Algoritmo ‘primal-dual’ para el problema de flujo a costo mínimo

Blanca Gómez Sanz

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Director del trabajo: Pedro M. Mateo Collazos
31 de enero de 2024

Summary

In the field of network optimization, the Minimum Cost Flow Problem (MCFP) stands out as an important problem with many applications. This is about determining the most cost-effective way to transport goods through a network, considering the cost implications of each flow and aiming to minimize overall transportation expenses while taking capacity constraints into account.

In this scenario, networks are used to represent and model real-life systems. Nodes represent locations, and arcs, connections between them. These connections denote feasible paths for transporting goods, with the flow of goods representing the quantity being transported. The associated cost of each arch reflects the expenses incurred.

The goal of the MCFP is to find the optimal flow configuration that minimizes the total transportation cost. This involves determining the flow quantity along each arc to satisfy demand while considering cost constraints. The significance of MCFP extends beyond logistics, reaching into diverse fields such as power supply networks and financial management. In the field of power supply, it aids in optimizing the flow of electricity, ensuring efficient energy distribution. In finances, its application offers insights into optimizing cash flow and maximizing financial efficiency across various scenarios. In logistics, the MCFP is used to design efficient supply chain routes, ensuring timely deliveries at a minimal cost. Its impact is even more pronounced in transportation networks, where it aids in traffic management, optimizing the movement of vehicles to reduce congestion. Additionally, the MCFP is also employed in communication networks, guiding data transmission routes for optimal resource utilization.

To solve the MCFP, we delve into the development of the 'primal-dual' algorithm. This algorithm employs a dual ascent approach, adjusting primal and dual variables iteratively to converge towards an optimal solution. Our study shows that the dual problem consistently possesses a feasible solution for any set of associated values of a certain subset of the dual variables, a critical insight for the algorithmic development.

Let us summarize the content of this work, which has been structured in 3 chapters.

It begins in Chapter 1 by introducing the minimum cost flow problem along with relevant concepts. Then, the associated dual problem is defined, and a result is highlighted, asserting that the dual problem always possesses a feasible solution for any set of w_i values. Additionally, the Complementary Slackness Theorem, explored in the Operations Research course [3], is recalled. Finally, a graphical representation of the Complementary Slackness Conditions is provided.

Moving to the second Chapter, concepts and outcomes related to dual ascent algorithms are examined to ultimately define and present the 'primal-dual' algorithm for the Minimum Cost Flow Problem. After the theoretical overview, a detailed step-by-step example is presented. Furthermore, a Python code is developed from scratch using this algorithm, which is later utilized in the subsequent chapter and provided in Appendix A.

In Chapter 3, a comprehensive study of the algorithm is carried out, involving three configurations

where each modification targets a specific element. The first configuration adjusts the number of arcs, the second modifies number of nodes with supply/demand, and the third alters minimum and maximum capacities. With the help of R and RCommander, we evaluate the average resolution time of the algorithm using statistical tools like ANOVA and Tukey's test, and also we represent the results graphically. For that, some simple R-scripts, provided in Appendix B, have been elaborated.

In summary, the purpose of this document has been to present and study in depth the Minimum Cost Flow Problem along with its 'primal-dual' algorithm for resolution. The work also includes an implementation in Python of the algorithm and a computational study to gain a more detailed understanding of the algorithm's behavior.

For the development of this work, I have used the knowledge acquired during my Mathematics degree in courses such as Graphs and Combinatorics, Computer Science I, Mathematical Statistics, and, above all, in Operations Research in the 3rd year.

Índice general

Summary III

1. Introducción **1**
1.1. El problema de flujo a costo mínimo 1
1.2. El problema dual del PFCM 2
1.3. Las condiciones de la holgura complementaria (CHC) 4

2. Algoritmo 'primal-dual' **7**
2.1. Ascenso Dual 7
2.2. Descripción general del algoritmo de ascenso dual 11
2.3. Algoritmo primal-dual para el PFCM 12

3. Estudio computacional del algoritmo **21**

Bibliografía **27**

Anexos **29**

A. Algoritmo 'primal-dual' **29**

B. Script utilizado en R **37**
B.1. Cálculo de la media y la desviación típica del tiempo de ejecución 37
B.2. Test de normalidad 40
B.3. Test de Levene 40
B.4. Test de ANOVA con aproximación de Welch 41
B.5. Test de Tukey 41

Capítulo 1

Introducción

1.1. El problema de flujo a costo mínimo

Una *red de flujo* [1] es un grafo dirigido $G = (N,A)$ con $N = \{1,...,n\}$ un conjunto de nodos conectados mediante un conjunto de arcos $A = \{(i,j)|i \in N, j \in N\} \subseteq N \times N$, representando por (i,j) al arco que conecta el nodo i con el nodo j. En una red, un cierto ítem es enviado de nodo a nodo a través de los arcos. Nos podemos encontrar con dos tipos de nodos: aquellos que tienen oferta y asociamos con flujo positivo, conocidos como *fuente*, y otros que tienen demanda y asociamos con flujo negativo, los *sumideros*.

El problema de flujo a costo mínimo (PFCM) sobre una red de flujo viene definido por:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{sujeto a} \quad & \sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = b_i, \quad \forall i \in N, \\ & l_{ij} \leq x_{ij} \leq u_{ij}, \qquad\qquad \forall (i,j) \in A \end{aligned} \tag{1.1}$$

donde c_{ij}, l_{ij}, u_{ij}, y b_{ij}, $\forall (i,j) \in A$, son escalares dados y su significado es el siguiente:

- x_{ij}: el *flujo*, la cantidad enviada desde el nodo i al nodo j
- c_{ij}: el *coeficiente de costo* (o simplemente *costo*) de (i,j), costo de envío de una unidad de flujo desde el nodo i al nodo j
- l_{ij}: el *límite inferior de flujo* de (i,j)
- u_{ij}: el *límite superior de flujo* de (i,j)
- $[l_{ij},u_{ij}]$: el *rango de flujo factible* de (i,j), el cual nos dice la *capacidad* del arco (i,j)

[1] Para el estudio y desarrollo del algoritmo he utilizado básicamente las referencias [1] y [2]. He utilizado [1] principalmente para el establecimiento de la notación y elementos del PFCM, así como para el desarrollo de los elementos de dualidad necesarios en el Capítulo 2. En [1] se presentan para un PFCM particular denominado circulación con costo mínimo y yo lo he adaptado a mi problema, el problema de flujo a costo mínimo, utilizando además los conocimientos adquiridos sobre Teoría de la dualidad en PL en la asignatura de Investigación Operativa del grado [3]. El desarrollo del algoritmo primal-dual se ha obtenido del capítulo 3 de la segunda referencia y en el cual he incluido algún desarrollo que en este se dejaban para el lector o se realizaban de forma diferente (cálculo del gradiente de la función objetivo del problema dual, teorema de convergencia del algoritmo). En cuanto a las otras referencias, [4] ha sido utilizada para la implementación de los códigos del algoritmo, [5] para los scripts de R con los que se han realizado el estudio computacional, y de [6] se ha sacado la guía para la aplicación de las técnicas del ANOVA.

- b_i: el *suministro* del nodo i
  $b_i > 0$ nodo con oferta (fuente)
  $b_i < 0$ nodo con demanda (sumidero)
  $b_i = 0$ nodo de transbordo

El objetivo del PFCM es minimizar la *función objetivo* $\sum_{(i,j)\in A} c_{ij}x_{ij}$ sujeto a las restricciones de (1.1), es decir, tratar de satisfacer la demanda de los nodos demandantes con la oferta disponible en los ofertantes de forma que el flujo enviado por cada arco respete sus cotas y el envío tenga el costo total mínimo. Las primeras se denominan *restricciones de conservación de flujo*, mientras que las segundas son las *restricciones de capacidad*. Un vector de flujo que satisface ambas restricciones se denomina *factible*, y si satisface solo las restricciones de capacidad, se denomina *factible en capacidad*. Si existe al menos un vector de flujo factible, el problema (PFCM) se dirá *factible*; de lo contrario se dirá *no factible*. Notar que una condición necesaria para la factibilidad es que

$$\sum_{i\in N} b_i = 0, \tag{1.2}$$

Hemos presentado el Problema de Flujo a Costo Mínimo (PFCM) general, pero podemos aplicarlo a casos particulares, como son los conocidos: *Problema de transporte*, *Problema de asignación* y *Problema de transbordo*, entre otros.

Vamos a mostrar gráficamente un sencillo ejemplo de red de flujo, Figura 1.1, en el que tenemos un nodo (1) con 4 unidades de oferta, un nodo (4) con 4 unidades de demanda:

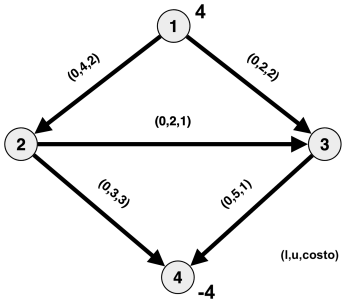


Figura 1.1: Ejemplo de red de flujo.

En cada arco dirigido tenemos *(l,u,costo)*, donde l corresponde con el límite inferior de flujo, u con el límite superior de flujo, y *costo* con el costo de envío de una unidad de flujo por el arco (i, j). Por ejemplo, en el arco (2,3) que va del nodo (2) al nodo (3) podemos mandar de 0 hasta 2 unidades, con costo por unidad de 1.

1.2. El problema dual del PFCM

El problema (1.1) es un problema de programación lineal y por tanto se puede definir su problema dual asociado. [2]

Si asociamos una variable dual w_i a cada ecuación de conservación de flujo de cada nodo en el Problema (1.1), una variable dual h_{ij} a cada restricción $x_{ij} \leq u_{ij}$ (puesta en la forma $-x_{ij} \geq -u_{ij}$), y una variable dual v_{ij} con cada restricción $x_{ij} \geq l_{ij}$, el problema dual asociado al problema de flujo a costo mínimo es:

[2]El caso general de dualidad para Problemas de Programación Lineal (PPL) se estudió en la asignatura Investigación Operativa de 3^{er} curso.

$$\begin{array}{lll} \text{Maximizar} & \displaystyle\sum_{i\in N} w_i\, b_i + \sum_{(i,j)\in A} l_{ij}\, v_{ij} - \sum_{(i,j)\in A} u_{ij}\, h_{ij} & \\ \text{sujeto a} & w_i - w_j + v_{ij} - h_{ij} = c_{ij}, & \forall (i,j) \in A \\ & h_{ij}, v_{ij} \geq 0, & \forall (i,j) \in A \\ & w_i \text{ no restringido}, & \forall i \in N \end{array} \tag{1.3}$$

La estructura de este problema nos permite obtener los siguientes resultados que son de gran importancia.

Lema 1.2.1. El problema dual siempre posee una solución factible dado cualquier conjunto de valores de w_i. De hecho, las elecciones de v_{ij} y h_{ij} que se proporcionan en la demostración producen valores óptimos de v_{ij} y h_{ij} para un conjunto fijo de valores de w_i.

Demostración. Supongamos que se selecciona un conjunto arbitrario de valores para w_i, asumiendo que estos valores son enteros. Además, consideremos que los valores l_{ij} y u_{ij} son fijos. Luego, la restricción dual para el arco (i,j) se expresa como:

$$v_{ij} - h_{ij} = c_{ij} - w_i + w_j, \quad h_{ij} \geq 0, \quad v_{ij} \geq 0$$

Estas restricciones se satisfacen seleccionando v_{ij} y h_{ij} de la siguiente manera:

$$\begin{array}{l} v_{ij} = \text{máx}\left\{0,\, c_{ij} - w_i + w_j\right\}, \\ h_{ij} = \text{máx}\left\{0,\, -(c_{ij} - w_i + w_j)\right\} \end{array} \tag{1.4}$$

Si $c_{ij} - w_i + w_j$ es positivo, asignamos a v_{ij} esa cantidad. Si $c_{ij} - w_i + w_j$ es negativo, le cambio el signo, $-(c_{ij} - w_i + w_j)$, y asignamos a h_{ij} esa cantidad.

Ahora, vamos a demostrar que estas elecciones son óptimas para los distintos casos:

Sea $(i,j) \in A$ con $c_{ij} - w_i + w_j > 0$, entonces $v_{ij} = c_{ij} - w_i + w_j + h_{ij}$. Sustituyendo esto en su sumando de la función objetivo obtenemos que la aportación de $[ij]$ en la función objetivo dual es:

$$\begin{array}{l} l_{ij}\,(c_{ij} - w_i + w_j + h_{ij}) - \, u_{ij}\, h_{ij} \\ = l_{ij}\,(c_{ij} - w_i + w_j) + \, l_{ij}\, h_{ij} - \, u_{ij}\, h_{ij} \\ = l_{ij}\,(c_{ij} - w_i + w_j) + \, h_{ij}\,(l_{ij} - u_{ij}) \end{array}$$

Dado que $c_{ij} - w_i + w_j > 0$ y $l_{ij} \leq u_{ij}$ $(l_{ij} - u_{ij} \leq 0)$, y el objetivo es maximizar la función objetivo con $h_{ij} \geq 0$ y $v_{ij} \geq 0$, es óptimo que $h_{ij} = 0$, lo más pequeño posible y, por lo tanto, $v_{ij} = c_{ij} - w_i + w_j$, lo más grande posible.

Del mismo modo, sea $(i,j) \in A$ con $c_{ij} - w_i + w_j < 0$, entonces $h_{ij} = -(c_{ij} - w_i + w_j - v_{ij})$. Sustituyendo de nuevo,

$$\begin{array}{l} l_{ij}\, v_{ij} \, - [-(c_{ij} - w_i + w_j - v_{ij})\, u_{ij}] \\ = l_{ij}\, v_{ij} \, - \, u_{ij}\, v_{ij} \, - [-(c_{ij} - w_i + w_j)\, u_{ij}] \\ = (l_{ij} - u_{ij})\, v_{ij} \, - [-(c_{ij} - w_i + w_j)\, u_{ij}] \end{array}$$

Dado que $c_{ij} - w_i + w_j < 0$ (luego $-(c_{ij} - w_i + w_j) > 0$) y $l_{ij} \leq u_{ij}$ $(l_{ij} - u_{ij} \leq 0)$, y el objetivo es maximizar la función objetivo sujeta a $v_{ij} \geq 0$ y $h_{ij} \geq 0$, es óptimo que $v_{ij} = 0$, lo más pequeño posible,

lo que conduce a $h_{ij} = -(c_{ij} - w_i + w_j)$.

Por último, si $c_{ij} - w_i + w_j = 0$, entonces $v_{ij} = h_{ij}$. En este caso, la aportación a la función objetivo se reduce a

$$\begin{aligned} & l_{ij}\, h_{ij} - \ u_{ij}\, h_{ij} \\ & = (l_{ij} - u_{ij})\, h_{ij} \end{aligned}$$

y como $l_{ij} - u_{ij} \leq 0$ y queremos maximizar, la elección apropiada es $h_{ij} = 0$, y por tanto, $v_{ij} = 0$.

En resumen, las elecciones v_{ij} y h_{ij} presentadas en la demostración generan las mejores soluciones factibles para cualquier elección previa de w_i. □

1.3. Las condiciones de la holgura complementaria (CHC)

Vamos a recordar unos resultados estudiados en la asignatura de Investigación Operativa para luego particularizarlos:

Teorema 1.3.1 (Teorema de la holgura complementaria)**.** Sean $\bar{\mathbf{x}}$ y $\bar{\mathbf{w}}$, respectivamente, soluciones factibles de los problemas primal y dual siguientes

Problema primal		Problema dual	
max	$\mathbf{c}\,\mathbf{x}$	min	$\mathbf{b}'\mathbf{w}$
sujeto a	$A\,\mathbf{x} \leq \mathbf{b}$	sujeto a	$A'\mathbf{w} \geq \mathbf{c}'$
	$\mathbf{x} \geq \mathbf{0}$		$\mathbf{w} \geq \mathbf{0}$

donde $\mathbf{c}$ es $1xn$, A es mxn, $\mathbf{b}$ es $mx1$, $\mathbf{x}$ es $nx1$, y $\mathbf{w}$ es $mx1$.
Dichas soluciones son óptimas si y sólo si

$$\bar{\mathbf{w}}'(\mathbf{b} - A\bar{\mathbf{x}}) + (\bar{\mathbf{w}}'A - \mathbf{c})\bar{\mathbf{x}} = 0 \tag{1.5}$$

Definiendo $\bar{u}_i,\ i = 1,...,m$, los valores de las variables de holgura del problema primal asociados a la solución $\bar{\mathbf{x}}$, y $\bar{v}_i,\ j = 1,...,n$, los valores de las variables de holgura del problema dual asociados a la solución $\bar{\mathbf{w}}$, la condición del teorema anterior puede reescribirse:

$$0 = \bar{\mathbf{w}}'(\mathbf{b} - A\bar{\mathbf{x}}) + (\bar{\mathbf{w}}'A - \mathbf{c})\bar{\mathbf{x}} = \bar{\mathbf{w}}'\ \bar{\mathbf{u}} + \bar{\mathbf{v}}\ \bar{\mathbf{x}} \tag{1.6}$$

de donde obtenemos que $\bar{\mathbf{w}}'\ \bar{\mathbf{u}} = 0$ y $\bar{\mathbf{v}}\ \bar{\mathbf{x}} = 0$.
Lo podemos expresar de manera equivalente, siendo esta la forma en la que lo utilizaremos:

$$\begin{aligned} \bar{x}_j\ \bar{v}_j &= 0, \quad j = 1,\ldots,n, \\ \bar{w}_i\ \bar{u}_i &= 0, \quad i = 1,\ldots,m \end{aligned} \tag{1.7}$$

A continuación vamos a desarrollar estas condiciones para el PFCM:

Notar que tenemos que el dual tiene todas las restricciones de igualdad, con lo cual no hay holguras y por tanto nos olvidamos de la condición $\bar{w}_i\ \bar{u}_i = 0$ ya que se cumple trivialmente. Luego las condiciones de la holgura complementaria para la optimización del problema de flujo a costo mínimo se reducen a:

$$\begin{aligned} (x_{ij} - l_{ij})\ v_{ij} &= 0, \quad \forall (i,j) \in A \\ (u_{ij} - x_{ij})\ h_{ij} &= 0, \quad \forall (i,j) \in A \end{aligned} \tag{1.8}$$

Por tanto, vamos a trabajar con las condiciones de holgura complementaria restringidas a las w_i y a la definición dada de v_{ij} y h_{ij} en (1.4) de manera que en lugar de utilizar las v_{ij} y h_{ij}, escribimos $(c_{ij} - w_i + w_j)$ cambiado de signo o no, dependiendo de lo que corresponda. De esta forma, las CHC quedarán:

$$\begin{aligned} &\text{si } c_{ij} - w_i + w_j > 0 \Rightarrow v_{ij} > 0 \Rightarrow x_{ij} = l_{ij}, && \forall (i,j) \in A \\ &\text{si } c_{ij} - w_i + w_j < 0 \Rightarrow h_{ij} > 0 \Rightarrow x_{ij} = u_{ij}, && \forall (i,j) \in A \\ &\text{si } c_{ij} - w_i + w_j = 0 \Rightarrow v_{ij} = h_{ij} = 0 \Rightarrow l_{ij} \leq x_{ij} \leq u_{ij}, && \forall (i,j) \in A \end{aligned} \tag{1.9}$$

La tercera ecuación solo refleja el hecho de que en ese tercer caso, que cumple trivialmente (1.8), el flujo respeta las restricciones de cota.
A partir de ahora, clasificaremos los arcos en tres tipos,

$$\begin{aligned} &\text{inactivo si} && c_{ij} - w_i + w_j > 0, \\ &\text{balanceado si} && c_{ij} - w_i + w_j = 0, \\ &\text{activo si} && c_{ij} - w_i + w_j < 0. \end{aligned}$$

Esta es la notación que luego utilizaremos en el desarrollo del algoritmo.

Sabemos que si tenemos un conjunto de soluciones x_{ij} y w_i factibles que verifican las restricciones de sus respectivos problemas, y se cumplen las condiciones de holgura complementaria, entonces ambas soluciones son óptimas para sus problemas respectivos; esto lo enunciamos en el siguiente teorema cuya demostración son los desarrollos anteriores.

Teorema 1.3.2. Sea **x** un flujo factible de (1.1), y sea $\mathbf{w} = (w_1, ..., w_n)$ un conjunto de valores $w_j \geq 0, j = 1, ..., n$. Entonces **x** y (**w**, **v**, **h**) con **v** y **h** construidos de acuerdo a (1.4) son, respectivamente, soluciones óptimas del problema de flujo a costo mínimo y su dual si y sólo si para todo $(i,j) \in A$ se cumple:

$$\begin{aligned} &\text{si } c_{ij} - w_i + w_j > 0 \text{ (}\textit{arco inactivo}\text{) entonces } x_{ij} = l_{ij}, \\ &\text{si } c_{ij} - w_i + w_j = 0 \text{ (}\textit{arco balanceado}\text{) entonces } l_{ij} \leq x_{ij} \leq u_{ij}, \\ &\text{si } c_{ij} - w_i + w_j < 0 \text{ (}\textit{arco activo}\text{) entonces } x_{ij} = u_{ij} \end{aligned}$$

□

Una forma de entender y representar estos resultados gráficamente sería mediante la silla de Kilter[3] (Figura 1.2).

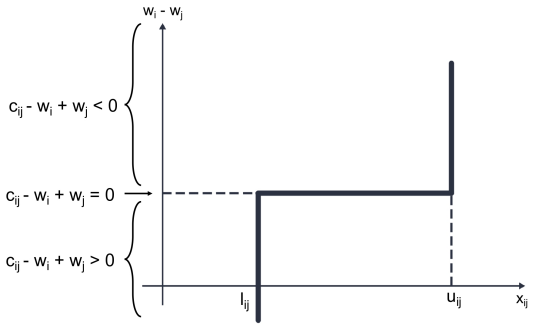


Figura 1.2: Ilustración de las Condiciones de la Holgura Complementaria. Un arco cumple las CHC si el punto $(x_{ij}, w_i - w_j)$ se encuentra sobre la línea gruesa de la función.

[3]Dicho nombre hace referencia a otro algoritmo para el PFCM para circulaciones ($b_i = 0, \forall i = 1, ..., n$) denominado algorithm "out–of–kilter".

Capítulo 2

Algoritmo 'primal-dual'

2.1. Ascenso Dual

La mayoría de los métodos para hallar un óptimo en PFCM son por búsqueda direccional: partiendo de un punto, se define una dirección de búsqueda y se determina un nuevo punto a lo largo de ésta que mejora la función objetivo.
Por ese motivo, los principales algoritmos de ascenso dual se basan en construir una dirección $\mathbf{d} \neq \mathbf{0}$ tal que a partir del $\mathbf{w}$ actual moverse $\mathbf{w} + \alpha_0 \mathbf{d}$, $\alpha_0 > 0$ de forma que $\mathbf{w} + \alpha_0 \mathbf{d}$ tenga mayor valor de la función objetivo del dual que $\mathbf{w}$. Para ello, en nuestro algoritmo, se seleccionan en cada iteración un subconjunto conectado de nodos $S \subseteq N$ y se cambian los precios de estos nodos (es decir, los valores de las variables duales w_i) en cantidades iguales, dejando los precios de todos los demás nodos sin cambios. En otras palabras, cada iteración implica un cambio en el vector de precios a lo largo de una dirección de la forma $\mathbf{d_S} = (d_1, ..., d_N)$, donde

$$d_i = \begin{cases} 1 & \text{si } i \in S \\ 0 & \text{si } i \notin S \end{cases} \tag{2.1}$$

y S es un subconjunto conectado de nodos. Estas direcciones $\mathbf{d_S}$ se llamarán *direcciones elementales* y generarán nuevas soluciones del dual de la forma $\mathbf{w} + \alpha \mathbf{d_S}$ con $\alpha \geq 0$.

Para comprobar si $\mathbf{d_S}$ es una dirección de ascenso dual, necesitamos calcular la derivada direccional correspondiente del costo dual a lo largo de $\mathbf{d_S}$ y comprobar si es positiva. De la expresión de la función objetivo dual $q(\mathbf{w}) = \sum_{i \in N} w_i \, b_i + \sum_{(i,j) \in A} l_{ij} \, v_{ij} - \sum_{(i,j) \in A} u_{ij} \, h_{ij}$ y teniendo en cuenta la definición de las variables h_{ij} y v_{ij}, calculamos su derivada direccional:

$$q'(\mathbf{w}; \mathbf{d_S}) = \lim_{\alpha \downarrow 0} \frac{q(\mathbf{w} + \alpha \mathbf{d_S}) - q(\mathbf{w})}{\alpha}$$

Para facilitar las cuentas, voy a recuadrar del mismo color los sumandos de $q(\mathbf{w} + \alpha \mathbf{d_S})$ y de $q(\mathbf{w})$ que luego restaré y darán $q(\mathbf{w} + \alpha \mathbf{d_S}) - q(\mathbf{w})$, utilizando en esta última también los mismos colores. Los sumandos de $q(\mathbf{w} + \alpha \mathbf{d_S})$ y $q(\mathbf{w})$ que al hacer la diferencia dan igual a 0 no los he recuadrado, ya que no juegan un papel en el resultado final de $q(\mathbf{w} + \alpha \mathbf{d_S}) - q(\mathbf{w})$.

Primero, calculamos el sumando $q(\mathbf{w} + \alpha \mathbf{d_S})$ siendo $\mathbf{d_S} = (d_1, d_2, ..., d_N)$ el vector con $d_i = 1$ si $i \in S$ y $d_i = 0$ en caso contrario. Para ello, vamos a separar los distintos casos posibles, dependiendo de si $i \in S$

y $j \in S$, y dependiendo de si se trata de arcos activos, inactivos o balanceados:

$$
\begin{aligned}
q(\mathbf{w}+\alpha\,\mathbf{d_S}) = & \sum_{i\in S}(w_i+\alpha)\,b_i + \sum_{i\notin S} w_i\,b_i \\
& + \sum_{\substack{i\in S,\ j\notin S \\ c_{ij}-w_i+w_j>0}} (c_{ij}-w_i-\alpha+w_j)\,l_{ij} + \sum_{\substack{j\notin S,\ i\in S \\ c_{ji}-w_j+w_i>0}} (c_{ji}-w_j+w_i+\alpha)\,l_{ji} \\
& + \sum_{\substack{i\in S,\ j\notin S \\ c_{ij}-w_i+w_j<0}} (c_{ij}-w_i-\alpha+w_j)\,u_{ij} + \sum_{\substack{j\notin S,\ i\in S \\ c_{ji}-w_j+w_i<0}} (c_{ji}-w_j+w_i+\alpha)\,u_{ji} \\
& + \sum_{\substack{i\in S,\ j\in S \\ i\notin S,\ j\notin S \\ c_{ij}-w_i+w_j>0}} (c_{ij}-w_i+w_j)\,l_{ij} + \sum_{\substack{i\in S,\ j\in S \\ i\notin S,\ j\notin S \\ c_{ij}-w_i+w_j<0}} (c_{ij}-w_i+w_j)\,u_{ij} \\
& + \sum_{\substack{j\in S,\ i\in S \\ j\notin S,\ i\notin S \\ c_{ji}-w_j+w_i>0}} (c_{ji}-w_j+w_i)\,l_{ji} + \sum_{\substack{j\in S,\ i\in S \\ j\notin S,\ i\notin S \\ c_{ji}-w_j+w_i<0}} (c_{ji}-w_j+w_i)\,u_{ji} \\
& + \sum_{\substack{i\in S,\ j\notin S \\ c_{ij}-w_i+w_j=0}} (c_{ij}-w_i-\alpha+w_j)\,u_{ij} + \sum_{\substack{j\notin S,\ i\in S \\ c_{ji}-w_j+w_i=0}} (c_{ji}-w_j+w_i+\alpha)\,l_{ji} \\
& + \sum_{\substack{i\in S,\ j\in S \\ i\notin S,\ j\notin S \\ c_{ij}-w_i+w_j=0}} (c_{ij}-w_i+w_j)\,l_{ij}
\end{aligned}
$$

Notar que los dos primeros sumatorios podemos reagruparlos de la siguiente forma para usarlos en la resta:

$$\sum_{i\in S}(w_i+\alpha)\,b_i + \sum_{i\notin S} w_i\,b_i = \sum_{i\in N} w_i\,b_i + \sum_{i\in S}\alpha\,b_i$$

A continuación vamos a calcular la función objetivo del dual $q(\mathbf{w})$, distinguiendo también los distintos casos posibles:

$$
\begin{aligned}
q(\mathbf{w}) = & \sum_{i\in N} w_i\,b_i + \sum_{\substack{i\in S,\ j\notin S \\ c_{ij}-w_i+w_j>0}} (c_{ij}-w_i+w_j)\,l_{ij} + \sum_{\substack{j\notin S,\ i\in S \\ c_{ji}-w_j+w_i>0}} (c_{ji}-w_j+w_i)\,l_{ji} \\
& + \sum_{\substack{i\in S,\ j\notin S \\ c_{ij}-w_i+w_j<0}} (c_{ij}-w_i+w_j)\,u_{ij} + \sum_{\substack{j\notin S,\ i\in S \\ c_{ji}-w_j+w_i<0}} (c_{ji}-w_j+w_i)\,u_{ji} \\
& + \sum_{\substack{i\in S,\ j\in S \\ i\notin S,\ j\notin S \\ c_{ij}-w_i+w_j>0}} (c_{ij}-w_i+w_j)\,l_{ij} + \sum_{\substack{i\in S,\ j\in S \\ i\notin S,\ j\notin S \\ c_{ij}-w_i+w_j<0}} (c_{ij}-w_i+w_j)\,u_{ij} \\
& + \sum_{\substack{j\in S,\ i\in S \\ j\notin S,\ i\notin S \\ c_{ji}-w_j+w_i>0}} (c_{ji}-w_j+w_i)\,l_{ji} + \sum_{\substack{j\in S,\ i\in S \\ j\notin S,\ i\notin S \\ c_{ji}-w_j+w_i<0}} (c_{ji}-w_j+w_i)\,u_{ji} \\
& + \sum_{\substack{i\in S,\ j\notin S \\ c_{ij}-w_i+w_j=0}} (c_{ij}-w_i+w_j)\,u_{ij} + \sum_{\substack{j\notin S,\ i\in S \\ c_{ji}-w_j+w_i=0}} (c_{ji}-w_j+w_i)\,l_{ji} \\
& + \sum_{\substack{i\in S,\ j\in S \\ i\notin S,\ j\notin S \\ c_{ij}-w_i+w_j=0}} (c_{ij}-w_i+w_j)\,l_{ij}
\end{aligned}
$$

Finalmente, calculamos la diferencia:

$$
\begin{aligned}
q(\mathbf{w}+\alpha\,\mathbf{d_S})-q(\mathbf{w}) = & \sum_{\substack{j\notin S,\, i\in S \\ c_{ji}-w_j+w_i<0}} \alpha\, u_{ji} + \sum_{\substack{j\notin S,\, i\in S \\ c_{ji}-w_j+w_i>0}} \alpha\, l_{ji} + \sum_{\substack{j\notin S,\, i\in S \\ c_{ji}-w_j+w_i=0}} \alpha\, l_{ji} \\
& - \sum_{\substack{i\in S,\, j\notin S \\ c_{ij}-w_i+w_j<0}} \alpha\, u_{ij} - \sum_{\substack{i\in S,\, j\notin S \\ c_{ij}-w_i+w_j=0}} \alpha\, u_{ij} - \sum_{\substack{i\in S,\, j\notin S \\ c_{ij}-w_i+w_j>0}} \alpha\, l_{ij} \\
& + \sum_{i\in N} \alpha\, b_i
\end{aligned}
$$

Dividiendo esta expresión entre α y teniendo en cuenta que: si $c_{ij}-w_i+w_j<0$ (o $c_{ji}-w_j+w_i<0$) es activo; si $c_{ij}-w_i+w_j>0$ (o $c_{ji}-w_j+w_i>0$) es inactivo; y si $c_{ij}-w_i+w_j=0$ (o $c_{ji}-w_j+w_i=0$) es balanceado, tenemos

$$
\begin{aligned}
\frac{q(\mathbf{w}+\alpha\,\mathbf{d_S})-q(\mathbf{w})}{\alpha} = & \sum_{\substack{j\notin S,\, i\in S \\ \text{activo}}} u_{ji} + \sum_{\substack{j\notin S,\, i\in S \\ \text{inactivo}}} l_{ji} + \sum_{\substack{j\notin S,\, i\in S \\ \text{balanceado}}} l_{ji} \\
& - \sum_{\substack{i\in S,\, j\notin S \\ \text{activo}}} u_{ij} - \sum_{\substack{i\in S,\, j\notin S \\ \text{balanceado}}} u_{ij} - \sum_{\substack{i\in S,\, j\notin S \\ \text{inactivo}}} l_{ij} \\
& + \sum_{i\in N} b_i \\
= & \sum_{(j,i)\,:\,\text{activo},\; j\notin S,\, i\in S} u_{ji} + \sum_{(j,i)\,:\,\text{inactivo o balanceado},\; j\notin S,\, i\in S} l_{ji} \\
& - \sum_{(i,j)\,:\,\text{activo o balanceado},\; i\in S,\, j\notin S} u_{ij} - \sum_{(i,j)\,:\,\text{inactivo},\; i\in S,\, j\notin S} l_{ij} \\
& + \sum_{i\in S} b_i
\end{aligned}
$$

Finalmente, aplicando el límite cuando $\alpha \to 0$ obtenemos la expresión de la derivada direccional.

$$
\begin{aligned}
q'(\mathbf{w};\mathbf{d_S}) = & \lim_{\alpha\downarrow 0} \frac{q(\mathbf{w}+\alpha\,\mathbf{d_S})-q(\mathbf{w})}{\alpha} \\
= & \sum_{(j,i)\,:\,\text{activo},\; j\notin S,\, i\in S} u_{ji} + \sum_{(j,i)\,:\,\text{inactivo o balanceado},\; j\notin S,\, i\in S} l_{ji} \\
& - \sum_{(i,j)\,:\,\text{activo o balanceado},\; i\in S,\, j\notin S} u_{ij} - \sum_{(i,j)\,:\,\text{inactivo},\; i\in S,\, j\notin S} l_{ij} \\
& + \sum_{i\in S} b_i
\end{aligned}
\tag{2.2}
$$

Esto es, la derivada direccional $q'(\mathbf{w};\mathbf{d_S})$ es la diferencia entre el flujo de entrada y salida a través del conjunto de nodos S cuando los flujos de los arcos inactivos y activos se establecen en sus límites inferior y superior, respectivamente, y el flujo de cada arco balanceado incidente en S se establece en su límite inferior o superior dependiendo de si el arco entra a S o sale de S.

Para obtener un conjunto adecuado S, con derivada direccional positiva $q'(\mathbf{w};\mathbf{d_S})$, es conveniente mantener un vector de flujo $\mathbf{x}$ que satisfaga las CHC junto con $\mathbf{w}$. Esto ayuda a organizar la búsqueda de una dirección de ascenso y a detectar la optimización, como se explicará a continuación.

Para un vector de flujo $\mathbf{x}$, definimos el *exceso* g_i del nodo i como la diferencia entre el flujo total de entrada a i menos el flujo total de salida de i, es decir,

$$g_i = \sum_{\{j|(j,i)\in A\}} x_{ji} - \sum_{\{j|(i,j)\in A\}} x_{ij} + b_i \tag{2.3}$$

y tenemos que el *exceso total* de un conjunto de nodos S es

$$\sum_{i\in S} g_i = \sum_{\{(j,i)\in A | j\notin S, i\in S\}} x_{ji} - \sum_{\{(i,j)\in A | i\in S, j\notin S\}} x_{ij} + \sum_{i\in S} b_i \tag{2.4}$$

Obsérvese que si $j \in S$ e $i \in S$ entonces aparece x_{ji} en el primer sumando de g_i y x_{ji} en el segundo sumando de g_j y se cancelan, por lo que solo quedan en (2.4) arcos (i,j) y (j,i) con $i \in S$ y $j \notin S$

Si $\mathbf{x}$ satisface las CHC junto con $\mathbf{w}$, es decir, se cumple el Teorema (1.3.2) [$x_{ij} = u_{ij}$ si es activo; $x_{ij} = l_{ij}$ si es inactivo; $l_{ij} \le x_{ij} \le u_{ij}$ si es balanceado], podemos reescribir (2.4) como

$$\begin{aligned}\sum_{i\in S} g_i = & \sum_{(j,i)\,:\,\text{activo},\ j\notin S,\ i\in S} u_{ji} + \sum_{(j,i)\,:\,\text{inactivo},\ j\notin S,\ i\in S} l_{ji} + \sum_{(j,i)\,:\,\text{balanceado},\ j\notin S,\ i\in S} x_{ji} \\ & - \left[\sum_{(i,j)\,:\,\text{activo},\ i\in S,\ j\notin S} u_{ij} + \sum_{(i,j)\,:\,\text{inactivo},\ i\in S,\ j\notin S} l_{ij} + \sum_{(i,j)\,:\,\text{balanceado},\ i\in S,\ j\notin S} x_{ij} \right] \\ & + \sum_{i\in S} b_i\end{aligned} \tag{2.5}$$

Despejando $\sum_{i\in S} b_i$ de (2.2) y sustituyendo con este en (2.5) obtenemos

$$\begin{aligned}\sum_{i\in S} g_i &= q'(\mathbf{w};\mathbf{d_S}) + \sum_{(j,i)\,:\,\text{balanceado},\ j\notin S,\ i\in S} (x_{ji} - l_{ji}) \\ &\qquad + \sum_{(i,j)\,:\,\text{balanceado},\ i\in S,\ j\notin S} (u_{ij} - x_{ij}) \\ &\ge q'(\mathbf{w};\mathbf{d_S})\end{aligned} \tag{2.6}$$

Vemos, por lo tanto, que sólo un conjunto de nodos S que tiene exceso total positivo puede ser candidato para generar una dirección $\mathbf{d_S}$ de ascenso dual, porque en caso contrario, por la desigualdad anterior, si la suma de las g_i es negativa, la derivada direccional $q'(\mathbf{w};\mathbf{d_S})$ sería menor o igual que algo negativo y sería una dirección de descenso en lugar de ascenso. En particular, si no hay un arco balanceado (i,j) con $i \in S,\ j \notin S$, y $x_{ij} < u_{ij}$ (es decir, si $\sum_{(i,j)\,:\,\text{balanceado},\ i\in S,\ j\notin S} (u_{ij} - x_{ij}) = 0$), y no hay un arco balanceado (j,i) con $j \notin S,\ i \in S$, y $l_{ji} < x_{ji}$ (es decir, si $\sum_{(j,i)\,:\,\text{balanceado},\ j\notin S,\ i\in S} (x_{ji} - l_{ji}) = 0$), entonces la desigualdad (2.6) queda

$$\sum_{i\in S} g_i = q'(\mathbf{w};\mathbf{d_S}) \tag{2.7}$$

luego si S tiene un exceso total positivo, entonces $\mathbf{d_S}$ es una dirección de ascenso.
El siguiente lema expresa esta idea y proporciona la base para los algoritmos posteriores.

Lema 2.1.1. Supongamos que $\mathbf{x}$ y $\mathbf{w}$ satisfacen las CHC y sea S un subconjunto de nodos. Sea $\mathbf{d_S} = (d_1, d_2, ..., d_N)$ el vector con $d_i = 1$ si $i \in S$ y $d_i = 0$ en caso contrario, y supongamos que

$$\sum_{i\in S} g_i > 0$$

Entonces, $\mathbf{d_S}$ es una dirección de ascenso dual, es decir,

$$q'(\mathbf{w};\mathbf{d_S}) > 0,$$

o existen nodos $i \in S$ y $j \notin S$ tales que (i,j) es un arco balanceado con $x_{ij} < u_{ij}$ o (j,i) es un arco balanceado con $l_{ji} < x_{ji}$.

Demostración. Se sigue de la Ecuación (2.6):

Si $\sum_{i \in S} g_i > 0$ y $q'(\mathbf{w};\mathbf{d_S}) > 0$, $\mathbf{d_S}$ es una dirección de ascenso dual y se cumple el Lema.

En caso contrario, si $\sum_{i \in S} g_i > 0$ pero $q'(\mathbf{w};\mathbf{d_S}) \leq 0$, por (2.6) vemos que se tiene que cumplir la siguiente desigualdad

$$\sum_{(j,i)\,:\,\text{balanceado},\ j \notin S,\ i \in S} (x_{ji} - l_{ji}) + \sum_{(i,j)\,:\,\text{balanceado},\ i \in S,\ j \notin S} (u_{ij} - x_{ij}) > 0$$

Es decir, tienen que existir nodos $i \in S$ y $j \notin S$ tales que (i,j) es un arco balanceado con $x_{ij} < u_{ij}$ o (j,i) es un arco balanceado con $l_{ji} < x_{ji}$; lo que corresponde con la segunda parte del Lema. □

2.2. Descripción general del algoritmo de ascenso dual

Los algoritmos comienzan con un par de vectores enteros flujo-precio $(\mathbf{x},\mathbf{w})$, que satisfacen las CHC, y funcionan realizando una serie de iteraciones. Al comienzo de cada iteración, tendremos un subconjunto de nodos S tal que

$$\sum_{i \in S} g_i > 0;$$

es decir, inicialmente S consta de uno o más nodos con exceso positivo. Según el lema anterior, existen dos posibilidades:

(a) S define una dirección de ascenso dual $\mathbf{d_S} = (d_1, d_2, ..., d_N)$, donde $d_i = 1$ si $i \in S$ y $d_i = 0$ en caso contrario.

(b) S se puede ampliar agregando un nodo $j \notin S$ con la propiedad descrita en el Lema 2.1.1, es decir, para algún $i \in S$, (i,j) es un arco balanceado con $x_{ij} < u_{ij}$, o (j,i) es un arco balanceado con $l_{ji} < x_{ji}$.
En el caso (b), hay dos posibilidades para el nodo j agregado:

(1) $g_j \geq 0$, en cuyo caso,

$$\sum_{i \in S \cup \{j\}} g_i > 0;$$

y el proceso puede continuar con

$$S \leftarrow S \cup \{j\}$$

(2) $g_j < 0$, en cuyo caso, se puede ver que hay un camino que se origina en algún nodo i del conjunto inicial S y termina en el nodo j que no está *bloqueado*, es decir, todos sus arcos tienen espacio para un aumento de flujo en el dirección de i a j. Tal camino se llama *camino de aumento*. Se puede aumentar el flujo de los arcos hacia adelante (dirección de i a j) del camino y disminuir el flujo de los arcos hacia atrás (dirección de j a i) del camino, podemos acercar ambos excesos g_i y g_j a cero sin afectar el exceso de todos los demás nodos y manteniendo las CHC.

Dado que el exceso absoluto total $\sum_{i\in N}|g_i|$ no puede reducirse indefinidamente, se ve que a partir de un par entero de vectores flujo-precio que satisfaga las CHC, después de como máximo un número finito de iteraciones en las que se producen aumentos de flujo sin encontrar una dirección de ascenso, sucederá una de estas tres cosas:

(a) Se encontrará una dirección de ascenso dual; esta dirección se puede utilizar para mejorar el costo dual.

(b) $g_i = 0$ para todo i; en este caso el vector de flujo $\mathbf{x}$ es factible, y dado que satisface las CHC junto con $\mathbf{w}$, según el Teorema 1.3.2, $\mathbf{x}$ es óptimo primal y $\mathbf{w}$ es óptimo dual.

(c) $g_i \leq 0$ para todo i pero $g_i < 0$ para al menos un i; de la ecuación (2.4) tenemos $\sum_{i\in N} b_i = \sum_{i\in N} g_i$, luego $\sum_{i\in N} b_i < 0$, y por consiguiente el problema es no factible.

Por lo tanto, para un problema factible, el procedimiento que acabamos de describir se puede utilizar para encontrar una dirección de ascenso dual y mejorar el costo dual comenzando en cualquier vector de precios no óptimo.

2.3. Algoritmo primal-dual para el PFCM

El algoritmo primal-dual comienza con cualquier par de enteros $(\mathbf{x},\mathbf{w})$ que satisfaga las CHC. Una posibilidad es elegir arbitrariamente el vector entero $\mathbf{w}$ y establecer $x_{ij} = l_{ij}$ si (i,j) está inactivo o balanceado, y $x_{ij} = u_{ij}$ en caso contrario. Otra posibilidad podría ser elegir estos $\mathbf{x}$ y $\mathbf{w}$ basándonos, por ejemplo, en los resultados de una optimización anterior. El algoritmo preserva las CHC del par $(\mathbf{x},\mathbf{w})$ en todo momento.

Al comienzo de la iteración, tenemos un par de enteros $(\mathbf{x},\mathbf{w})$ que satisface las CHC. La iteración indicará: que el problema primal es no factible; o bien indicará que $(\mathbf{x},\mathbf{w})$ es óptimo; o bien transformará este par en otro par que satisfacerá las CHC.
En particular, si $g_i \leq 0$ para todo i, entonces teniendo en cuenta $\sum_{i\in N} b_i = \sum_{i\in N} g_i$ (por la Ecuación (2.4) con $S = N$), hay dos posibilidades:

(1) $g_i < 0$ para algún i, en cuyo caso $\sum_{i\in N} b_i < 0$ y el problema es no factible.

(2) $g_i = 0$ para todo i, en cuyo caso $\mathbf{x}$ es factible y por lo tanto también es óptimo, ya que satisface CHC junto con $\mathbf{w}$.

En cualquiera de los dos casos, el algoritmo termina.

Si por otro lado tenemos $g_i > 0$ para al menos un nodo i, la iteración comienza seleccionando un subconjunto I no vacío de nodos i con $g_i > 0$. La iteración mantiene dos conjuntos de nodos S y L, con $S \subset L$. Inicialmente, S está vacío y L consta del subconjunto I.

Vamos a usar la siguiente terminología:

S : Conjunto de *nodos escaneados* (son los nodos cuyos arcos incidentes se han "examinado" durante la iteración).

L : Conjunto de *nodos etiquetados* (estos son los nodos que se han escaneado durante la iteración o son candidatos actuales para escanear).

En el transcurso de la iteración, continuamos agregando nodos a L y S hasta que se encuentre un camino de aumento o $L = S$, en cuyo caso se demostrará que $\mathbf{d_S}$ es una dirección de ascenso. La iteración

también mantiene una etiqueta para cada nodo $i \in L-I$, que es un arco incidente de i. Las etiquetas son útiles para construir caminos de aumento (lo veremos en el *Paso 3* del algoritmo).

Pasos de la iteración:

Paso 0 (Inicialización):
Seleccionamos un conjunto I de nodos i con $g_i > 0$, tomamos $L := I$ y $S := \emptyset$, y vamos al *Paso 1*.
Si no hay un nodo i con $g_i > 0$, terminamos, ya que:

- Si $g_i = 0 \; \forall i$, entonces $\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = b_i$ se cumple $\Rightarrow$ el par $(\mathbf{x}, \mathbf{w})$ es óptimo.
- Si $g_i \leq 0 \;\; \forall i$ y $\exists\, g_j < 0$, el problema es no factible.

Paso 1 (Eligir un nodo para escanear): Si $S = L$, vamos al *Paso 4*; de lo contrario, seleccionamos un nodo $i \in L - S$, establecemos $S := S \cup \{i\}$ y vamos al *Paso 2*.

Paso 2 (Etiquetar los nodos vecinos de i): Añadimos a L todos los nodos $j \notin L$ tales que (j,i) esté balanceado y $l_{ji} < x_{ji}$ o (i,j) esté balanceado y $x_{ij} < u_{ij}$; también para cada uno de esos j, asignamos a j la etiqueta "$(i^-, x_{ji} - l_{ji})$" si (j,i) está balanceado y $l_{ji} < x_{ji}$, y de lo contrario asignamos a j la etiqueta "$(i^+, u_{ij} - x_{ij})$". Si para todos los nodos j recién agregados a L tenemos $g_j \geq 0$, vamos al *Paso 1*. De lo contrario, seleccionamos uno de estos nodos j con $g_j < 0$ y vamos al *Paso 3*.

Paso 3 (Aumento de flujo): Hemos encontrado un camino de aumento P que comienza en un nodo i perteneciente al conjunto inicial I y termina en el nodo j identificado en el *Paso 2*. La ruta se construye rastreando las etiquetas hacia atrás comenzando desde j, y es tal que tenemos

$$\begin{array}{lll} x_{mn} < u_{mn}, & \forall (m,n) \in P^+, & \forall n \text{ con etiqueta } m^+,\; u_{mn} - x_{mn} \\ x_{mn} > l_{mn}, & \forall (m,n) \in P^-, & \forall m \text{ con etiqueta } n^-,\; u_{mn} - x_{mn} \end{array}$$

donde P^+ y P^- son los conjuntos de arcos hacia delante y hacia atrás de P, respectivamente. Sea

$$\delta = \text{mín}\left\{ g_i, -g_j, \{u_{mn} - x_{mn} \mid (m,n) \in P^+\}, \{x_{mn} - l_{mn} \mid (m,n) \in P^-\} \right\}$$

Aumentamos en δ los flujos de todos los arcos en P^+, disminuimos en δ los flujos de todos los arcos en P^-, restamos δ a g_i y sumamos δ a g_j, y vamos a la siguiente iteración (*Paso 0*).

Paso 4 (Cambio de precio): Sea

$$\begin{aligned} \alpha = \text{mín} \Bigg\{ & \Big\{ c_{ij} - w_i + w_j \mid (i,j) \in A,\; x_{ij} < u_{ij},\; i \in S,\; j \notin S \Big\}, \\ & \Big\{ -(c_{ji} - w_j + w_i) \mid (j,i) \in A,\; l_{ji} < x_{ji},\; i \in S,\; j \notin S \Big\} \Bigg\} \end{aligned} \tag{2.8}$$

Establecemos:

$$w_i = \begin{cases} w_i + \alpha, & \text{si } i \in S \\ w_i, & \text{en otro caso} \end{cases}$$

Añadimos a L todos los nodos j para los cuales el mínimo en (2.8) se obtiene mediante un arco (i,j) o un arco (j,i); también para cada tal j, asignamos a j la etiqueta "$(i^+, u_{ij} - x_{ij})$" si el mínimo en Eq. (2.8) se obtiene mediante un arco (i,j), y en caso contrario damos a j la etiqueta "$(i^-, x_{ji} - l_{ji})$". Si para todos los nodos j recién agregados a L tenemos $g_j \geq 0$, vamos al *Paso 1*. De lo contrario, seleccionamos uno de estos nodos j con $g_j < 0$ y vamos al *Paso 3*. [Nota: si no hay arcos (i,j) con $x_{ij} < u_{ij}$, $i \in S$, y $j \notin S$, o arcos (j,i) con $l_{ji} < x_{ji}$, $i \in S$, y $j \notin S$, el problema es no factible y el algoritmo termina; véase la Prop. 2.3.1 que sigue.]

Tengamos en cuenta lo siguiente con respecto a la iteración típica del algoritmo primal-dual:

(a) Todas las operaciones de la iteración preservan la integralidad del par de vectores flujo-precio.

(b) La iteración mantiene las CHC del par de vectores flujo-precio. Para ver esto, observamos que los arcos con ambos extremos en S, que están balanceados justo antes de un cambio de precio, continúan estando balanceados después de un cambio de precio. Esto significa que un camino de aumento de flujo, incluso si ocurre después de varias ejecuciones del *Paso 4*, cambia solo los flujos de arcos balanceados, por lo que no puede destruir las CHC. Además, un cambio de precio en el *Paso 4* mantiene las CHC porque no se modifica ningún flujo de arco en este paso y el incremento de precio α de la Ecuación (2.8) es tal que ningún arco cambia de estado de activo a inactivo o viceversa.

(c) En todo momento tenemos $S \subset L$. Además, cuando vamos al *Paso 4*, tenemos $S = L$ y L no contiene ningún nodo con exceso negativo. Por lo tanto, según la lógica del *Paso 2*, no existe un arco balanceado (i,j) con $x_{ij} < u_{ij}$, $i \in S$ y $j \notin S$, ni un arco balanceado (j,i) con $l_{ji} < x_{ji}$, $i \in S$, y $j \notin S$. Luego por el razonamiento utilizado para obtener la Ec. (2.7), llegamos a que $\mathbf{d_S}$ es una dirección de ascenso.

(d) Sólo se producen un número finito de cambios de precios en cada iteración, por lo que cada iteración se ejecuta hasta su finalización, ya sea terminando con un aumento de flujo en el *Paso 3* o con una indicación de no factibilidad en el *Paso 4*. Para ver esto, observamos que entre dos cambios de precio, el conjunto L se amplía en al menos un nodo, por lo que no puede haber más de N cambios de precio por iteración.

(e) El algoritmo sólo ejecuta un número finito de pasos de aumento de flujo, ya que cada uno de ellos reduce el exceso absoluto total $\sum_{i \in N} |g_i|$ en una cantidad entera, mientras que los cambios de precio no afectan al exceso absoluto total.

(f) El algoritmo termina. La razón es que cada iteración se ejecutará hasta su finalización (por (d)) e involucrará exactamente un aumento, mientras que solo habrá un número finito de aumentos (por (e)).

La siguiente proposición establece la validez del método.

Proposición 2.3.1. Considerar el problema del flujo de costo mínimo y suponer que c_{ij}, l_{ij}, u_{ij} y b_i son todos números enteros.

(a) Si el problema es factible, entonces el método primal-dual termina con un vector de flujo óptimo entero $\mathbf{x}$ y un vector de precio óptimo entero $\mathbf{w}$.

(b) Si el problema no es factible, entonces el método primal-dual termina porque $g_i \leq 0$ para todo i y $g_i < 0$ para al menos un i o porque no hay ningún arco (i,j) con $x_{ij} < u_{ij}$, $i \in S$, y $j \notin S$, y ningún arco (j,i) con $l_{ji} < x_{ji}$, $i \in S$ y $j \notin S$ en el *Paso 4*.

Demostración. Los items (a) a (f) anteriores garantizan (a) de esta proposición.
Para demostrar (b), nos fijamos que al acabar pueden darse 3 casos:

- En el *Paso 0*, con $g_i = 0 \ \forall i$, llegamos a: solución óptima, función objetivo factible, $\mathbf{w}$ factible y se verifican las CHC (apartado (a)).
- En el *Paso 0*, con $g_i \leq 0 \ \forall i$ y $\exists\, g_j < 0$, el problema es no factible. Suma de ofertas distinta de suma de demandas.
- En el cálculo de α en (2.8), si no encontramos arcos para definir α, llegamos a que el problema es no factible.

Vamos a ver este último caso:

Empezamos la iteración tomando un conjunto I de nodos i con $g_i > 0$. Continuamos con el algoritmo y llegados a un punto tenemos que $S = L$, con lo que iríamos al *Paso 4*. Notar que como $I \subseteq L = S$, entonces $\sum_{i \in S} g_i > 0$ ya que $g_i \geq 0, \ \forall i \in S$ y, en particular, $g_i > 0, \ \forall i \in I \subseteq S$. Aplicando el Lema 2.1.1, tenemos que $\mathbf{d_S} = (d_1, d_2, ..., d_N)$ con $d_i = 1$ si $i \in S$ y $d_i = 0$ si $i \notin S$ es una dirección de ascenso dual.

Ahora, en el *Paso 4*, suponer que no encontramos arcos para definir α; por lo tanto, los arcos (i, j) balanceados con $i \in S$ y $j \notin S$ cumplen que $x_{ij} = u_{ij}$, y los arcos (j, i) balanceados con $i \in S$ y $j \notin S$ cumplen que $x_{ji} = l_{ji}$ (porque si no en el *Paso 4* sí que podría definirse α). Esto implica que $\sum_{i \in S} g_i = q'(\mathbf{w}; \mathbf{d_S})$ (eq. 2.7). Por tanto, al movernos en la dirección $\mathbf{d_S}$, el nuevo $\mathbf{w}_\alpha = \mathbf{w} + \alpha \mathbf{d_S}, \ \alpha > 0$, mantendrá los arcos (i, j) y (j, i), $i \in S, \ j \notin S$, en las mismas condiciones.

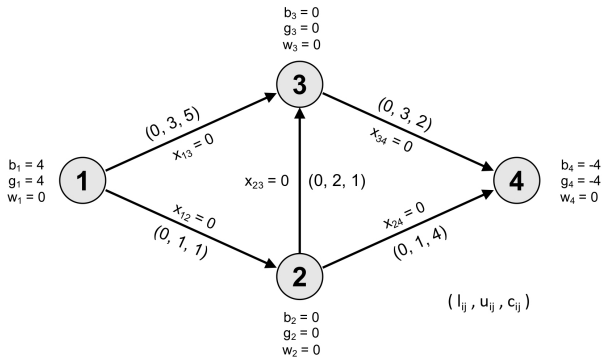
- Si $(i, j) \in A, \ i \in S, \ j \notin S$ y $c_{ij} - w_i + w_j \leq 0$ entonces el nuevo costo marginal es $c_{ij} - w_i + w_j - \alpha \leq 0, \ \forall \alpha > 0$, por lo tanto todos esos arcos siguen estando activos o balanceados.
- Si $(j, i) \in A, \ i \in S, \ j \notin S$ y $c_{ji} - w_j + w_i \geq 0$ entonces el nuevo costo marginal es $c_{ji} - w_j + w_i + \alpha \geq 0, \ \forall \alpha > 0$, por lo tanto todos esos arcos siguen estando inactivos o balanceados.

Como se cumplen estas condiciones, se sigue manteniendo la expresión (2.7), de manera que $\sum_{i \in S} g_i = q'(\mathbf{w}_\alpha; \mathbf{d_S})$, y tenemos que $q'(\mathbf{w}_\alpha; \mathbf{d_S})$ se mantiene constante e igual a $\sum_{i \in S} g_i > 0, \ \forall \alpha > 0$.

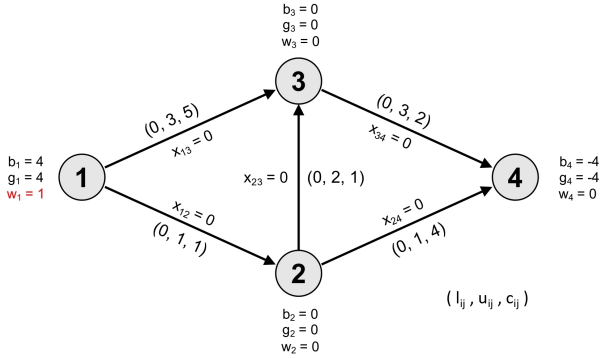
Resumiendo: $\mathbf{d_S}$ es una dirección de ascenso que mantiene constante $q'(\mathbf{w}_\alpha; \mathbf{d_S}), \ \forall \alpha > 0$, por lo que si $\alpha \to +\infty$, el valor de la función objetivo del dual tenderá a $+\infty$ (ya que aumenta en $\alpha \sum_{i \in S} g_i$), con lo cual, el problema dual es no acotado. Aplicando el Teorema fundamental de la dualidad, si el problema dual es no acotado, el primal es no factible y con eso finalizamos la demostración.

□

Con el objetivo de entender el funcionamiento del algoritmo descrito, vamos a proceder a estudiar en detalle el siguiente ejemplo:

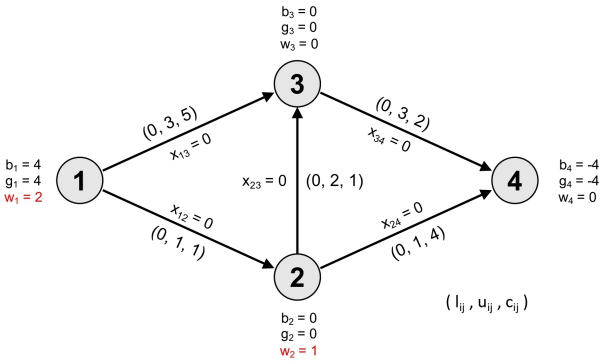


Comenzamos con el *Paso 0*: seleccionamos el nodo $i = \{1\}$ ya que es el único con $g_i > 0$, tomamos $L := \{1\}$ y $S := \emptyset$ y vamos al *Paso 1*. Como $S \neq L$, tomamos un nodo $i \in L - S = \{1\}$, establecemos $S := S \cup \{i\} = \{1\}$ y continuamos con el *Paso 2*. No encontramos ningún arco balanceado, luego no añadimos ningún arco a L y volvemos al *Paso 1*. Ahora sí, tenemos que $S = L = \{1\}$, y continuamos con el *Paso 4*. Calculamos α: $\alpha = \text{mín}\{c_{12} = 1,\ c_{13} = 5\} = 1$, y establecemos $w_i = \begin{cases} w_i + \alpha, & \text{si } i \in S \\ w_i, & \text{en otro caso} \end{cases}$ es decir, $w_1 = 1,\ w_2 = 0,\ w_3 = 0,\ w_4 = 0$. Actualizamos los precios en el grafo:

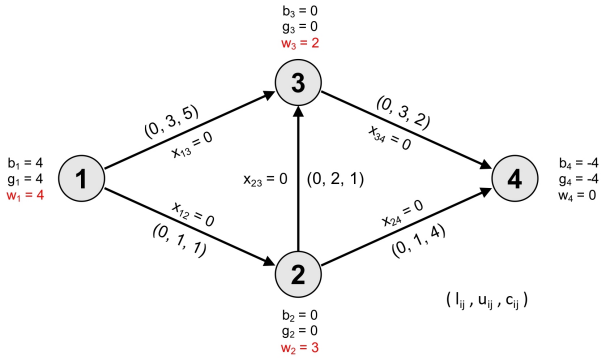


Añadimos a L todos los nodos j para los cuales el mínimo $\alpha = 1$ se obtiene mediante un arco (i, j) o un arco (j, i); en este caso, el nodo 2, y lo etiquetamos: $(2) \longrightarrow (1^+, 1)$. Así, el conjunto pasa a ser $L = \{1, 2\}$. Como $g_2 \geq 0$, vamos al *Paso 1*.

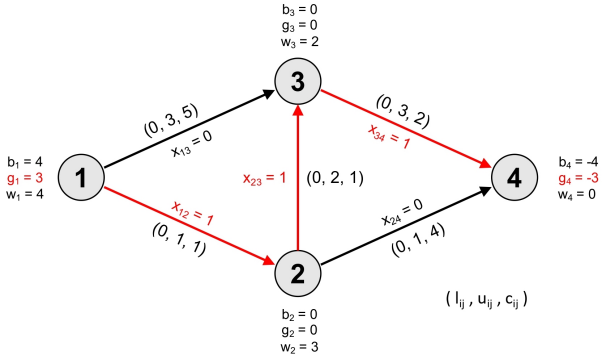
Tenemos que $\{1\} = S \neq L = \{1, 2\}$, luego seleccionamos $i \in L - S = \{2\}$, establecemos $S := S \cup \{i\} = \{1, 2\}$ y vamos al *Paso 2*. No encontramos ningún arco balanceado, luego volvemos al *Paso 1*. Ahora sí, se cumple que $S = L = \{1, 2\}$ y podemos ir al *Paso 4*. Hallamos $\alpha = \text{mín}\{c_{13} - w_1 = 5 - 1,\ c_{23} = 1,\ c_{24} = 4\} = 1$ y actualizamos los precios, de manera que $w_1 = 2,\ w_2 = 1,\ w_3 = 0,\ w_4 = 0$:



Añadimos el nodo 3 a L, $L = \{1, 2, 3\}$, y lo etiquetamos $(3) \longrightarrow (2^+, 2)$. Como $g_3 \geq 0$, vamos al *Paso 1*. Vemos que $S \neq L$, luego seleccionamos $i \in L - S = \{3\}$, definimos $S := S \cup \{i\} = \{1, 2, 3\}$ y vamos al *Paso 2*. De nuevo, no encontramos ningún arco balanceado y pasamos al *Paso 1*. Como $S = L = \{1, 2, 3\}$, podemos ir al *Paso 4*, en el que calculamos $\alpha = \text{mín}\{c_{34} = 2,\ c_{24} - w_2 = 4 - 1\} = 2$ y actualizamos los precios, $w_1 = 4,\ w_2 = 3,\ w_3 = 2,\ w_4 = 0$:

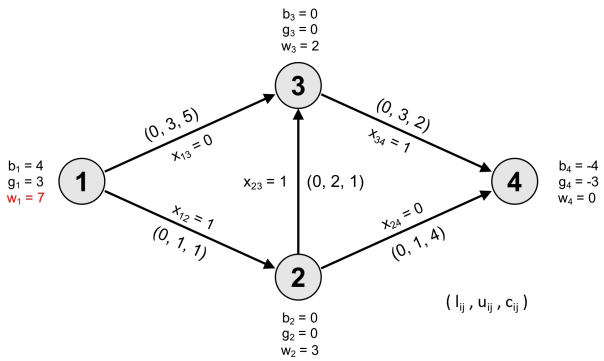


Añadimos a L el nodo 4, $L = \{1,2,3,4\}$, y lo etiquetamos: $(4) \longrightarrow (3^+, 3)$. Como $g_4 < 0$, procedemos a ir al *Paso 3*. Notar que hemos encontrado un camino de aumento P que comienza en (1) y termina en (4), siendo este: $(1,2),\ (2,3),\ (3,4)$.
Hallamos $\delta = \text{mín}\{g_1 = 4, -g_4 = 4, u_{12} - x_{12} = 1, u_{23} - x_{23} = 2, u_{34} - x_{34} = 3\} = 1$.
Aumentamos en $\delta = 1$ los flujos de todos los arcos en P^+, es decir, $x_{12} = 1, x_{23} = 1, x_{34} = 1$. También, restamos δ a g_1: $g_1 := 4 - 1 = 3$; y sumamos δ a g_4: $g_4 := -4 + 1 = -3$:



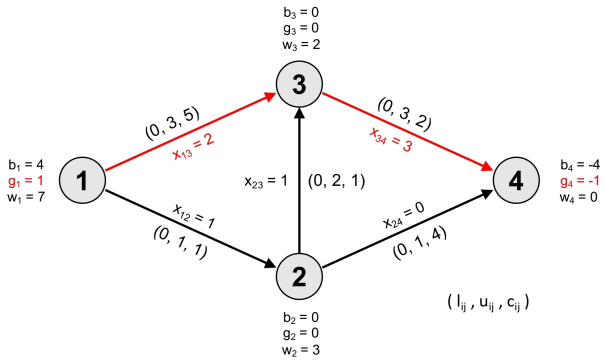
A continuación, pasamos a la siguiente iteración (*Paso 0*).

Iniciamos de nuevo tomando $L = \{1\}$ y $S := \emptyset$, y como $S \neq L$, tomamos $i \in L - S = \{1\}$, definimos $S := S \cup \{i\} = \{1\}$ y vamos al *Paso 2*. Notar que ahora sí hay un arco balanceado ($c_{12} - w_1 + w_2 = 1 - 4 + 3 = 0$); sin embargo, no cumple la cota $x_{ij} < u_{ij}$, luego no añadimos ningún nodo y volvemos al *Paso 1*. Tenemos $S = L = \{1\}$, así que continuamos con el *Paso 4*. Calculamos $\alpha = \text{mín}\{c_{13} - w_1 + w_3 = 3\} = 3$ y actualizamos los precios: $w_1 = 7,\ w_2 = 3,\ w_3 = 2,\ w_4 = 0$.

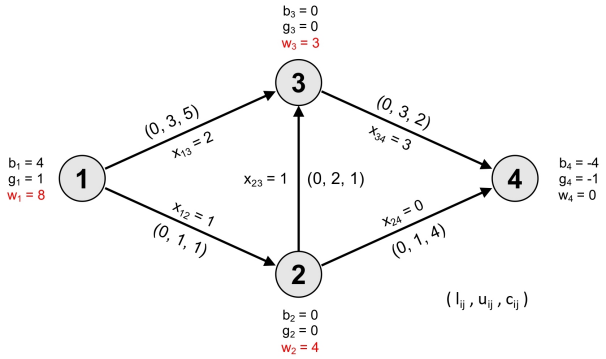


Añadimos el nodo 3 a L, $L = \{1,3\}$, y procedemos a su etiquetación: $(3) \longrightarrow (1^+, 3)$. Como $g_3 \geq 0$, vamos al *Paso 1*. Tenemos $\{1\} = S \neq L = \{1,3\}$, luego modificamos S tal que $S := S \cup \{i\} = \{1,3\}$ y continuamos con el *Paso 2*. Nos encontramos con dos nodos $j \notin L$ tales que son balanceados y cumplen

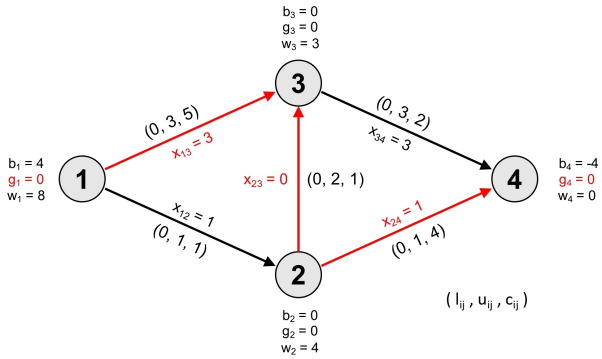
las cotas: el primero, el nodo (4), cumple que $c_{34} - w_3 + w_4 = 2 - 2 + 0 = 0$ y $x_{34} < u_{34}$; y el segundo, el nodo (2), cumple que $c_{23} - w_2 + w_3 = 1 - 3 + 2 = 0$ y $l_{23} < x_{23}$. Llevamos a cabo sus respectivas etiquetaciones: (4) $\longrightarrow (3^+, 1)$, (2) $\longrightarrow (3^-, 1)$. Como $g_4 < 0$, nos dirigimos al *Paso 3*. Hemos encontrado un camino de aumento P que comienza en (1) y termina en (4), cuyos arcos son: $(1,3)$, $(3,4)$. Hallamos $\delta = \text{mín}\{g_1 = 3,\ -g_4 = 3,\ u_{13} - x_{13} = 3,\ u_{34} - x_{34} = 2\} = 2$. Aumentamos en $\delta = 2$ los flujos de todos los arcos en P^+, es decir, $x_{13} = 2, x_{34} = 3$. También, restamos δ a g_1: $g_1 := 3 - 2 = 1$; y sumamos δ a g_4: $g_4 := -3 + 2 = -1$:



A continuación, vamos a llevar a cabo la siguiente iteración, comenzando de nuevo con el *Paso 0*: $L = \{1\}$ y $S := \emptyset$. Como $S \neq L$, añadimos $i = \{1\}$ a S y nos movemos al *Paso 2*. Nos encontramos con que el nodo 3 cumple que $c_{13} - w_1 + w_3 = 5 - 7 + 2 = 0$ y $x_{13} < u_{13}$, luego lo añadimos a L y lo etiquetamos tal que (3) $\longrightarrow (1^+, 1)$. Como $g_3 \geq 0$, vamos al *Paso 1*. Notar que $\{1\} = S \neq L = \{1,3\}$, luego añadimos el nodo 3 a S y seguimos con el *Paso 2*. Estudiamos los nodos $j \notin L$ y vemos que el nodo 2 cumple las condiciones necesarias: $c_{23} - w_2 + w_3 = 1 - 3 + 2 = 0$ y $l_{23} < x_{23}$. Lo añadimos a L, $L = \{1,3,2\}$, y lo etiquetamos (2) $\longrightarrow (3^-, 1)$. (Notar que se trata de un arco hacia atrás.) Como $g_2 \geq 0$, vamos al *Paso 1*. Tenemos $\{1,3\} = S \neq L = \{1,3,2\}$, entonces añadimos $i = \{2\}$ a S y proseguimos con el *Paso 2*. No encontramos ningún arco balanceado así que volvemos al *Paso 1*. Ahora sí que se cumple que $S = L = \{1,3,2\}$, y continuamos con el *Paso 4*. Hallamos $\alpha = \text{mín}\{c_{24} - w_2 + w_4 = 1\} = 1$, y con este actualizamos los precios, de forma que: $w_1 = 8,\ w_2 = 4,\ w_3 = 3,\ w_4 = 0$.



Añadimos a L el nodo 4, $L = \{1,3,2,4\}$, y le asignamos: (4) $\longrightarrow (2^+, 1)$. Vemos que $g_4 < 0$, luego seguimos con el *Paso 3*. Hemos conseguido un camino de aumento P que comienza en (1) y termina en (4). En este caso, el camino es: $(1,3)$, $(3,2)$, $(2,4)$.
Calculamos el valor de $\alpha = \text{mín}\{g_1 = 1,\ -g_4 = 1,\ u_{13} - x_{13} = 1,\ x_{23} - l_{23} = 1,\ u_{24} - x_{24} = 1\} = 1$.
Procedemos a aumentar en $\delta = 1$ los flujos de todos los arcos en P^+ y disminuir en $\delta = 1$ los flujos de todos los arcos en P^-, es decir, $x_{13} = 3$, $x_{23} = 0$, $x_{24} = 1$. Por último, restamos $\delta = 1$ a g_1: $g_1 := 1 - 1 = 0$, y sumamos $\delta = 1$ a g_4: $g_4 := -1 + 1 = 0$, y vamos al *Paso 0*:



Llegados a este punto, vemos que $g_i = 0$, $\forall i$, lo que significa que el par $(\mathbf{x}, \mathbf{w})$ es óptimo y finalizamos las iteraciones del algoritmo. Por tanto, como podemos ver en el último grafo, la solución óptima de nuestro ejemplo es:

$$\mathbf{x} = (x_{12} = 1,\ x_{13} = 3,\ x_{23} = 0,\ x_{24} = 1,\ x_{34} = 3), \qquad \mathbf{w} = (w_1 = 8,\ w_2 = 4,\ w_3 = 3,\ w_4 = 0)$$

Capítulo 3

Estudio computacional del algoritmo

En este capítulo vamos a realizar un estudio computacional del algoritmo estudiado. El algoritmo primal-dual presentado en el *Capítulo 2* ha sido implementado en Python (su código se proporciona en el *Anexo B*) y vamos a aplicarlo a una colección de 450 problemas[1] creados con una versión simplificada del generador de redes PGRIDGEN.

Estos 450 problemas están agrupados en 3 configuraciones de redes distintas. En cada una de ellas, fijaremos los valores de todas las variables de interés excepto de una, la cual variaremos con el objetivo de estudiar el comportamiento del tiempo de resolución del algoritmo. Para la Configuración 1, incrementaremos el número de arcos de 100 a 500; para la Configuración 2, el número de nodos con oferta y demanda de 10 a 50; y para la Configuración 3, la capacidad de los arcos [2] desde (100-200) a (800-1000). Los parámetros considerados en detalle se muestran en la Tabla 3.1.

	nodos	arcos	of/dem	costo	capacidad	of total
Conf 1	200	*1000*	20	(100-500)	(500-1000)	20000
	200	*2000*	20	(100-500)	(500-1000)	20000
	200	*3000*	20	(100-500)	(500-1000)	20000
	200	*4000*	20	(100-500)	(500-1000)	20000
	200	*5000*	20	(100-500)	(500-1000)	20000
Conf 2	400	2500	*10*	(100-500)	(500-1000)	20000
	400	2500	*20*	(100-500)	(500-1000)	20000
	400	2500	*30*	(100-500)	(500-1000)	20000
	400	2500	*40*	(100-500)	(500-1000)	20000
	400	2500	*50*	(100-500)	(500-1000)	20000
Conf 3	200	2500	20	(100-500)	$(100-200)$	20000
	200	2500	20	(100-500)	$(200-400)$	20000
	200	2500	20	(100-500)	$(400-600)$	20000
	200	2500	20	(100-500)	$(600-800)$	20000
	200	2500	20	(100-500)	$(800-1000)$	20000

Tabla 3.1: Características de las configuraciones de los problemas test.

Para cada caso de cada configuración se han generado 30 problemas aleatorios y se han resuelto almacenando su tiempo de ejecución. Posteriormente, he desarrollado unos pequeños scripts en R (código proporcionado en el *Anexo C*) para realizar distintos cálculos que utilizaré en el estudio.

[1] Los problemas los proporciona el director del TFG para hacer el estudio.

[2] En este estudio computational hemos fijado el valor de la cota inferior de los arcos de manera que $l_{ij} = 0$ para todos los problemas. Además, cuando hablemos de `cap_min` y `cap_max`, nos referiremos al rango de valores que puede tomar u_{ij}, la cota superior, siendo esta el factor de estudio en la Configuración 3, al que llamaremos `capacidad`.

En la Tabla 3.2 presentamos las medias y desviaciones típicas del tiempo de ejecución obtenidas para cada configuración (Anexo C.1). Para determinar si existen diferencias significativas en los tiempos para cada configuración realizaré un test ANOVA de un factor en cada configuración (Anexo C.4) y posteriormente una comparación múltiple de las medias de los tiempos en cada valor del factor de interés mediante el test de Tukey.

Configuración 1

Número de arcos	Media	Desv. típica
1000	5.6118211	0.5414283
2000	5.0743180	0.4201189
3000	4.9174770	0.4554643
4000	4.7513687	0.2361825
5000	4.8604421	0.3902182

Configuración 2

Número de ofertas	Media	Desv. típica
10	19.029985	1.879225
20	19.971793	1.733904
30	22.157543	1.605307
40	22.263399	1.892302
50	24.982612	1.663857

Configuración 3

Capacidad arcos	Media	Desv. típica
100 - 200	9.6475211	0.6203105
200 - 400	8.1587474	0.5838491
400 - 600	6.9060912	0.4068665
600 - 800	6.8726857	0.4366017
800 - 1000	6.8750893	0.4283918

Tabla 3.2: Medias y desviaciones típicas de los tiempos de ejecución de cada una de las distintas configuraciones.

Para entender y analizar más fácilmente estos resultados, podemos representarlos gráficamente. En las figuras 3.1 a 3.3 podemos observar los tiempos medios de ejecución.

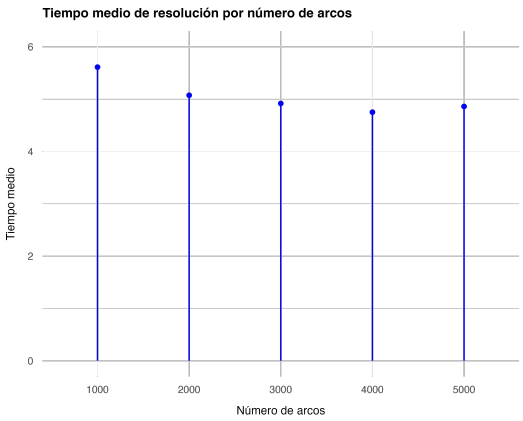


Figura 3.1: Tiempo medio de resolución por número de arcos para la Configuración 1.

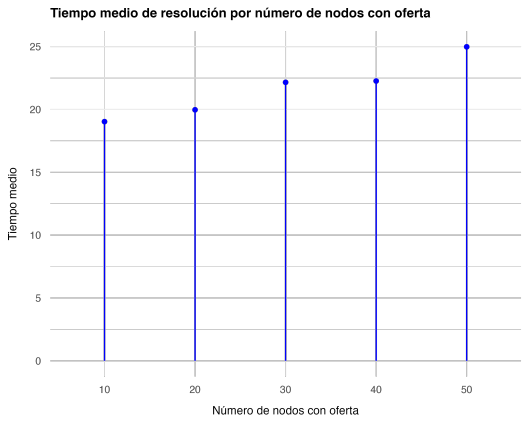


Figura 3.2: Tiempo medio de resolución por número de ofertas para la Configuración 2.

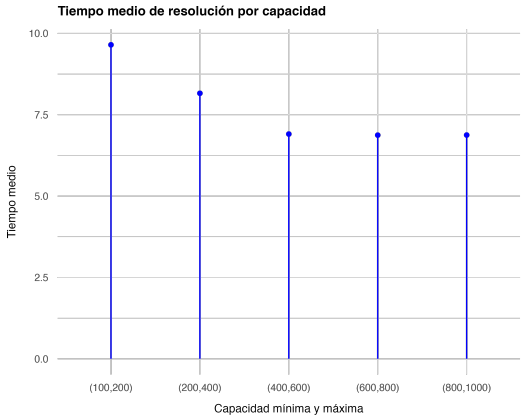


Figura 3.3: Tiempo medio de resolución por capacidad de los arcos para la Configuración 3.

Para realizar el ANOVA se ha realizado previamente un test de normalidad de Shapiro-Wilk (*Anexo C.2*) de la variable `tiempoEjecucion` con las variables que van cambiando, es decir, `numero_arcos` para la Configuración 1; `num_ofertas` para la Conf. 2; y `cap_min` y `cap_max` para la Conf. 3. Examinando los p-valores obtenidos (ver en la Figura 3.4), no se rechaza la normalidad de los datos ($\alpha = 0{,}05$) para la mayoría de los conjuntos (se rechaza únicamente con Conf. 2, cuando of $= 20$, que da un p-valor $= 0{,}01$ debido a un único dato un poco elevado. Se eliminó dicho dato pero los resultados del ANOVA eran iguales por lo que se mantiene el resto con los datos originales).

Configuración 1

```
 p-values adjusted by the Holm method:
     unadjusted adjusted
1000 0.079936   0.31974
2000 0.869184   0.86918
3000 0.300212   0.74534
4000 0.248445   0.74534
5000 0.062910   0.31455
```

Configuración 2

```
 p-values adjusted by the Holm method:
   unadjusted adjusted
10 0.175233   0.700931
20 0.013828   0.069139
30 0.217773   0.700931
40 0.811329   0.811329
50 0.176101   0.700931
```

Configuración 3

```
 p-values adjusted by the Holm method:
           unadjusted adjusted
(100,200)  0.692964   1.00000
(200,400)  0.071743   0.35871
(400,600)  0.338557   1.00000
(600,800)  0.428021   1.00000
(800,1000) 0.352664   1.00000
```

Figura 3.4: Test de normalidad de Shapiro-Wilk para cada una de las Configuraciones.

Seguidamente, procedemos a realizar un test de igualdad de varianzas: test de Levene (*Anexo C.3*), con el factor correspondiente en cada caso y la variable `tiempoEjecucion`, rechazándose la hipótesis de igualdad de varianzas en las configuraciones 1 y 3 ($\alpha = 0{,}05$).

Configuración 1

```
Levene's Test for Homogeneity of Variance (center = "median")
      Df F value    Pr(>F)
group   4  4.8987 0.0009833 ***
      145
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Configuración 2

```
Levene's Test for Homogeneity of Variance (center = "median")
      Df F value Pr(>F)
group   4  0.3494 0.8441
      145
```

Configuración 3

```
Levene's Test for Homogeneity of Variance (center = "median")
      Df F value Pr(>F)
group   4  2.6457 0.0359 *
      145
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figura 3.5: Test de Levene para cada una de las Configuraciones.

Por tanto, aplicamos el test de ANOVA de Welch (*Anexo C.4*), ya que esta nos permite evitar asumir la igualdad de varianzas y obtenemos en los 3 casos (ver en la Figura 3.6) un p-valor prácticamente nulo, por lo que asumimos que los tiempos de ejecución medios no son iguales para los distintos valores del factor de cada configuración.

Configuración 1

```
> summary(AnovaModel.1)
             Df Sum Sq Mean Sq F value   Pr(>F)
numero_arcos   4  13.76   3.440   19.43 8.04e-13 ***
Residuals    145  25.67   0.177
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> print(oneway.test(tiempoEjecucion ~ numero_arcos, data = datos_conf1))

        One-way analysis of means (not assuming equal variances)

data:  tiempoEjecucion and numero_arcos
F = 16.772, num df = 4.000, denom df = 70.458, p-value = 0.000000001059
```

Configuración 2

```
> summary(AnovaModel.2)
             Df Sum Sq Mean Sq F value Pr(>F)
num_ofertas   4  642.5  160.62   51.93 <2e-16 ***
Residuals   145  448.5    3.09
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> print(oneway.test(tiempoEjecucion ~ num_ofertas, data = datos_conf2))

        One-way analysis of means (not assuming equal variances)

data:  tiempoEjecucion and num_ofertas
F = 51.472, num df = 4.000, denom df = 72.424, p-value < 2.2e-16
```

Configuración 3

```
> summary(AnovaModel.3)
             Df Sum Sq Mean Sq F value Pr(>F)
capacidad     4  179.9   44.99   177.8 <2e-16 ***
Residuals   145   36.7    0.25
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> print(oneway.test(tiempoEjecucion ~ capacidad, data = datos_conf3))

        One-way analysis of means (not assuming equal variances)

data:  tiempoEjecucion and capacidad
F = 140.38, num df = 4.000, denom df = 72.051, p-value < 2.2e-16
```

Figura 3.6: Test de ANOVA de Welch para cada una de las Configuraciones.

Finalmente, realizamos el test de Tukey (*Anexo C.5*), y con él obtenemos los siguientes resultados mostrados en la Figura 3.7.

Configuración 1

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = tiempoEjecucion ~ numero_arcos, data = datos_conf1)

$numero_arcos
               diff        lwr         upr     p adj
2000-1000 -0.53750308 -0.8376017 -0.23740447 0.0000201
3000-1000 -0.69434412 -0.9944427 -0.39424551 0.0000000
4000-1000 -0.86045236 -1.1605510 -0.56035375 0.0000000
5000-1000 -0.75137904 -1.0514776 -0.45128043 0.0000000
3000-2000 -0.15684104 -0.4569396  0.14325757 0.6005473
4000-2000 -0.32294927 -0.6230479 -0.02285067 0.0280980
5000-2000 -0.21387595 -0.5139746  0.08622266 0.2865610
4000-3000 -0.16610823 -0.4662068  0.13399037 0.5453997
5000-3000 -0.05703491 -0.3571335  0.24306370 0.9846877
5000-4000  0.10907332 -0.1910253  0.40917193 0.8531681
```

Configuración 2

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = tiempoEjecucion ~ num_ofertas, data = datos_conf2)

$num_ofertas
              diff       lwr      upr     p adj
20-10  0.9418082 -0.312545 2.196161 0.2369746
30-10  3.1275584  1.873205 4.381912 0.0000000
40-10  3.2334140  1.979061 4.487767 0.0000000
50-10  5.9526269  4.698274 7.206980 0.0000000
30-20  2.1857502  0.931397 3.440103 0.0000360
40-20  2.2916058  1.037253 3.545959 0.0000130
50-20  5.0108187  3.756465 6.265172 0.0000000
40-30  0.1058556 -1.148498 1.360209 0.9993391
50-30  2.8250685  1.570715 4.079422 0.0000000
50-40  2.7192129  1.464860 3.973566 0.0000002
```

Configuración 3

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = tiempoEjecucion ~ capacidad, data = datos_conf3)

$capacidad
                        diff       lwr       upr     p adj
(200,400)-(100,200)  -1.488773704 -1.8475812 -1.1299662 0.0000000
(400,600)-(100,200)  -2.741429949 -3.1002374 -2.3826225 0.0000000
(600,800)-(100,200)  -2.774835428 -3.1336429 -2.4160280 0.0000000
(800,1000)-(100,200) -2.772431874 -3.1312393 -2.4136244 0.0000000
(400,600)-(200,400)  -1.252656245 -1.6114637 -0.8938488 0.0000000
(600,800)-(200,400)  -1.286061724 -1.6448692 -0.9272543 0.0000000
(800,1000)-(200,400) -1.283658171 -1.6424656 -0.9248507 0.0000000
(600,800)-(400,600)  -0.033405479 -0.3922130  0.3254020 0.9990264
(800,1000)-(400,600) -0.031001925 -0.3898094  0.3278055 0.9992746
(800,1000)-(600,800)  0.002403553 -0.3564039  0.3612110 1.0000000
```

Figura 3.7: Test de Tukey para las distintas configuraciones.

En la Configuración 1 se obtiene lo siguiente: el tiempo de ejecución de las redes con 1000 arcos es el mayor y significativamente diferente del resto; 2000, 3000 y 5000 arcos son equivalentes en tiempo y menores significativamente que 1000; y el tiempo de ejecución con 4000 arcos es menor que con 2000 y equivalente a 3000 y 5000 arcos.

Por tanto, se observa que al ir aumentando el número de arcos el tiempo disminuye hasta llegar a 4000, momento en el que comienza a aumentar otra vez.

Por la estructura de la red, podemos interpretar este resultado pensando en que al tener más arcos es más fácil que existan trayectorias cortas que si tenemos menos arcos. Si hay menos arcos pueden existir trayectorias cortas pero tendremos que redistribuir y aprovechar más los arcos porque la oferta es constante, es decir, hay fijadas 20000 unidades de flujo para repartir de los 20 nodos iniciales a los 20 finales, luego al tener la red con pocos arcos hay que saturarlos y utilizarlos mucho, lo cual implica que le cuesta más tiempo que si tiene muchísimos arcos ya que tendrá más para elegir y será capaz de encontrar trayectorias de menor costo computacional (de menos arcos).

Esto sucede hasta que llega un momento en el que el hecho de tener más arcos ya no da beneficios, sino que hace que el tiempo de ejecución aumente debido a que hay demasiados arcos para etiquetar y mirar, pero que no llegan a aportar más caminos nuevos cortos.

Por otro lado, en el test de Tukey de la Configuración 2, podemos ver que 10 y 20 nodos con oferta se muestran equivalentes, y también 30 y 40. En el resto de comparaciones, podemos ver que hay diferencias significativas en la media de los tiempos de ejecución entre las siguientes parejas de números de nodos con oferta/demanda: 10 - 30; 10 - 40; 10 - 50; 20 - 30; 20 - 40; 20 - 50; 30 - 50; 40 - 50. Ahora, en la columna `diff` podemos ver que los valores son positivos, lo cual indica que el tiempo medio de ejecución va aumentando conforme aumenta el número de nodos con oferta/demanda (también podemos verlo en la Figura 3.2. Si, por ejemplo, empezamos con 10 nodos con oferta y 10 con demanda, entonces tenemos que manejar menos que si tuviésemos 20, porque tan solo al inicio el conjunto I ya tendría más nodos. Luego conforme aumentamos el número de nodos con oferta, la oferta total está más repartida y por tanto tendrá que hacer muchas más búsquedas para ir redistribuyendo el exceso de los nodos de I.

Por último, en la Configuración 3 podemos observar que el p-valor es menor que 0.05 en todas las parejas excepto en las 3 últimas, es decir, $(100,200)$ presenta el mayor tiempo de resolución; y $(200,400)$ mayor que $(400,600)$, $(600,800)$ y $(800,1000)$, los cuales son equivalentes entre sí. Valorando el test de Tukey y la Figura 3.3, llegamos a la conclusión de que si las capacidades son muy pequeñas, digamos entre 100-200, el algoritmo tiene que buscar muchos caminos y hacer muchas iteraciones para transportar el flujo; pero conforme aumentamos la capacidad, al emplear un arco puedo utilizarlo para enviar mucho flujo, luego tendrá que realizar menos iteraciones.

Llega un punto en el que se estabiliza, porque los arcos tienen tal capacidad que aunque fuese au-

mentada ya no se podría enviar más porque no dispondríamos de más flujo.

Como conclusión de este estudio estadístico, podemos decir que en la Configuración 1 el tiempo de ejecución disminuye conforme aumenta el número de arcos, hasta tener un determinado número de arcos en el que volvería a crecer el tiempo medio, es decir, el incremento del número de arcos no afecta en general negativamente como sería de esperar. En la Configuración 2, cuantos más nodos con oferta haya, mayor será su tiempo medio de ejecución, lo cual concuerda con lo esperado. Finalmente, en la Configuración 3, el algoritmo es más rápido cuanto mayor sea la capacidad de los arcos, hasta que llegados a un punto se estabiliza y el tiempo medio es el mismo aunque se incremente la capacidad, lo cual nuevamente es lo esperado ya que si la capacidad de los arcos es pequeña hay muchas saturaciones de arcos y esto obliga a tener que hacer muchos envíos de flujos y las correspondientes actualizaciones de w's que nos permitan generar nuevos arcos balanceados para continuar el etiquetado.

Bibliografía

[1] M. S. BAZARAA, J. J. JARVIS Y H. D. SHERALI, *Linear Programming and Network Flows*, New Jersey, 2010.

[2] D. P. BERTSEKAS, *Linear Network Optimization: Algorithms and Codes*, Massachusetts, 1991.

[3] *Material de estudio de la asignatura Investigación Operativa del grado en Matemáticas de la Universidad de Zaragoza*, 2020.

[4] *Python*, `https://docs.python.org/3/tutorial/index.html`.

[5] *Introducción a R*, `https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf`.

[6] AURORA GONZÁLEZ VIDAL, *FEIR 30: Comparación paramétrica de medias*, `https://gauss.inf.um.es/feir/30/`, Enero 2024.

Anexo A

Algoritmo 'primal-dual'

Vamos a presentar el código de Python que corresponde con el algoritmo primal-dual visto en el *Capítulo 2*.

```
import sys
import numpy as np
import time
import os

ruta_principal = "./Datos/"
fichero_principal = ruta_principal + "listado.txt"
CONF_1_ficheros = []
CONF_2_ficheros = []
CONF_3_ficheros = []

def leer_ficheros(fichero_principal):
    with open(fichero_principal,"r") as f:
        ficheros = f.readlines()

    for fichero in ficheros:
        if fichero.strip().startswith('CONF_1_'):
            nombre_fichero = fichero.strip().strip("\n")
            CONF_1_ficheros.append(nombre_fichero)
            nombre_fichero_CONF_1 = 'CONF_1_listado.txt'
            ruta_completa = os.path.join(ruta_principal, nombre_fichero_CONF_1)
            with open(ruta_completa, 'a') as resultado:
                resultado.write(f"{nombre_fichero}\n")

        elif fichero.strip().startswith('CONF_2_'):
            nombre_fichero = fichero.strip().strip("\n")
            CONF_2_ficheros.append(nombre_fichero)
            nombre_fichero_CONF_2 = 'CONF_2_listado.txt'
            ruta_completa = os.path.join(ruta_principal, nombre_fichero_CONF_2)
            with open(ruta_completa, 'a') as resultado:
                resultado.write(f"{nombre_fichero}\n")

        elif fichero.strip().startswith('CONF_3_'):
            nombre_fichero = fichero.strip().strip("\n")
            CONF_3_ficheros.append(nombre_fichero)
            nombre_fichero_CONF_3 = 'CONF_3_listado.txt'
            ruta_completa = os.path.join(ruta_principal, nombre_fichero_CONF_3)
```

```
            with open(ruta_completa, 'a') as resultado:
                resultado.write(f"{nombre_fichero}\n")

leer_ficheros(fichero_principal)

def leeFichero(nombre_fichero):
    with open(nombre_fichero, "r") as f:
        lineas = f.readlines()

    # Definimos las variables que usaremos
    numeroNodos = 0
    numeroArcos = 0
    num_ofertas = 0
    nodo = 0
    Of = 0
    nodoInicial = 0
    nodoFinal = 0
    minflow = 0
    maxflow = 0
    cost = 0
    cost_min = 0
    cost_max = 0
    cap_min = 0
    cap_max = 0
    seed = 0
    arcs_count = 0

    # Variables para almacenar los datos como vectores
    l = []
    u = []
    c = []
    x = []
    w = []

    # Leemos el fichero linea por linea
    for linea in lineas:
        # Si encontramos una linea que empieza con 'c', leemos el num_ofertas, cost_min,
        # cost_max, cap_min, cap_max, seed
        if linea.strip().startswith('c'):
            trozos = linea.strip().strip("\n").split()
            if trozos[1] == 'no._supplies/demands':
                num_ofertas = int(trozos[2])
            elif trozos[1] == 'cost_min':
                cost_min = int(trozos[2])
            elif trozos[1] == 'cost_max':
                cost_max = int(trozos[2])
            elif trozos[1] == 'cap_min_arcs':
                cap_min = int(trozos[2])
            elif trozos[1] == 'cap_max_arcs':
                cap_max = int(trozos[2])
            elif trozos[1] == 'seed':
                seed = int(trozos[2])
```

```
        # Si encontramos una linea que empieza con 'p', definimos las matrices
        elif linea.strip().startswith('p'):
            trozos = linea.strip().strip("\n").split()
            numeroNodos = int(trozos[2])
            numeroArcos = int(trozos[3])
            A = np.full((numeroNodos, numeroNodos), -1, dtype=int)
            l = np.zeros(numeroArcos, dtype=int)
            u = np.zeros(numeroArcos, dtype=int)
            c = np.zeros(numeroArcos, dtype=int)
            x = np.zeros(numeroArcos, dtype=int)
            w = np.zeros(numeroNodos, dtype=int)
            Of = np.zeros(shape=numeroNodos, dtype=int)

        # Si encontramos una linea que empieza con 'n', leemos el nodo y su oferta/demanda
        elif linea.strip().startswith('n'):
            trozos = linea.strip().strip("\n").split()
            nodo = int(trozos[1]) - 1
            Of[nodo] = int(trozos[2])

        # Si encontramos una linea que empieza con 'a', leemos los nodos final e inicial,
        # la cota inferior y superior, y el costo
        elif linea.strip().startswith('a'):
            trozos = linea.strip().strip("\n").split()
            nodoInicial = int(trozos[1]) - 1
            nodoFinal = int(trozos[2]) - 1
            minflow = int(trozos[3])
            maxflow = int(trozos[4])
            cost = int(trozos[5])

            # Asignamos un numero de arco a la posicion de la matriz A
            A[nodoInicial][nodoFinal] = arcs_count

            # Guardamos las cotas y costos en los vectores correspondientes
            l[arcs_count] = minflow
            u[arcs_count] = maxflow
            c[arcs_count] = cost
            arcs_count += 1

            continue

    return (A, l, u, c, Of, numeroNodos, numeroArcos, x, w, arcs_count,
            num_ofertas, cost_min, cost_max, cap_min, cap_max, seed)

def calcular_exceso(i, A, x, Of, numeroNodos):
    # Calculamos el flujo de entrada a i (sum_ji) y el flujo de salida desde i (sum_ij)
    sum_ij = []
    sum_ji = []

    for j in range(numeroNodos):
        if A[i][j] != -1:
            a = A[i][j]
            x_ij = x[a]
```

```
            sum_ij.append(x_ij)

        if A[j][i] != -1:
            a = A[j][i]
            x_ji = x[a]
            sum_ji.append(x_ji)

    total_sum_ij = sum(sum_ij)
    total_sum_ji = sum(sum_ji)
    exceso = total_sum_ji - total_sum_ij + Of[i]

    return exceso

def paso1(L, S):
    for i in L:
        if i not in S:
            S.append(i)
            break # Detenemos el bucle al encontrar el primer nodo en L-S

    return i

def paso2(L, S, i, c, w, arcs_count, l, u, x, Of, numeroNodos, A,excesos,etiquetas_nodos):
    # Comprobamos todos los nodos vecinos de i que cumplan las condiciones:
    for j in range(numeroNodos):
        if A[i][j] != -1 and j not in L:
            a = A[i][j]  # Indice del arco (i,j)
            if c[a] - w[i] + w[j] == 0 and x[a] < u[a]:
                L.append(j)
                # Etiqueta para arcos (i,j)
                etiqueta = (i, '+', min(u[a] - x[a],etiquetas_nodos[i][2]),a)
                etiquetas_nodos[j] = etiqueta

        if A[j][i] != -1 and j not in L:
            a = A[j][i]  # Indice del arco (j,i)
            if c[a] - w[j] + w[i] == 0 and l[a] < x[a]:
                L.append(j)
                # Etiqueta para arcos (j, i)
                etiqueta = (i, '-', min(x[a] - l[a],etiquetas_nodos[i][2]),a)
                etiquetas_nodos[j] = etiqueta

    if all(excesos[j]>=0 for j in L):
        return 0

    else:
        # Seleccionamos uno de esos nodos j con exceso < 0 y vamos al paso3
        # Usamos un bucle sobre los nodos en L para encontrar el primer nodo con exceso < 0
        for j in L:
            if excesos[j] < 0:
                return j
                break  # Detenemos el bucle al encontrar el primer nodo con exceso < 0

def paso3(nodo_j, etiquetas_nodos, L, A, x, u, l, Of, numeroNodos,excesos):
```

```
    delta = etiquetas_nodos[nodo_j][2]

    excesos[nodo_j] += delta

    while etiquetas_nodos[nodo_j][0] != -1 :
        if etiquetas_nodos[nodo_j][1] == '+':
            x[etiquetas_nodos[nodo_j][3]] += delta
        if etiquetas_nodos[nodo_j][1] == '-':
            x[etiquetas_nodos[nodo_j][3]] -= delta
        nodo_j = etiquetas_nodos[nodo_j][0]
    excesos[nodo_j] -= delta


def calcular_alpha(S, numeroNodos, x, w, A, u, l, c):
    alphas_forward = []  # Almacenara los valores minimos de arcos (i, j)
    alphas_backward = []  # Almacenara los valores minimos de arcos (j, i)

    for i in S:
        for j in range(numeroNodos):
            if j not in S and A[i][j] != -1 and x[A[i][j]] < u[A[i][j]]:
                alphas_forward.append(c[A[i][j]] - w[i] + w[j])

            if j not in S and A[j][i] != -1 and x[A[j][i]] > l[A[j][i]]:
                alphas_backward.append(-(c[A[j][i]] - w[j] + w[i]))

    # Verificamos si las listas estan vacias antes de encontrar el minimo
    alpha_forward = min(alphas_forward) if alphas_forward else float('inf')
    alpha_backward = min(alphas_backward) if alphas_backward else float('inf')

    alpha = min(alpha_forward, alpha_backward)

    return alpha

def paso4(S, L, numeroNodos, x, w, A, u, l, c, Of,excesos,etiquetas_nodos):
    # Calculamos alpha usando la función 'calcular_alpha'
    alpha = calcular_alpha(S, numeroNodos, x, w, A, u, l, c)

    # Verificamos si alpha es infinito (lo que significa que no hay arcos que cumplan las
    # condiciones para calcular 'alpha'), lo que indica un problema no factible
    if alpha == float('inf'):
        print("El problema es no factible.")
        sys.exit()

    else:
        # Añadimos a L todos los nodos j para los cuales el minimo se obtiene
        # mediante un arco (i,j) o un arco (j,i)
        for i in S:
            for j in range(numeroNodos):
                if j not in S and j not in L:
                    if (A[i][j] != -1 and x[A[i][j]] < u[A[i][j]]
                        and c[A[i][j]] - w[i] + w[j] == alpha):
                        a = A[i][j]  # Indice del arco (i,j)
```

```
                        L.append(j)
                        # Etiqueta para arcos (i,j)
                        etiqueta = (i, '+', min(u[a] - x[a],etiquetas_nodos[i][2]),a)
                        etiquetas_nodos[j] = etiqueta

                    if (A[j][i] != -1 and x[A[j][i]] > l[A[j][i]]
                        and -(c[A[j][i]] - w[j] + w[i]) == alpha):
                        a = A[j][i]  # Indice del arco (j,i)
                        L.append(j)
                        # Etiqueta para arcos (j, i)
                        etiqueta = (i, '-', min(x[a] - l[a],etiquetas_nodos[i][2]),a)
                        etiquetas_nodos[j] = etiqueta

        # Actualizamos precios:
        for i in range(numeroNodos):
            if i in S:
                w[i] = w[i] + alpha

        if all(excesos[j]>=0 for j in L):
            return 0

        else:
            # Seleccionamos uno de esos nodos j con exceso < 0 y vamos al paso3
            # Usamos un bucle sobre los nodos en L para encontrar el primer nodo
            # con exceso < 0
            for j in L:
                if excesos[j] < 0:
                    return j
                    break  # Detenemos el bucle al encontrar el primer nodo con exceso < 0

def programa_principal(nombre_fichero):
    (A, l, u, c, Of, numeroNodos, numeroArcos, x, w, arcs_count, num_ofertas, cost_min,
     cost_max, cap_min, cap_max, seed) = leeFichero(nombre_fichero)

    etiquetas_nodos = []*numeroNodos
    L = []
    S = []
    x = [0]*numeroArcos
    w = [0]*numeroNodos
    excesos= [0] * numeroNodos

    etiquetas_nodos=[[]]*numeroNodos
    L.clear()
    S.clear()

    for i in range(numeroNodos):
        excesos[i] = Of[i]
        if excesos[i] > 0:
            L.append(i)
            etiquetas_nodos[i] = (-1, '*', excesos[i]);

    it=0
```

```
    while(L!=0):
        it += 1
        if S == L:
            j = paso4(S, L, numeroNodos, x, w, A, u, l, c, Of,excesos,etiquetas_nodos)
            if j == 0:
                continue
            else:
                paso3(j, etiquetas_nodos, L, A, x, u, l, Of, numeroNodos,excesos)
                etiquetas_nodos = [[]]*numeroNodos

                S.clear()
                L.clear()

                for i in range(numeroNodos):
                    if excesos[i] > 0:
                        L.append(i)
                        etiquetas_nodos[i] = (-1, '*', excesos[i]);

        else:
            i = paso1(L, S)
            j = paso2(L, S, i, c, w, arcs_count, l, u, x, Of, numeroNodos, A,excesos,
                      etiquetas_nodos)

            if j == 0:
                continue
            else:
                paso3(j, etiquetas_nodos, L, A, x, u, l, Of, numeroNodos,excesos)

                etiquetas_nodos = [[]]*numeroNodos
                L.clear()
                S.clear()

                for i in range(numeroNodos):
                    if excesos[i] > 0:
                        L.append(i)
                        etiquetas_nodos[i] = (-1, '*', excesos[i]);
        if len(L) == 0:
            if all(exceso_i == 0 for exceso_i in excesos):
                tiempo_fin = time.time()
                tiempo_ejecucion = tiempo_fin - tiempo_inicio

                if fichero in CONF_1_ficheros:
                    nombre_fichero_resultados = 'CONF_1_resultados.txt'
                    ruta_completa = os.path.join(ruta_principal, nombre_fichero_resultados)
                    with open(ruta_completa, 'a') as resultado:
                        resultado.write(f"{seed} {numeroNodos} {numeroArcos} {num_ofertas} "
                                        f"{cost_min} {cost_max} {cap_min} {cap_max} "
                                        f"{tiempo_ejecucion}\n")
                        break

                if fichero in CONF_2_ficheros:
                    nombre_fichero_resultados = 'CONF_2_resultados.txt'
```

```
                    ruta_completa = os.path.join(ruta_principal, nombre_fichero_resultados)
                    with open(ruta_completa, 'a') as resultado:
                        resultado.write(f"{seed} {numeroNodos} {numeroArcos} {num_ofertas} "
                                        f"{cost_min} {cost_max} {cap_min} {cap_max} "
                                        f"{tiempo_ejecucion}\n")
                        break

                if fichero in CONF_3_ficheros:
                    nombre_fichero_resultados = 'CONF_3_resultados.txt'
                    ruta_completa = os.path.join(ruta_principal, nombre_fichero_resultados)
                    with open(ruta_completa, 'a') as resultado:
                        resultado.write(f"{seed} {numeroNodos} {numeroArcos} {num_ofertas} "
                                        f"{cost_min} {cost_max} {cap_min} {cap_max} "
                                        f"{tiempo_ejecucion}\n")
                        break

            else:
                print('El problema es no factible')
                sys.exit()

for fichero in CONF_1_ficheros:
    nombre_fichero = ruta_principal + fichero
    tiempo_inicio = time.time()
    programa_principal(nombre_fichero)

for fichero in CONF_2_ficheros:
    nombre_fichero = ruta_principal + fichero
    tiempo_inicio = time.time()
    programa_principal(nombre_fichero)

for fichero in CONF_3_ficheros:
    nombre_fichero = ruta_principal + fichero
    tiempo_inicio = time.time()
    programa_principal(nombre_fichero)
```

Anexo B

Script utilizado en R

B.1. Cálculo de la media y la desviación típica del tiempo de ejecución

```
install.packages("ggplot2")
library(ggplot2)
library(Rcmdr)

# Leemos los datos de cada configuracion
datos_conf1 <- read.table("CONF_1_resultados.txt", header = FALSE)
datos_conf2 <- read.table("CONF_2_resultados.txt", header = FALSE)
datos_conf3 <- read.table("CONF_3_resultados.txt", header = FALSE)

# Damos nombres a las columnas
colnames(datos_conf1) <- c("seed", "numero_nodos", "numero_arcos",
                           "num_ofertas", "cost_min", "cost_max", "cap_min",
                           "cap_max", "tiempoEjecucion")
colnames(datos_conf2) <- c("seed", "numero_nodos", "numero_arcos",
                           "num_ofertas", "cost_min", "cost_max", "cap_min",
                           "cap_max", "tiempoEjecucion")
colnames(datos_conf3) <- c("seed", "numero_nodos", "numero_arcos",
                           "num_ofertas", "cost_min", "cost_max", "cap_min",
                           "cap_max", "tiempoEjecucion")

# Cambiamos los niveles de 'numero_arcos'
datos_conf1$numero_arcos <- factor(datos_conf1$numero_arcos,
                                   levels = c(1020, 2020, 3020, 4020, 5020),
                                   labels = c('1000', '2000', '3000', '4000', '5000'))

# Creamos la variable 'capacidad'
datos_conf3$capacidad <- factor(datos_conf3$cap_min,
                                levels = c(100, 200, 400, 600, 800),
                                labels = c('(100,200)', '(200,400)', '(400,600)',
                                           '(600,800)', '(800,1000)'))
```

```
##CONFIGURACIÓN 1:
# Calculamos el tiempo medio y la desviacion estandar para cada
# numero de arcos en la Configuracion 1
media_desviacion_conf1 <- function(datos) {
  resumen <- aggregate(tiempoEjecucion ~ numero_arcos, data = datos,
                       FUN = function(x) c(mean = mean(x), desviacion = sd(x)))
  return(resumen)
}
```

```r
# Aplicamos esta funcion a la Configuracion 1
resumen_conf1 <- media_desviacion_conf1(datos_conf1)
print(resumen_conf1)

# Creamos una tabla para la Configuracion 1
tabla_conf1 <- data.frame(
  numero_arcos = resumen_conf1$numero_arcos,
  tiempo_medio = resumen_conf1$tiempoEjecucion[, "mean"],
  desviacion_tipica = resumen_conf1$tiempoEjecucion[, "desviacion"]
)

print(tabla_conf1)

# Generamos un gráfico de puntos con líneas verticales para la Configuración 1
grafico_conf1 <- ggplot(tabla_conf1, aes(x = factor(numero_arcos), y = tiempo_medio)) +
  geom_point(color = "blue", size = 3) +
  geom_segment(aes(xend = as.numeric(factor(numero_arcos)), yend = 0),
               color = "blue", size = 1) +
  labs(title = "Tiempo medio de resolución por número de arcos",
       x = "Número de arcos",
       y = "Tiempo medio") +
  theme_minimal() +
  scale_y_continuous(limits = c(0, 6)) +
  theme(
    plot.title = element_text(size = 18, face = "bold", margin = margin(b = 15)),
    axis.title.x = element_text(size = 16, margin = margin(t = 15)),
    axis.title.y = element_text(size = 16, margin = margin(r = 15)),
    axis.text.x = element_text(size = 14, margin = margin(t = 10)),
    axis.text.y = element_text(size = 14, margin = margin(r = 10))
  )

print(grafico_conf1)

# Guardamos el gráfico como archivo PNG
ggsave("grafico_conf1.png", plot = grafico_conf1, width = 10, height = 8)

##CONFIGURACIÓN 2:
# Calculamos el tiempo medio y la desviacion estandar para cada
# numero de ofertas en la Configuracion 2
media_desviacion_conf2 <- function(datos) {
  resumen <- aggregate(tiempoEjecucion ~ num_ofertas, data = datos,
                       FUN = function(x) c(mean = mean(x), desviacion = sd(x)))
  return(resumen)
}

# Aplicamos esta funcion a la Configuracion 2
resumen_conf2 <- media_desviacion_conf2(datos_conf2)
print(resumen_conf2)

# Creamos una tabla para la Configuracion 2
tabla_conf2 <- data.frame(
  num_ofertas = resumen_conf2$num_ofertas,
  tiempo_medio = resumen_conf2$tiempoEjecucion[, "mean"],
  desviacion_tipica = resumen_conf2$tiempoEjecucion[, "desviacion"]
)

print(tabla_conf2)
```

```r
# Generamos un grafico de puntos con lineas verticales para la Configuracion 2
grafico_conf2 <- ggplot(tabla_conf2, aes(x = factor(num_ofertas), y = tiempo_medio)) +
  geom_point(color = "blue", size = 3) +
  geom_segment(aes(xend = as.numeric(factor(num_ofertas)), yend = 0),
               color = "blue", size = 1) +
  labs(title = "Tiempo medio de resolución por número de nodos con oferta",
       x = "Número de nodos con oferta",
       y = "Tiempo medio") +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 18, face = "bold", margin = margin(b = 15)),
    axis.title.x = element_text(size = 16, margin = margin(t = 15)),
    axis.title.y = element_text(size = 16, margin = margin(r = 15)),
    axis.text.x = element_text(size = 14, margin = margin(t = 10)),
    axis.text.y = element_text(size = 14, margin = margin(r = 10))
  )

print(grafico_conf2)

# Guardamos el grafico como archivo PNG
ggsave("grafico_conf2.png", plot = grafico_conf2, width = 10, height = 8)


##CONFIGURACIÓN 3:
# Calculamos el tiempo medio y la desviacion estandar para cada
# combinación de capacidad en la Configuracion 3
media_desviacion_conf3 <- function(datos) {
  resumen <- aggregate(tiempoEjecucion ~ capacidad, data = datos,
                       FUN = function(x) c(mean = mean(x), desviacion = sd(x)))
  return(resumen)
}

# Aplicamos esta funcion a la Configuracion 3
resumen_conf3 <- media_desviacion_conf3(datos_conf3)
print(resumen_conf3)

# Creamos una tabla para la Configuracion 3
tabla_conf3 <- data.frame(
  capacidad = resumen_conf3$capacidad,
  tiempo_medio = resumen_conf3$tiempoEjecucion[, "mean"],
  desviacion_tipica = resumen_conf3$tiempoEjecucion[, "desviacion"]
)

print(tabla_conf3)

# Generamos un gráfico de puntos con líneas verticales para la Configuración 3
grafico_conf3 <- ggplot(tabla_conf3, aes(x = factor(capacidad), y = tiempo_medio)) +
  geom_point(color = "blue", size = 3) +
  geom_segment(aes(xend = as.numeric(factor(capacidad)), yend = 0),
               color = "blue", size = 1) +
  labs(title = "Tiempo medio de resolución por capacidad",
       x = "Capacidad mínima y máxima",
       y = "Tiempo medio") +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 18, face = "bold", margin = margin(b = 15)),
    axis.title.x = element_text(size = 16, margin = margin(t = 15)),
    axis.title.y = element_text(size = 16, margin = margin(r = 15)),
    axis.text.x = element_text(size = 14, margin = margin(t = 10)),
    axis.text.y = element_text(size = 14, margin = margin(r = 10))
  )

print(grafico_conf3)

# Guardamos el grafico como archivo PNG
ggsave("grafico_conf3.png", plot = grafico_conf3, width = 10, height = 8)
```

B.2. Test de normalidad

```
##CONFIGURACION 1:
# Convertimos 'numero_arcos' a factor
datos_conf1$numero_arcos <- as.factor(datos_conf1$numero_arcos)

# Test de normalidad para 'numero_arcos'
normalityTest(tiempoEjecucion ~ numero_arcos, test="shapiro.test",
              data=datos_conf1)


##CONFIGURACION 2:
# Convertimos 'num_ofertas' a factor
datos_conf2$num_ofertas <- as.factor(datos_conf2$num_ofertas)

# Test de normalidad para 'num_ofertas'
normalityTest(tiempoEjecucion ~ num_ofertas, test="shapiro.test",
              data=datos_conf2)


##CONFIGURACION 3:
# Convertimos 'capacidad' a factor
datos_conf3$capacidad <- as.factor(datos_conf3$capacidad)

# Test de normalidad para 'capacidad'
normalityTest(tiempoEjecucion ~ capacidad, test = "shapiro.test",
              data = datos_conf3)
```

B.3. Test de Levene

```
##CONFIGURACION 1:
# Test igualdad de varianzas
print(leveneTest(tiempoEjecucion ~ numero_arcos, data=datos_conf1,
                 center="median"))


##CONFIGURACION 2:
print(leveneTest(tiempoEjecucion ~ num_ofertas, data=datos_conf2,
                 center="median"))


##CONFIGURACION 3:
print(leveneTest(tiempoEjecucion ~ capacidad, data=datos_conf3,
                 center="median"))
```

B.4. Test de ANOVA con aproximación de Welch

```r
##CONFIGURACION 1:
AnovaModel.1 <- aov(tiempoEjecucion ~ numero_arcos, data = datos_conf1)
summary(AnovaModel.1)
print(oneway.test(tiempoEjecucion ~ numero_arcos, data = datos_conf1)) # Welch test


##CONFIGURACION 2:
AnovaModel.2 <- aov(tiempoEjecucion ~ num_ofertas, data = datos_conf2)
summary(AnovaModel.2)
print(oneway.test(tiempoEjecucion ~ num_ofertas, data = datos_conf2)) # Welch test


##CONFIGURACION 3:
AnovaModel.3 <- aov(tiempoEjecucion ~ capacidad, data = datos_conf3)
summary(AnovaModel.3)
print(oneway.test(tiempoEjecucion ~ capacidad, data = datos_conf3)) # Welch test
```

B.5. Test de Tukey

```r
##CONFIGURACION 1:
res_tukey_conf1 <- TukeyHSD(AnovaModel.1)
print(res_tukey_conf1)


##CONFIGURACION 2:
res_tukey_conf2 <- TukeyHSD(AnovaModel.2)
print(res_tukey_conf2)


##CONFIGURACION 3:
res_tukey_conf3 <- TukeyHSD(AnovaModel.3)
print(res_tukey_conf3)
```