de de distancia $d(i)$ y $d(j)$. Puesto que se realizó un aumento por el arco (i, j) el arco es admisible y las etiquetas de distancia verifican que $d(i) = d(j) + 1$, luego se tiene probada la desigualdad para el nuevo arco.

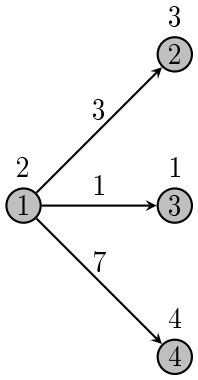
(b) En el caso de un retroceso(i) siempre se realiza un reetiquetado del nodo i, luego tenemos que comprobar que cada arco de llegada y cada arco de salida al nodo i siguen verificando las condiciones de validación. Supongamos que $d'(i)$ es la nueva etiqueta de distancia para el nodo i. Sabemos que el algoritmo realiza un reetiquetado cuando no existe ningún arco admisible en $A(i)$, es decir, cuando $d(i) \neq d(j) + 1$ para todo $(i, j) \in A(i)$ con $r_{ij} > 0$. Por hipótesis de inducción sabemos que las etiquetas de distancia iniciales son válidas, luego se verifica que $d(i) \leq d(j) + 1$, y, por lo tanto, $d(i) < d(j) + 1$ para todo $(i, j) \in A(i)$, $r_{ij} > 0$. Ahora bien, por definición $d'(i) = \text{mín}\{d(j) + 1 : (i, j) \in A(i)\ con\ r_{ij} > 0\}$, por tanto $d(i) < d'(i)$ y $d'(i) \leq d(j) + 1$ para todo $(i, j) \in A(i)$ con $r_{ij} > 0$. Hemos probado que para todo arco de salida del nodo i se siguen verificando las condiciones de validación. Además, cada reetiquetado aumenta estrictamente el valor de la etiqueta de distancia. Nos queda por ver si para un arco (k, i), $r_{ki} > 0$, siguen siendo válidas las etiquetas. Puesto que las etiquetas de distancia iniciales eran válidas, $d(k) \leq d(i) + 1$, luego, como acabamos de demostrar que $d(i) < d'(i)$, en particular se verifica también que $d(k) \leq d'(i) + 1$ y, por tanto, la nueva etiqueta también es válida.

□

El algoritmo de trayectorias aumentadas más cortas finaliza cuando $d(s) \geq n$, lo cual nos indica, por la Proposición 3.3, que no existen trayectorias aumentadas en la red residual. Por consiguiente, el flujo obtenido al finalizar el algoritmo es un flujo máximo.

5.3.2. Complejidad del algoritmo

Dada la lista de adyacencia de un nodo arbitrario i, diremos que el arco (i, j) es el *arco actual* si es el próximo arco que va a ser analizado, es decir, si es el próximo arco para el cual se va a comprar su admisibilidad.



Supongamos que la figura anterior nos muestra la lista de adyacencia del nodo 1. Entonces, el algoritmo comienza seleccionando el arco $(1, 2)$ como el arco actual y comprueba si se trata de un arco admisible. En este caso no es un arco admisible pues $d(1) \neq d(2) + 1$ por lo que, para poder continuar con el algoritmo, se actualiza la elección del arco actual al siguiente arco en la lista de adyacencia, es decir, el arco $(1, 3)$, que si que resulta ser admisible. Si se diera el caso de que el algoritmo analizase toda la lista de adyacencia y no encontrara ningún arco admisible, entonces se realizaría un reetiquetado del nodo 1 y el algoritmo volvería a comprobar, uno a uno, cada arco de la lista de adyacencia.

Cada vez que el algoritmo necesita hacer un reetiquetado del nodo i debe sustraer las etiquetas de distancia para todos aquellos nodos j tales que $(i, j) \in A(i)$, luego el tiempo empleado en un reetiquetado coincide con el tiempo empleado en identificar los arcos admisibles. Podemos deducir, por tanto, el siguiente resultado:

Proposición 5.3. *Si el algoritmo reetiqueta un nodo como mucho k veces, entonces el tiempo total que se necesita para encontrar arcos admisibles y reetiquetar los nodos es $O(k \sum_{i \in N} |A(i)|) = O(km)$.*

Otro de los resultados en los que nos apoyaremos para deducir la complejidad del algoritmo de trayectorias aumentadas más cortas es el que anunciamos a continuación.

Lema 5.4. *Si el algoritmo reetiqueta un nodo a lo sumo k veces, el algoritmo satura arcos en la red residual (es decir, reduce su capacidad residual a cero) como mucho km veces.*

Demostración. La base de la demostración se basa en ver que dadas dos saturaciones consecutivas del arco (i, j), tanto $d(i)$ como $d(j)$ aumentan su valor al menos 2 unidades. Si esto es cierto, como por hipótesis el algoritmo aumenta cada etiqueta a lo sumo k veces, entonces el algoritmo podría saturar el arco como mucho k veces.Así, se realizarían como mucho km saturaciones en la red residual.
Supongamos, por tanto, que tenemos un aumento que satura el arco (i, j). Como este arco

es admisible, se satisface que

$$d(i) = d(j) + 1.$$

Para que el arco (i, j) pueda volver a ser saturado, es necesario que antes exista un envío de flujo a lo largo del arco (j, i) pero entonces, por ser (j, i) un arco admisible, las nuevas etiquetas de distancia deben verificar que

$$d'(j) = d'(i) + 1.$$

Análogamente, si ahora se satura el arco (i, j) de nuevo, tenemos

$$d''(i) = d''(j) + 1.$$

Por el Lema 5.2 sabemos que un reetiquetado aumenta el valor de la etiqueta de distancia del nodo, con lo que tenemos las siguientes desigualdades:

$$d''(i) = d''(j) + 1 \geq d'(j) + 1 = d'(i) + 2 \geq d(i) + 2,$$

$$d''(j) = d''(i) - 1 \geq d(i) + 1 = d(j) + 2.$$

Obtenemos, por tanto, que dos saturaciones consecutivas del arco (i, j) implican que las etiquetas $d(i)$ y $d(j)$ aumenten su valor en al menos dos unidades. □

Lema 5.5.

(a) En el algoritmo de trayectorias aumentadas más cortas cada etiqueta de distancia aumenta su valor como mucho n veces. Por tanto, en total se tendrán como mucho n^2 reetiquetados.

(b) El número de aumentos es como mucho nm.

Demostración. En cada reetiquetado el valor de la etiqueta de distancia aumenta como mínimo una unidad, por lo que después de n reetiquetados tendríamos que $d(i) \geq n$. Esto supone que el algoritmo nunca volverá a seleccionar el nodo i para realizar un avance, pues para todo nodo k perteneciente al camino admisible parcial se tiene que $d(k) < d(s) < n$. Podemos concluir por lo tanto que el algoritmo reetiquetará cada nodo un máximo de n veces, realizando así como mucho n^2 reetiquetados.
Por el lema anterior, y dado que acabamos de probar que el algoritmo reetiqueta un nodo a lo sumo n veces, tenemos que se saturarán como mucho nm arcos. Además, por definición, cada aumento satura al menos un arco, luego nm es una cota para el número total de aumentos en la red. □

Combinando los resultados vistos en la Proposición 5.3 con los del Lema 5.5 se tiene que el tiempo necesario para encontrar los arcos admisibles en la red residual y el tiempo empleado en reetiquetar los nodos son ambos $O(nm)$. Por otra parte, el Lema 5.5 también nos dice que el número total de reetiquetados, o lo que es lo mismo, de retrocesos, será $O(n^2)$ y que se realizarán como mucho $O(nm)$ aumentos, lo que supone un tiempo total de $O(n^2m)$ para los aumentos.
Teniendo en cuenta ahora que cada avance añade un arco al camino parcial admisible y que cada retroceso lo elimina, puesto que toda trayectoria aumentada tiene a lo sumo longitud n, necesitaremos realizar como mucho n avances "limpios" (sin ningún retroceso) para alcanzar el sumidero y poder realizar un aumento. Además debemos compensar con los avances los retrocesos realizados, con lo que necesitaremos $O(n^2 + n^2m)$ avances. Obtenemos, por lo tanto, que la suma de todas las operaciones de los distintos tipos nos proporciona un orden $O(n^2 + n^2m)$ o, lo que es equivalente, $O(n^2m)$.

5.3.3. Mejora práctica

Como ya hemos visto, el algoritmo de trayectorias aumentadas más cortas no termina hasta que $d(s) \geq n$. A pesar de que este criterio es aceptable para el análisis del peor caso, puede suponer un coste computacional elevado cuando el flujo máximo ya se ha alcanzado en pocas iteraciones. El problema reside en que el algoritmo no es capaz de saber si se ha alcanzado un flujo máximo, con lo la finalidad de este apartado será el desarrollar una técnica que nos permita identificar un corte de capacidad mínima en la red.

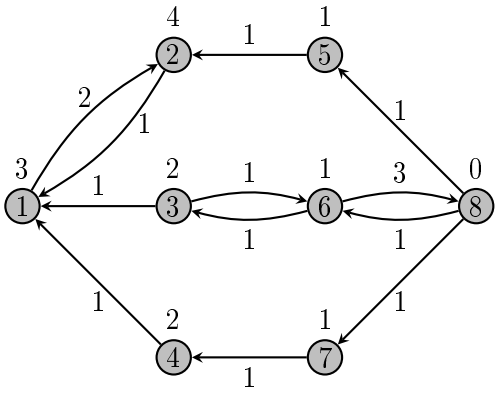


Figura 5.3: Red residual tras el último aumento.

Consideramos la red residual resultante tras ejecutar el último aumento en la ilustración del algoritmo de trayectorias aumentadas más cortas. Como podemos observar, esta red no posee ningún camino dirigido entre la fuente y el sumidero, lo cual implica que no se

podrá mandar flujo adicional entre estos nodos y que, por tanto, tenemos un flujo máximo. Para que el algoritmo sea capaz de identificar este hecho y finalizar sin necesidad de hacer más iteraciones, creamos un vector n-dimensional, $\boldsymbol{numb}$, tal que cada entrada $\boldsymbol{numb}(k)$ reflejará el número de nodos de la red cuya etiqueta de distancia tenga el valor k. Para nuestro ejemplo, ver Figura 5.2, el vector se inicializaría con los valores $\boldsymbol{numb}(0) = 1$, $\boldsymbol{numb}(1) = 3$, $\boldsymbol{numb}(2) = 3$, $\boldsymbol{numb}(3) = 1$ y el resto de entradas serían cero.

Cuando el algoritmo realice un reetiquetado y una etiqueta pase de un valor k_1 a un valor k_2, el valor de $\boldsymbol{numb}(k_1)$ disminuirá una unidad para aumentar en esa misma unidad el valor de $\boldsymbol{numb}(k_2)$. Si $\boldsymbol{numb}(k_1)$=0, el algoritmo finaliza asegurando que ya se ha alcanzado el flujo máximo para la red. Si aplicamos este criterio sobre la Figura 5.3, como el nodo 1 no posee arcos admisibles, se aumenta el valor de su etiqueta de distancia de 3 a 5 unidades y se obtiene $\boldsymbol{numb}(3) = 0$ y $\boldsymbol{numb}(5) = 1$, lo cual finalizaría el algoritmo.

Para argumentar por qué esto funciona, consideramos los subconjuntos de N definidos como $S = \{i \in N : d(i) > k_1\}$ y $\overline{S} = \{i \in N : d(i) < k_1\}$. Es fácil ver que $s \in S$ y $t \in \overline{S}$, con lo que tenemos un corte $s - t$ tal que $d(i) > d(j) + 1$ para todo $(i, j) \in (S, \overline{S})$. Como las etiquetas de distancia son válidas, deben verificar las condiciones de validación, en particular, deben verificar que $d(i) \leq d(j) + 1$ para todo $(i, j) \in R(\boldsymbol{f}, \Delta)$ con $r_{ij} > 0$, por lo que deducimos que $r_{ij} = 0$ para todo $(i, j) \in (S, \overline{S})$. Es decir, hemos encontrado un corte $s - t$ en la red residual cuya capacidad es 0, luego es un corte de capacidad mínima y el valor de flujo máximo sobre esta red también será 0. En otras palabras, no es posible enviar más unidades de flujo a lo largo de la red residual $R(\boldsymbol{f})$, por lo que $\boldsymbol{f}$ será un flujo máximo para la red original.

5.4. Algoritmo de escalado de capacidades con caminos más cortos

En la Sección 5.2 hemos desarrollado el algoritmo de escalado de capacidades capaz de resolver el problema de flujo máximo en un tiempo $O(m^2 \log_2 U)$. Ahora bien, veamos se puede mejorar el rendimiento de dicho algoritmo si introducimos las ideas del algoritmo de trayectorias aumentadas más cortas.

Este nuevo algoritmo combinado consistirá en aumentar flujo a lo largo de los caminos de longitud mínima entre la fuente y el sumidero en la red $R(\boldsymbol{f}, \Delta)$. Dichos caminos serán identificados por medio de las etiquetas de distancia de igual forma que se hacía para el algoritmo de trayectorias aumentadas más cortas.

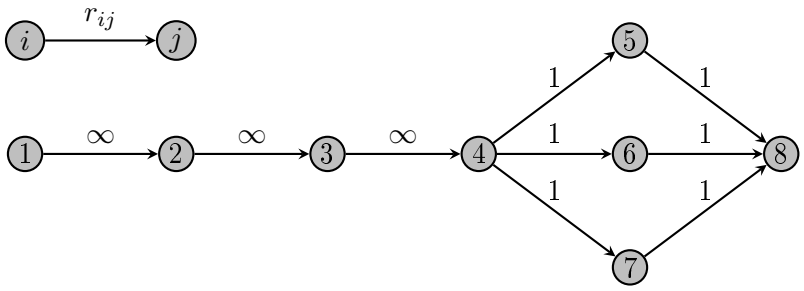
Recordemos que el algoritmo de escalado de capacidades realizaba $O(\log_2 U)$ fases de escala y en cada una de ellas necesitaba $O(m)$ aumentos. Por el análisis de complejidad

visto para el algoritmo de trayectorias aumentadas más cortas, para $O(m)$ aumentos se requiere un tiempo $O(mn)$. Como el resto de operaciones no sen ven afectadas con esta modificación del algoritmo mantienen un tiempo $O(nm)$, con lo que podemos concluir que este nuevo algoritmo, al que denominaremos *algoritmo de escalado de capacidades con caminos más cortos* tendrá un tiempo de ejecución $O(nm \log_2 U)$.

5.5. Algoritmo preflow-push

Para finalizar este capítulo ilustraremos cuál es la filosofía detrás del *algoritmo genérico preflow-push*.

Supongamos que tenemos el siguiente caso extremo:



Utilizando cualquier algoritmo de trayectorias aumentadas tendríamos que identificar los tres caminos de longitud cinco de la red y enviar a través de cada uno de ellos una unidad de flujo. Sin embargo, puesto que en todos los aumentos se repiten las tres primeras aristas, el algoritmo estará identificando y recorriendo el camino entre el nodo 1 y el nodo 4 tres veces. El algoritmo preflow-push pretende evitar esta situación teniendo en cuenta que si se enviaran 3 unidades de flujo del nodo 1 al nodo 4 y después se distribuyeran entre los tres caminos de longitud dos, el algoritmo se ahorraría el coste computacional derivado de repetir los arcos.

El algoritmo preflow-push consiste en enviar flujo a lo largo de arcos admisibles, en lugar de por una trayectoria aumentada completa. Este hecho podría propiciar que en algún nodo intermedio entre más cantidad de flujo de la que puede salir, es decir, tendremos nodos con *exceso* de flujo, $e(i) > 0$. La presencia de estos nodos, sin embargo, indica que el flujo no es factible para el problema puesto que no se verifican la condición de conservación de flujo en los nodos intermedios. El algoritmo, por tanto, identificará un nodo i con exceso y distribuirá dicha cantidad de flujo sobrante entre aquellos nodos adyacentes al mismo para los que exista un arco (i, j) admisible. El algoritmo finalizará cuando no existan nodos intermedios con exceso de flujo.

Nótese que si realizamos un envío de δ unidades desde el nodo i al nodo j, tanto $e(i)$ como r_{ij} disminuyen su valor δ unidades, mientras que $e(j)$ y r_{ji} aumentan tales unidades.

Consideremos el ejemplo dado en la Figura 5.4. Vamos a describir sobre la red inicial, Figura 5.4(a), los pasos que realizaría el algoritmo preflow-push para resolver el problema de flujo máximo:

- Primeramente, el algoritmo proporciona a todos los nodos adyacentes a la fuente un exceso positivo, es decir, satura todos los arcos $(1, j)$ de la red, consiguiendo así que $e(j) = r_{1j}$. Además, como el nodo 1 ya no posee arcos admisibles, fija $d(1) = n = 4$ (ver Figura 5.4(b)).

- Supongamos que el algoritmo selecciona el nodo 2 por tener exceso de flujo. Entonces, como el arco $(2, 4)$ es admisible, envía $\delta = \text{mín}\{e(2), r_{24}\} = 2$ unidades de flujo a través del mismo, con lo que se actualizan los valores $e(2) = e(2) - \delta = 0$ y $e(4) = e(4) + \delta = 2$ (ver Figura 5.4(c)).

- El algoritmo selecciona el nodo 3 por ser el único con exceso positivo. Como el arco $(3, 4)$ es admisible, mandamos $\delta = \text{mín}\{e(3), r_{34}\} = 2$ unidades de flujo por este arco y actualizamos los excesos $e(3) = e(3) - \delta = 1$ y $e(4) = e(4) + \delta = 4$ (ver Figura 5.4(d)).

- Puesto que el nodo 3 posee exceso de flujo pero no arcos admisibles, aumentamos el valor de su etiqueta de distancia, $d(3) = \text{mín}\{d(j) : (3, j) \in A(3),\ r_{ij} > 0\} = 5$ (ver Figura 5.4(e)).

- El algoritmo vuelve a seleccionar el nodo 3 por tener exceso de flujo, e identifica el arco admisible $(3, 1)$. Aumenta $\delta = \text{mín}\{e(3), r_{31}\} = 1$ unidades de flujo a lo largo del arco y actualiza $e(3) = e(3) - \delta = 0$ y $e(1) = e(1) - \delta = -1$ (ver Figura 5.4(f)).

- Como la red residual ya no posee nodos intermedios con exceso de flujo, el algoritmo finaliza y el valor del flujo máximo será $e(4) = 4$.

Cuando el algoritmo termina el flujo obtenido es un flujo factible pues se verifican las condiciones de conservación de flujo para los nodos intermedios. Además, como en la primera iteración hemos fijado $d(s) = n$, la Proposición 3.3 nos asegura que no existe ningún camino dirigido de la fuente al sumidero, con lo que estamos ante un flujo máximo.

Por último, vamos a ver una interpretación física del algoritmo preflow-push: Supongamos que los arcos están representando trozos de tuberías, los nodos, intersecciones de tuberías y la función distancia mide la longitud, en número de nodos, entre el nodo actual y el suelo. La idea consiste en crear una tubería que conecte la fuente y el sumidero.

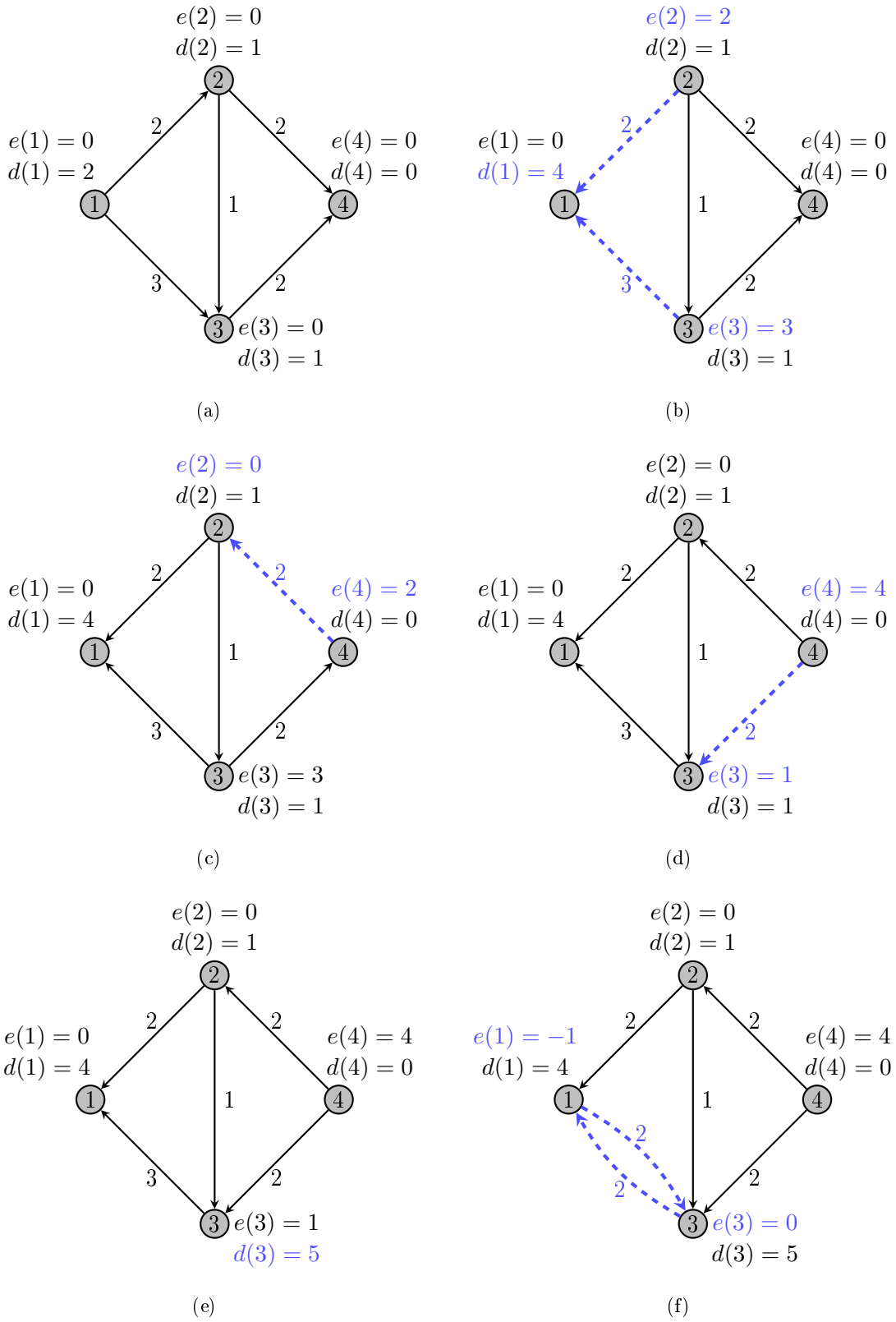


Figura 5.4: Ilustración del algoritmo preflow-push.

Para ello, inundamos todas las tuberías que conectan con la fuente. Si conseguimos que el agua fluya hacia el sumidero (lo cual se corresponde con que un arco sea admisible para el algoritmo) será porque existe una intersección a la que acoplar nuestra tubería que se encuentra por debajo de la misma. En caso de que el agua no fluya en una intersección, levantaremos dicha intersección hasta conseguir que se retome la bajada de agua. Debemos notar que si continuamos levantando las intersecciones podríamos provocar que el exceso de agua volviera a la fuente. Por tanto, podrían darse dos escenarios: que consigamos crear la tubería y llegue agua al sumidero, o que el agua vuelva hacia la fuente. En cualquiera de estos casos, el algoritmo finalizaría.

Para las variantes más eficientes de la familia de algoritmos preflow-push se tienen una complejidad computacional de $O(n^2\sqrt{m})$ y $O(nm + n^2 \log_2 U)$, las cuales mejoran la complejidad $O(n^2 m)$ del algoritmo de trayectorias más cortas aumentadas y la complejidad $O(m^2 \log_2 U)$ del algoritmo de escalado de capacidades, respectivamente.

Bibliografía

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993): *Network Flows: Theory, Algorithms, and Applications.*, Prentice Hall.

Julio González Díaz (2018-2019): *Programación Lineal y Entera.*