Algoritmos de tipo “subasta” para problemas de flujo en redes

Alicia García Martínez

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Director del trabajo: Pedro M. Mateo Collazos
12 de septiembre de 2022

Summary.

Linear network problems are of great importance in our area of interest, Operations Research. However, its relevance and usefulness in other fields such as engineering or management should be emphasized.

Of the many problems that can be faced in the area of network flow problems, the project will focus on two of them: the assignment problem and the minimum cost flow problem. Both of them, besides having theoretical significance, are very useful in practice, being able to reach a large number of different applications.

The assignment problem consists of finding the way to assign a number of origins (people, tasks, etc) to the same number of destinations (tasks, machines) with the aim to optimize the cost of linking a given origin with a given destination. The assignment must be given in such a way that each source is assigned to a single destination, and each destination to a single source. The assignment problem solves practical cases of notable importance, among which we highlight: the assignment of vendors to a set of territories, of employees of a company to tasks, of construction factories to products or of bidders to contracts.

The objective of the minimum cost flow problem is set to determine the optimal way to send goods (flow) through a network of arcs in order to satisfy the demand at the lowest cost. As it is mentioned in previous paragraphs, such problems have a relevant position among the optimization network problems since it involves the resolution of a variety of practical situations in real life. In addition to that, the mentioned examples starting with the distribution of a product from factories to warehouses, continuing to the optimization of a hydraulic network system to achieve supplying geographical points, or ending with the distribution of the flow of raw materials through several sales points or the optimal circulation of a train network through tracks and stations.

The algorithms that solve these two problems are based on Graph Theory and until the end of the seventies there were basically two types of algorithms that were able to optimize flow problems in networks: the simplex method and the simplex-dual method.

Furthermore, other types of algorithms were developed that challenged the aforementioned algorithms, both from a practical and theoretical point of view. In this paper we will focus on one of them, the auction algorithm.

The auction algorithm arose in 1979 [4] to solve the classical assignment problem and from it extensions to solve many other problems were derived. Extensions are derived to solve many other classical network problems such as the minimum cost flow problem.

These extensions are mainly based on performing transformations of the minimum cost flow problem to equivalent assignment problems. Since these transformations are possible, any method that solves the assignment problem will solve the minimum cost flow problem. It is for the reason that the assignment problem, despite its simplicity, is a good starting point for algorithmic ideas in linear programming. The auction algorithms have proven to be very efficient in practice for the problems of our study.

For all these reasons, the report is structured in 4 parts.

In the first chapter, we present in detail the two problems we are going to deal with and we introduce the duality theory. The duality theory associates a dual problem to each primal problem, and thanks to this we can deduce the complementary slackness conditions for the two problems, which are of fundamental importance to develop the auction algorithms that we will see in the next two chapters.

Secondly, chapter two will be composed of a brief introduction on the nature of the auction algorithms and then we begin to develop the Naif auction algorithm for the assignment problem, which is a predecessor of the general algorithm and it is introduced to facilitate the understanding of the general algorithm. The general auction algorithm for the assignment problem intuitively operated, resembling a real auction in which bidders bid on objects and raise their prices p_j, which are the dual variables. A detailed iteration of how it works will be described. Optimality and feasibility properties are then commented.

Therefore, in chapter three the auction algorithm for minimum cost flow problems is developed. For the correct understanding of this method, it has been necessary to introduce previous concepts and results. The general algorithm is described systematically, introducing in Appendix B a detailed and visual example of its operation. We will see that in each iteration of the algorithm the flows of a set of unblocked arcs are increased and if the slackness conditions do not allow to perform this increase at the beginning of the iteration, the prices of the nodes of the set are previously increased and then the flow increase is performed. Both price and flow increments are always performed on nodes that have excess. As for the assignment problem, optimality and feasibility conditions are studied.

Once described the general auction algorithm for the minimum cost flow problem, we will focus on an extension of it, the ε-relaxation method. The differences of this method with respect to the general method lie in the fact that here we operate in each iteration on a single node with excess, and not on a set of nodes.

The ε-relaxation algorithm turns out to be very efficient in computational aspects and that is why we have carried out the FORTRAN implementation of this method to solve flow problems at minimum cost, thus fulfilling another of our objectives.

Appendix C provides all the codes that we have built throughout the work. These manage to generate well-structured mínimum cost flow problems and then optimize them.

Thanks to these codes we have generated 434 minimum cost flow problems of different characteristics and solved them with the implemented auction algorithm.

Summarizing, and with the aim to reach a final conclusion Chapter 4 will discuss the data we have obtained to perform a small on the behaviour of the method. For this purpose we have used Rstudio to plot the variables we wish to study and Rcommander to obtain analytical numerical summaries.

Índice general

Capítulo 1

Problema de asignación y problema de flujo a costo mínimo.

1.1. El problema de asignación.

Este problema se presentó como un caso particular del problema de transporte, estudiado en la asignatura "Investigación Operativa" del tercer curso del grado de Matemáticas, si bien no se estudiaron algoritmos eficientes para su resolución[1].

El problema consiste en las siguientes idea, supongamos que tenemos n personas y n objetos y los queremos emparejar uno a uno. Hay un beneficio a_{ij} por unir la persona i con el objeto j, y queremos asignar personas con objetos de forma que el benificio total se maximice. Una de las restricciones del problema es que la persona i se puede unir con el objeto j solo si el par (i, j) pertenece a un conjunto dado A. Si un par $(i, j) \in A$, existe un arco conectando la persona i con el objeto j.

Matemáticamente, queremos encontrar un conjunto de pares persona-objeto $\{(1, j_i), ..., (n, j_n)\} \in A$ tal que los objetos $j_1, ..., j_n$ sean todos distintos y el beneficio total $\sum_{i=1}^{n} a_{ij_i}$ sea máximo.

Podemos asociar cualquier asignación con el conjunto de variables de decisión $\{x_{ij} \mid (i, j) \in A\}$, donde $x_{ij} = 1$ si la persona i ha sido asignada con el objeto j y $x_{ij} = 0$ en otro caso. Formulamos el problema de asignación como el problema lineal

$$\text{Maximizar} \sum_{\{(i,j)\in A\}} a_{ij}x_{ij} \tag{1.1a}$$

Sujeto a:

$$\sum_{\{j|(i,j)\in A\}} x_{ij} = 1, \qquad \forall i = 1, \ldots, n, \tag{1.1b}$$

$$\sum_{\{i|(i,j)\in A\}} x_{ij} = 1, \qquad \forall j = 1, \ldots, n, \tag{1.1c}$$

$$0 \leq x_{ij} \leq 1, \qquad \forall (i, j) \in A. \tag{1.1d}$$

En terminología de problemas de flujo en redes, la variable x_{ij} representa el flujo que recorre el arco (i, j). La restricción (1.1b) indica que el flujo total que sale del nodo i debe ser igual a 1. Análogamente, la segunda restricción (1.1c) indica que el flujo total que entra al nodo j también debe ser igual a 1. Estas dos restricciones definen la *oferta* de flujo de los nodos i y la *demanda* de flujo de los nodos j, respectivamente.

[1] El problema de asignación se puede resolver mediante el algoritmo de transporte, pero no resulta eficiente debido a la alta degeneración de las bases.

En realidad, se deberían restringir aún más los valores de las variables x_{ij} en (1.1d), para que sean los valores enteros 0 ó 1, pero el problema lineal de asignación tiene una notable propiedad; si tiene una solución factible, entonces tiene una solución óptima donde todas las variables x_{ij} son 0 ó 1[2].

Otra importante propiedad del problema de asignación es que se puede representar con un grafo bipartito, como el de la Figura 1.1, de manera que hay $2n$ nodos divididos en dos grupos: n correspondientes a las personas y n correspondientes a los objetos.

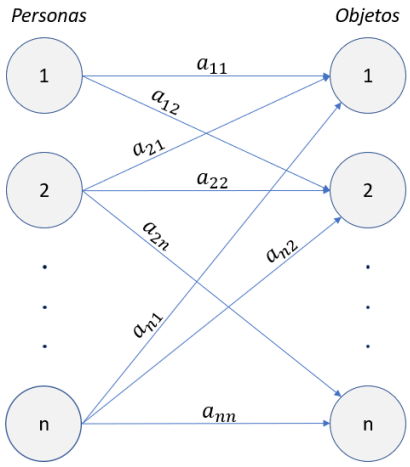

Figura 1.1: Grafo bipartito para un problema de asignación.

1.2. Problema de flujo a costo mínimo.

El problema consiste en encontrar un conjunto de arcos pertenecientes a una red dirigida $G = (N,A)$, donde N el conjunto total de nodos y A el conjunto total de arcos, que minimicen la función lineal de costos sujeta a restricciones. Esto es,

$$\text{Minimizar} \sum_{(i,j)\in A} a_{ij}x_{ij} \tag{1.2a}$$

Sujeto a:

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = s_i \qquad \forall i \in N, \tag{1.2b}$$

$$b_{ij} \leq x_{ij} \leq c_{ij}, \qquad \forall (i,j) \in A. \tag{1.2c}$$

Las variables x_{ij} son los flujos, y están definidos para cada arco $(i,j) \in A$ y a_{ij} es el coeficiente de costo, es el costo por unidad de flujo que atraviesa el arco (i,j). Así, la función objetivo (1.2a) representa el costo total de enviar los diferentes flujos a través de los arcos.

Llamamos a (1.2b) *restricción de conservación de flujo*, esta define el suministro s_i de un nodo i como el flujo total que sale del nodo i menos el flujo total que entra al nodo i. Si $s_i > 0$, decimos que es un nodo con oferta mientras que si $s_i < 0$, decimos que es un nodo con demanda. Cuando $s_i = 0$ decimos que el nodo es de transbordo. La segunda restricción (1.2c) es la *restricción de capacidad*, que se encarga de limitar la cantidad de flujo que puede fluir por cada arco (i,j). Asumiremos que los escalares a_{ij}, b_{ij}, c_{ij} y s_i son enteros y además consideraremos que $b_{ij} \geq 0$ y $c_{ij} \geq 0$.

[2]El problema de asignación es un caso particular del problema de transporte. En la asignatura Investigación operativa [1] ya se estudió dicha propiedad para el problema de transporte

Un vector de flujo que satisface estas dos restricciones se denomina vector de flujo factible. Si el problema de flujo a costo mínimo tiene al menos un vector de flujo factible, decimos que el problema lineal es factible. Una condición necesaria para la factibilidad es que[3] $\sum_{i\in N} s_i = 0$.

Existen numerosas transformaciones equivalentes para el problema lineal de flujo a costo mínimo. El problema formulado anteriormente puede ajustarse para conseguir que el límite inferior de flujo sea cero. En nuestro caso, vamos a trabajar con el problema de flujo a costo mínimo con límite inferior de flujo nulo; es decir, siguiendo la terminología anterior, debemos conseguir un problema equivalente con $b_{ij} = 0$.

La cota inferior b_{ij} puede ser cambiada a cero reemplazando x_{ij} por $x_{ij} - b_{ij}$ y ajustando el límite superior y el suministro

$$c_{ij} \leftarrow c_{ij} - b_{ij}$$

$$s_i \leftarrow s_i - \sum_{\{j|(i,j)\in A\}} b_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji}$$

Los flujos óptimos y el valor óptimo del problema original son obtenidos añadiendo b_{ij} al flujo óptimo de cada arco (i,j) y sumando $\sum_{\{(i,j)\in A\}} a_{ij}b_{ij}$ al valor óptimo del problema transformado. El problema con estas modificaciones tiene la siguiente forma:

$$\begin{aligned}
&\text{Minimizar } \sum_{(i,j)\in A} a_{ij}x_{ij} \\
&\text{Sujeto a:} \\
&\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = s_i \qquad \forall i \in N, \\
&0 \le x_{ij} \le c_{ij}, \qquad \forall (i,j) \in A.
\end{aligned} \tag{1.3}$$

1.3. Teoría de la dualidad. Problemas duales.

La teoria de la dualidad nos permite asociar a cada problema original de programación lineal, al que llamamos primal y denotaremos por $[P]$, otro problema lineal denominado dual[4], que denotamos por $[D]$. Las soluciones de ambos problemas van a estar estrechamente relacionadas. Además, la teoría de la dualidad nos proporciona herramientas para hacer comprobaciones de optimalidad de soluciones y nos permite realizar importantes interpretaciones económicas de los problemas de programación lineal.

1.3.1. Problema dual de asignación.

Para construir el problema dual $[D]$ del problema de asignación primal [P] dado en (1.1) asociamos una variable dual con cada restricción del primal, utilizaremos $q_i,\ i = 1,\dots,n$ para indicar las variables asociadas a las primeras n restricciones y $p_i,\ i = 1,\dots,n$ para las variables asociadas a las segundas n restricciones. Las variables para el problema dual se llamarán *precios*.

El problema dual del problema de asignación es el siguiente,

$$[D] \begin{cases} \text{Minimizar} & \sum_{i=1}^n q_i + \sum_{i=1}^n p_j & \text{(1.4a)} \\ \text{Sujeto a:} & & \\ & q_i + p_j \ge a_{ij} \qquad \forall (i,j) \in A, & \text{(1.4b)} \\ & q_i, p_i \in \mathbb{R} \qquad \forall i \in N. & \text{(1.4c)} \end{cases}$$

[3] Condiciones necesarias y suficientes de factibilidad de problemas de programación lineal se estudiaron en la asignatura "Grafos y Combinatoria" del primer curso del grado de Matemáticas. Podemos encontrar dichos resultados en [2].

[4] Los elementos de Teoría de Dualidad para Programación Lineal se estudiaron en [1], por lo que no se entrará en analizar el porqué de los problemas duales que aparecen a lo largo de esta sección, sino solo en detallar las consecuencias que de ellos derivan.

Ahora, vamos a dar herramientas para poder hallar las condiciones de holgura complementaria del problema de asignación y para reescribir el problema de forma más sencilla. Las condiciones de holgura complementaria nos asegurarán la optimalidad de la solución del problema. En primer lugar, se encuncia el Teorema de condiciones de holgura complementaria en su forma general.

Teorema 1.1. Condiciones de holgura complementaria.

Dadas $\bar{\mathbf{x}}$ *e* $\bar{\mathbf{y}}$ *soluciones factibles de un problema de programación lineal de mínimo y su dual, entonces* $\bar{\mathbf{x}}$ *e* $\bar{\mathbf{y}}$ *son óptimas para sus problemas respectivos si y solo si se verifica*

$$\bar{x}_j \bar{v}_j = 0 \qquad \forall j = 1, \ldots, n. \tag{1.5a}$$

$$\bar{y}_i \bar{u}_i = 0 \qquad \forall i = 1, \ldots, m. \tag{1.5b}$$

donde $\bar{\mathbf{u}}$ *y* $\bar{\mathbf{v}}$ *son las variables de holgura de las restricciones del primal y del dual, respectivamente.*

Antes de continuar desarrollando las condiciones de holgura complementaria para el problema de asignación, se presenta un resultado que nos permite reescribir el problema (1.4) de manera más útil.

Lema 1.2. *Dados unos valores* p_j con $j = 1, \ldots, n$. *cualesquiera, se puede construir una solución factible de* (1.4) *tomando*

$$q_i = \max_{j \in A(i)} \{a_{ij} - p_j\} \quad \forall i = 1, \ldots, n, \tag{1.6}$$

donde $A(i) = \{j | (i, j) \in A\}$ *es el conjunto de objetos que pueden ser asignados a la persona i. Además, fijados los valores* $p_j, j = 1, \ldots, n$ *la solución definida es la que proporciona el mayor valor de* (1.4a)

Demostración. Fijados los valores p_j, despejamos q_i en la primera restricción (1.4b) del problema dual de asignación, obteniendo;

$$q_i \geq a_{ij} - p_j \qquad \forall (i, j) \in A,$$

Luego q_i es mayor o igual que todos los valores $\{a_{ij} - p_j\}$ con $(i, j) \in A$ y por ello, podemos expresar q_i como el máximo de todos ellos,

$$q_i = \max_{j \in A(i)} \{a_{ij} - p_j\} \quad \forall i = 1, \ldots, n.$$

Por lo tanto estos valores q_i, junto a los valores p_j fijados, construyen una solución factible del problema. □

En el caso en el que nos encontramos, la segunda ecuación del Teorema 1.1 no se debe considerar ya que en el problema de asignación primal de máximo todas las restricciones son de igualdad, y por tanto, no hay variables de holgura $\bar{u}_j$. Consideramos únicamente la primera ecuación (1.5a) del Teorema 1.1 , donde intervienen las variables de holgura del problema dual.

$$x_{ij} v_j = x_{ij}(a_{ij} - q_i - p_j) = 0, \qquad \forall (i, j) \in A. \tag{1.7}$$

Y, teniendo en cuenta (1.6) y que las variables del primal son enteras y pertenecen al intervalo $[0, 1]$, la ecuación anterior (1.7) se puede reescribir, dado un conjunto de precios $p_j, j = 1, \ldots, n$ como

$$a_{ij} - p_j = \max_{j \in A(i)} \left\{a_{ij} - p_j\right\}, \quad \forall i = 1, \ldots, n, \quad (i, j) \in A \text{ y } x_{ij} > 0. \tag{1.8}$$

De ahora en adelante, este es el formato que utilizaremos de las condiciones de holgura complementaria para el problema de asignación.

Estas condiciones quieren decir, en términos del problema de asignación, que el objeto j tiene un precio p_j, y la persona que sea asignada al objeto j debe pagar ese precio p_j. El valor neto del objeto j para la persona i es $a_{ij} - p_j$, pero la persona i lógicamente quiere ser asignada a un objeto j_i que le reporte máximo beneficio, o lo que es lo mismo, que tenga el máximo valor neto.

Teniendo en cuenta la expresión de q_i que nos da el lema anterior, podemos expresar el problema dual del problema de asignación de la siguiente forma:

$$\text{mín} \left\{ \sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j \right\} \quad p_j \in \mathbb{R} \quad j = 1, \ldots, n. \tag{1.9}$$

Una vez definidas las condiciones de holgura complementaria estamos en condiciones de enunciar el siguiente teorema.

Teorema 1.3. *Si una asignación factible y un conjunto de precios $p_j, j = 1, \ldots, n$ del problema* (1.1) *satisfacen las condiciones de holgura complementaria* (1.8) *para todas las personas i entonces la asignación es óptima y dichos precios $p_j, j = 1, \ldots, n$ son una solución óptima del problema dual* (1.9). *Además el beneficio de la asignación óptima y el costo dual óptimo son iguales.*

Aunque este resultado es un caso particular de resultados estudiados en la asignatura "Investigación Operativa", y por tanto ya estaría demostrado, por su especial formato y su distinto enfoque se proporciona una demostración específica en [3] y se muestra en el Apéndice A[5].

1.3.2. Problema dual del problema de flujo a costo mínimo.

En esta sección se va a realizar un proceso similar al de la sección 1.3.1, esta vez para el problema de flujo a costo mínimo.

Consideramos el problema primal [P] de flujo a costo mínimo (1.3). Para su dual [D], asociamos una variable dual con cada restricción del primal, exceptuando la restricción de no negatividad de los flujos. Asociamos las variables p_i, $i = 1, \ldots, n$ con las restricción de conservación de flujo y la variable q_{ij} con $(i, j) \in A$ con la restricción de límite de capacidad superior, obteniendo un problema de la forma[6],

$$[D] \begin{cases} \text{Maximizar} \quad \sum_{i \in N} s_i p_i + \sum_{(i,j) \in A} c_{ij} q_{ij} & \text{(1.10a)} \\ \text{Sujeto a:} & \\ p_i - p_j + q_{ij} \leq a_{ij}, & \text{(1.10b)} \\ p_i \in \mathbb{R} \qquad \forall i \in N, \qquad q_{ij} \leq 0 \quad \forall (i,j) \in A. & \end{cases}$$

Utilizando el teorema de las Condiciones de Holgura Complementaria 1.1 expuesto en la anterior sección y teniendo en cuenta el siguiente lema, vamos a conseguir reescribir el problema dual de una forma más útil y sencilla.

Lema 1.4. *Dado un conjunto de valores p_i cualesquiera, se puede construir una solución factible de* (1.10) *tomando*

$$q_{ij} = \begin{cases} a_{ij} + p_j - p_i, & \text{si } p_i > a_{ij} + p_j. \\ 0, & \text{si } p_i \leq a_{ij} + p_j. \end{cases}$$

o equivalentemente,

$$q_{ij} = \text{mín}\{a_{ij} + p_j - p_i, 0\}.$$

Además, fijados los valores p_i la solución definida es la que proporciona el mayor valor de (1.10a).

[5] Si no se indica lo contrario, las demostraciones de todos los resultados que aparecen a lo largo de los capítulos se pueden encontrar en [3]

[6] En este problema se ha realizado un cambio de notación con el problema (1.1). Ahora no hay dos juegos de variables p_i.

Demostración. Fijando unos valores p_i cualesquiera y despejando q_{ij} de la primera restricción (1.10b) del problema dual del problema de flujo a costo mínimo, obtenemos la siguiente expresión,

$$q_{ij} \leq a_{ij} + p_j - p_i.$$

Tal y como se ha definido el problema dual, sabemos que $q_{ij} \leq 0$ y $c_{ij} \geq 0$ y como en la función objetivo estamos maximizando, queremos que la cantidad "restada" sea mínima y por ello,

$$q_{ij} = \text{mín}\{a_{ij} + p_j - p_i, 0\}.$$

Tal y como hemos construido estos valores, vemos que proporcionan una solución factible y además es la que maximiza el valor de la función objetivo del problema dual. □

De manera análoga al caso del problema de asignación, hallamos las condiciones de holgura complementaria para el problema de flujo a costo mínimo, considerando de nuevo las dos ecuaciones que nos aportaba el Teorema 1.1, tenemos que

$$\begin{aligned} q_{ij}(c_{ij} - x_{ij}) &= 0, \\ x_{ij}(p_j - p_i - q_{ij} + a_{ij}) &= 0. \end{aligned}$$

y, teniendo en cuenta el lema anterior, estas ecuaciones se satisfacen si para un conjunto de precios $p_i \in \mathbb{R}, \quad i = 1, \ldots, n$ y una solución factible x_{ij} de (1.3) se tiene:

$$\begin{aligned} x_{ij} &= c_{ij} && \text{si } p_i - p_j > a_{ij} \\ 0 \leq x_{ij} &\leq c_{ij} && \text{si } p_i - p_j = a_{ij} \\ x_{ij} &= 0 && \text{si } p_i - p_j < a_{ij}. \end{aligned} \tag{1.11}$$

Este es el formato de las condiciones de holgura complementaria del problema de flujo a costo mínimo que manejaremos a partir de este momento y que vemos representado en la Figura 1.2.

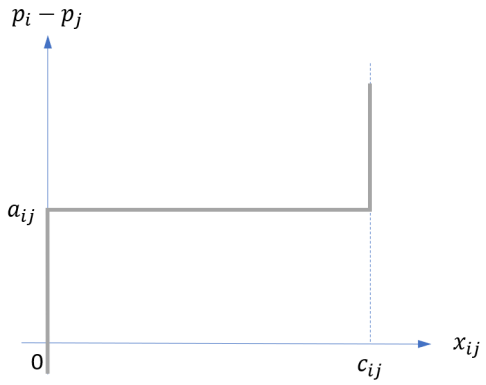

Figura 1.2: Condiciones de holgura complementaria. Para cada arco (i, j), el par $(x_{ij}, p_i - p_j)$ debe encontrarse en la gráfica.

Teniendo en cuenta los desarrollos anteriores, podemos expresar el problema dual del problema de flujo a costo mínimo de la siguiente manera

$$[D] \begin{cases} \text{Maximizar} & \sum_{i \in N} s_i p_i - \sum_{(i,j) \in A} c_{ij}\, \text{mín}\{a_{ij} + p_j - p_i, 0\} \\ \text{Sujeto a:} & \\ p_j \in \mathbb{R} \quad j = 1, \ldots, n. \end{cases} \tag{1.12}$$

Teorema 1.5. *Si un vector de flujo factible x^* y un vector de precios p^* satisfacen las condiciones de holgura complementaria dadas en* (1.11)*, entonces x^* es una solución óptima para el problema de flujo a costo mínimo dado en* (1.3) *y p^* es una solución óptima para su problema dual, dado en* (1.12)*.*

.

Capítulo 2

Algoritmo de subasta para el problema de asignación.

2.1. Idea básica del algoritmo de subasta.

Los algoritmos más comunes para resolver problemas de flujo en redes son los que mejoran el costo primal. El algoritmo de subasta tiene una naturaleza primal-dual y en algunos aspectos se asemeja a métodos muy conocidos como por ejemplo el método húngaro [1], que fue el primer método especializado para resolver el problema de asignación, pero es sustancialmente diferente en otros. Estrictamente hablando, en los algoritmos de subasta no hay una mejora de costos primales o duales. En cualquier iteración se puede deteriorar tanto el costo primal como el costo dual. El proceso finaliza cuando se encuentra una solución óptima del problema primal.

Al tratarse de un algoritmo de tipo primal-dual, además de los precios, los algoritmos de subasta también iteran sobre los flujos, que se relacionan con los precios a través de la propiedad ε-holgura complementaria, esta es una forma aproximada de la holgura complementaria, la cual hemos establecido en el Capítulo 1. Trataremos esta propiedad en profundidad posteriormente.

El proceso algorítmico que desarrollaremos a continuación es muy intuitivo y se asemeja a una subasta en la que los agentes económicos compiten por los recursos realizando ofertas cada vez más altas, generando secuencias de precios cada vez mayores. Los precios pueden ser vistos como las variables duales. El algoritmo termina en un número finito de iteraciones después de que los precios de los recursos alcanzan niveles en los que no es posible realizar más ofertas.

Antes de desarrollar el algoritmo de subasta general, se va a explicar un algoritmo básico que se denomina subasta Naif, que servirá para introducir la necesidad de la ε-Holgura Complementaria, propiedad con la que se desarrollará el algoritmo general de subasta.

2.2. Asignación mediante subasta Naif.

Nos situamos en el problema simétrico de asignación definido en (1.1). El objetivo del algoritmo de subasta Naif es encontrar una asignación factible, junto a un vector correspondiente de precios.

Entendemos la asignación como un conjunto, que denotaremos S, de pares persona-objeto (i, j), tal que cada persona i y cada objeto j están implicados, como máximo, en un par del conjunto. Se dará la asignación factible cuando S contenga n pares. En caso contrario, diremos que la asignación es parcial. La asignación óptima será obviamente la que maximice el valor de la función objetivo (1.1a) del problema de asignación.

El algoritmo Naif procede realizando iteraciones. En cada una de ellas se genera un vector de precios y una asignación parcial. Veremos a continuación que al principio de cada una de las iteraciones se satisface la condición de holgura complementaria del problema de asignación (1.8) para todos los pares (i, j) que componen dicha asignación parcial. Si todas las personas quedan asignadas en una determinada iteración, el algoritmo termina. De lo contrario, este realiza una nueva iteración, tal y como se describe a continuación.

2.2.1. Descripción de una iteración del algoritmo subasta Naif.

Alguna persona que aún no ha sido emparejada, por ejemplo la persona i, se selecciona. Esta persona encuentra un objeto, al que llamaremos j_i que le reporta el máximo beneficio; esto es,

$$j_i = \arg \max_{j \in A(i)} \{a_{ij} - p_j\}. \tag{2.1}$$

y entonces:

- La persona i se asigna al mejor objeto j_i. Este objeto j_i podía haber sido asignado previamente a otra persona en una iteración anterior, y en ese caso dicha persona pasaría a no estar emparejada y por tanto no estaría en la asignación parcial S una vez acabada la iteración. Por tanto, el algoritmo todavía no se encuentra en condiciones de terminar y en consecuencia, se iniciaría otra nueva iteración.
- Se establece el nuevo precio de j_i, $p_{j_i} + \gamma_i$, donde γ_i es el incremento de la puja, que se calcula como

  $$\gamma_i = v_i - w_i.$$

  v_i es el valor que reporta el mayor beneficio a la persona i, y w_i es el segundo mejor valor,

  $$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}. \tag{2.2}$$

  $$w_i = \max_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\}. \tag{2.3}$$

  En el caso en el que en $A(i)$ solo existiera un objeto j_i, definiríamos $w_i = -\infty$. Vemos en la Figura 2.1 como sería el proceso de elección de γ_i.

  Notar que como p_{j_i} ha aumentado, el valor $a_{ij_i} - p_{j_i}$ que reporta el objeto j_i a la persona i decrece. γ_i es el mayor incremento que puede crecer p_{j_i}, manteniendo la propiedad de que j_i reporta el máximo beneficio a i. Este proceso se repite hasta que se tiene una asignación factible S. Se debe destacar que γ_i nunca es negativo, ya que siempre se cumple que v_i es un valor mayor que w_i.

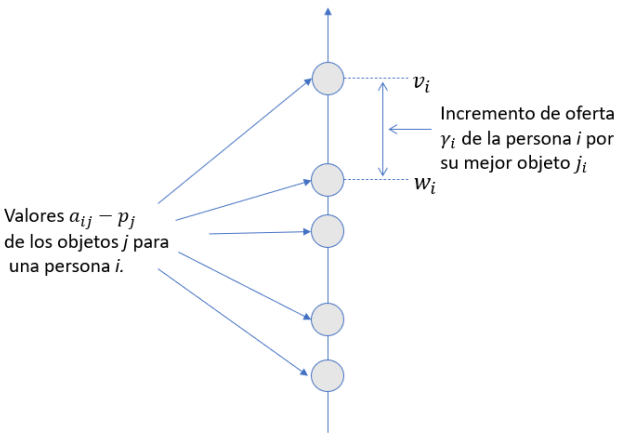

Figura 2.1: Elección del incremento de puja γ_i.

Podemos ver que cada iteración del algoritmo se asemeja a lo que ocurre en una subasta real. El postor i aumenta el precio de un objeto por el incremento de puja γ_i. Además, a medida que se van desarrollando iteraciones sucesivas, los precios de los objetos tienden a incrementarse y es lógico que los incrementos de oferta y los aumentos de precios estimulen la competencia[1].

El algoritmo que se acaba de describir tiene un grave defecto: no siempre es convergente. Este error motivará al desarrollo de un algoritmo de subasta correcto introduciendo una condición de convergencia que solucionará el problema. El problema mencionado reside en que el incremento de puja γ_i es cero cuando dos o más objetos reportan el valor máximo a la persona i. Entonces se da el caso en el que varias personas disputan por objetos igualmente deseables, por tanto no aumentan sus precios en ninguna iteración y se crea un ciclo sin fin. Veamos un ejemplo sencillo que ilustra este problema.

Ejemplo 1. Consideramos un problema de asignación simétrico con $n = 3$. El algoritmo de subasta Naif debería conseguir una asignación factible. Tomamos $a_{ij} = k$ para todo par (i, j) con $i = 1,2,3$ y $j = 2,3$ y $a_{ij} = 0$ en otro caso.

Representamos el problema en forma matricial, donde las filas de la matriz representan personas, las columnas objetos y los elementos son los a_{ij}. Tendríamos la siguiente matriz junto con un vector de precios incial de $(0,0,0)$ con el que el algoritmo comienza a iterar.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & k & k \\ 0 & k & k \\ 0 & k & k \end{pmatrix} \end{array}$$

Siguiendo los pasos descritos anteriormente, calculamos el valor máx $\{a_{ij} - p_j\}$ para todas las personas $i = 1,2,3$. Observar que para cada persona hay dos valores máximos iguales. Seleccionamos uno de los valores máximos para las personas $i = 1,2$ y lo representamos mediante k^*. Tendremos la siguiente matriz, donde se aprecia que comenzamos con una asignación inicial que contiene los pares persona-objeto $(1,2)$ y $(2,3)$.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & k^* & k \\ 0 & k & k^* \\ 0 & k & k \end{pmatrix} \end{array}$$

Se puede ver en la matriz que la persona 3 todavía no ha sido asignada a ningún objeto porque los dos objetos que le aportan el máximo beneficio, ya han sido emparejados previamente. Seguimos los pasos descritos anteriormente para una iteración. La persona 3 puja por el objeto 2, por tanto este deja de estar asignado a la persona 1. Se actualiza el precio del objeto 2 por $p_2 = p_2 + \gamma_3$, pero como tanto $v_3 = k$ como $w_3 = k$, $\gamma_3 = v_3 - w_3 = 0$. Por tanto, para esta asignación que contiene los pares persona-objeto $(2,3)$y $(3,2)$, el vector de precios sigue siendo $(0,0,0)$.

En la siguiente iteración, la persona 1 puja por el objeto 2. Análogamente a la iteración anterior tenemos la situación de que el incremento de puja vuelve a ser cero. El vector de precios sigue sin aumentar y esta vez tenemos los pares $(1,2)$ y $(2,3)$ en la asignación parcial. La persona 3 queda de nuevo sin asignar, volviendo así a la situación incial.

Si continuáramos realizando iteraciones de este tipo sucesivamente, seguiríamos constantemente teniendo un vector de precios nulo y entraríamos en un bucle sin fin entre las personas 1 y 3, que pujarían alternativamente por el objeto 2, pero nunca le aumentarían su precio.

[1] Es una subasta "irreal" en el sentido de que se puja en paralelo por distintos objetos en lugar de ir de uno en uno.

2.2.2. ε-Holgura Complementaria (ε-HC)

Para garantizar la convergencia del algoritmo anterior, hay que replantear su funcionamiento. Para ello se introduce un mecanismo de mejora en el que cada puja, al igual que en una subasta verdadera, debe aumentar el precio del objeto por un incremento mínimo positivo. La formalización de esta propiedad se recoge en la siguiente definición.

Definición 1. Fijando un escalar positivo ε, decimos que una asignación parcial y un vector de precios p satisfacen las *condiciones ε-holgura complementaria* si

$$a_{ij_i} - p_{j_i} \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \varepsilon. \tag{2.4}$$

para todas las parejas (i, j_i) asignadas.

Para satisfacer esta condición, todas las personas deben ser asignadas a objetos que están a ε de reportar el máximo beneficio. De esta manera se consigue que en cada iteración el incremento de puja sea al menos igual a ε.

2.3. Algoritmo de subasta general.

Una vez definida la condición (2.4) que evitará los problemas de convergencia que nos encontrábamos en el algoritmo de subasta Naif, describimos el algoritmo de subasta, que terminará en un número finito de pasos. Este procede iterativamente y finaliza cuando se llega a una asignación factible satisfaciendo la condición de ε-HC. El desarrollo del método es similar al del algoritmo de subasta Naif, pero en este caso definimos el incremento de puja para que sea siempre al menos ε,

$$\gamma_i = v_i - wi + \varepsilon. \tag{2.5}$$

Con esto se satisface la condición de ε-HC. Este incremento es la máxima cantidad que satisface dicha propiedad. Incrementos más pequeños también serían válidos porque $\gamma_i \geq \varepsilon$, pero usando el mayor incremento posible, aceleramos el algoritmo. Esto es otra característica que encontramos semejante a la subasta real, que tiende a terminar antes cuanto más agresiva es la puja.

Al igual que hemos hecho en la Sección 2.2, vamos a describir una iteración genérica del algoritmo. Lo vemos primero de una manera esquemática, en formato pseudocódigo, explicando a continuación detalladamente los pasos de su funcionamiento.

Algorithm 1: Algoritmo de subasta general.

$S = \emptyset$;
while *S no factible* **do**
  $I = \{$Subespacio de personas no asignadas.$\}$;
  for $i \in I$ **do**
    $j_i = arg\,\text{máx}_{j \in A(i)} \{a_{ij} - p_j\}$;
    $v_i \leftarrow \text{máx}_{j \in A(i)} \{a_{ij} - p_j\}$;
    $w_i \leftarrow \text{máx}_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\}$;
    $\gamma_i \leftarrow v_i - w_i + \varepsilon$;
    $b_{ij_i} \leftarrow p_{j_i} + \gamma_i$;
    $P(j_i) \leftarrow i$;
  end
  for $j \leftarrow 1$ to n **do**
    $i_m = arg\,\text{máx}_{i \in P(j)} b_{ij}$;
    $S = S \setminus (*, j)$;
    $S = S \cup (i_m, j)$;
    $p_j = b_{im_j}$;
  end
end

Cuando el algoritmo comienza una nueva iteración, tenemos una asignación parcial S y un vector de precios satisfaciendo las condiciones ε-HC. Para realizar la primera iteración, podemos utilizar un vector arbitrario de precios p y una asignación parcial vacía, que trivialmente satisface (2.4).

Una iteración está formada por dos fases: la fase de licitación y la fase de asignación.

- FASE 1: Licitación.
  Sea I un subconjunto de personas i que no han sido asignadas en la asignación parcial S. Para cada persona $i \in I$ hallamos

$$j_i = arg \max_{j \in A(i)} \{a_{ij} - p_{j_i}\},$$

  junto a su correspondiente valor

$$v_i = \max_{j \in A(i)} \{a_{ij_i} - p_j\}.$$

  y de la misma manera que hacíamos en Naif, seleccionamos el mayor beneficio que reportan otros objetos que no sean j_i. Este es el segundo mejor valor de todos los posibles.

$$w_i = \max_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\}.$$

  Si j_i fuera el único objeto en $A(i)$, definiríamos w_i como -∞ o, en aspectos computacionales, un número que sea mucho menor que v_i.

  Una vez hecho esto, calculamos la puja que realiza la persona i. Aquí es donde entran en juego las modificaciones conseguidas por la ε-HC. La oferta de la persona i viene dada por

$$b_{ij_i} = p_{j_i} + v_i - w_i + \varepsilon = a_{ij_i} - w_i + \varepsilon. \tag{2.6}$$

  Caracterizamos esta situación diciendo que la persona i puja por el objeto j_i y el objeto j_i recibe una oferta de la persona i. El algoritmo funciona si la oferta tiene cualquier valor entre $p_{j_i} + \varepsilon$ y $p_{j_i} + v_i - w_i + \varepsilon$. Cuanto mayor sea la oferta, mayor será la velocidad de convergencia del algoritmo.

- FASE 2: Asignación.
  Para cada objeto j, sea $P(j)$ el conjunto de personas de las que j recibe una oferta en la fase de licitación. Si $P(j)$ es no vacío, se incrementa el precio p_j hasta la mayor oferta.

$$p_j = \max_{i \in P(j)} b_{ij}. \tag{2.7}$$

  Se elimina de la asignación S el par (i, j) en el caso de que j ya hubiera sido asignado a alguna i en una iteración anterior, y se añade a S el par (i_j, j), donde i_j es una persona de $P(j)$ que alcanza el valor definido en (2.7).

Cabe notar que existe cierta libertad al elegir el conjunto de personas pertenecientes a I que pujan en una iteración. Una posibilidad es considerar que I solo está compuesto por una persona no asignada mientras que la otra es considerar que I lo componen todas aquellas personas no asignadas.

Durante cada iteración, los objetos cuyos precios se modifican son los que reciben alguna oferta. Como ya hemos mencionado cuando hemos definido las nuevas condiciones de holgura, cada cambio de precio implica un aumento de al menos ε, ya que tenemos

$$b_{ij_i} = a_{ij_i} - w_i + \varepsilon \geq a_{ij_i} - v_i + \varepsilon = p_{j_i} + \varepsilon.$$

Cabe destacar que si un objeto recibe una puja en m iteraciones del algoritmo, su precio aumentará al menos en una cantidad $m\varepsilon$. Si un objeto recibe una oferta un número infinito de veces, el precio aumentará a $+\infty$. Luego, para un m suficientemente grande, el objeto empezará a volverse "caro" frente a un objeto que no ha recibido todavía ninguna puja hasta el momento y que por tanto, se vuelve más atractivo.

Al final de cualquier iteración tenemos una asignación que difiere de la anterior, porque cada objeto que ha recibido una oferta se asigna a una persona que no estaba asignada a ningún objeto al principio de la iteración. Sin embargo, la asignación final de cada iteración puede tener el mismo número de parejas que la asignación incial, ya que se puede dar el caso de que todos los objetos que reciben una puja, estuvieran ya asignados previamente.

El algoritmo mantiene en todo momento las condiciones ε-HC. Lo vemos en la siguiente proposición.

Proposición 2.1. *El algoritmo de subasta conserva la condición ε-HC a lo largo de toda su ejecución. Esto es, si la asignación y el vector de precios del inicio de la iteración satisfacen ε-HC, también lo harán la asignación y el vector de precios obtenidos al final de la iteración.*

Demostración. Supongamos que el objeto j^* recibe una oferta de la persona i y es asignado a ella durante la iteración. Sean p_j y p'_j los precios del objeto antes y después de la fase de asignación, respectivamente. Tenemos, por las ecuaciones (2.6) y (2.7) que:

$$p'_{j^*} = a_{ij^*} - w_i + \varepsilon.$$

Usando esta ecuación, obtenemos:

$$a_{ij^*} - p'_{j^*} = w_i - \varepsilon = \max_{j \in A(i), j \neq j^*} \{a_{ij} - p_j\} - \varepsilon.$$

Como $p'_j \geq p_j$ para todo j, esta ecuación implica que

$$a_{ij^*} - p'_{j^*} = \max_{j \in A(i)} \{a_{ij} - p'_j\} - \varepsilon,$$

lo que significa que las condiciones ε-HC se siguen cumpliendo después de la fase de asignación para todos los pares (i, j^*) que han participado durante la iteración.

Consideramos ahora cualquier par (i, j^*) que pertenecía a la asignación parcial antes de la iteración, y sigue pertenenciendo una vez finalizada. Entonces j^* no ha recibido ninguna oferta en la fase de licitación, por lo que $p'_{j^*} = p_{j^*}$ y por tanto se mantiene la condición ε-HC que ya se cumplía al inicio de la iteración. Y así, hemos probado que se cumple la condición para todos los pares (i, j^*) que pertenecen a la asignación una vez finalizada la iteración. □

Acabamos de probar que cuando una iteración del algoritmo de subasta termina, se tiene una asignación que satisface las condiciones de holgura (2.4) y, por tanto, al finalizar el proceso algorítmico la solución obtenida también las cumplirá. Sin embargo, no sabemos nada de la optimalidad de la solución. Esta dependerá del tamaño de ε. Asemejando de nuevo el algoritmo a una subasta verdadera, el postor no hará una puja innecesariamente alta por ganar el objeto. Podemos intuir que si ε es pequeño, la solución será buena; es decir, estará cerca de ser óptima.

El siguiente resultado que daremos valida el algoritmo. Para su demostración se deben tener algunas consideraciones previas en cuenta que se explican a continuación.

- Una vez que un objeto es asignado, permanece asignado a lo largo de toda la ejecución del algoritmo. Puede haber una reasignación de dicho objeto a una persona diferente, pero nunca el objeto volverá a quedar desasignado. Además, excepto cuando el algoritmo finaliza, siempre existe al menos un objeto que no ha sido asignado y por tanto tiene un precio igual a su precio inicial.
- Cada $|A(i)|$ ofertas de la persona i, donde $|A(i)|$ es el número de objetos en el conjunto $A(i)$, el escalar $v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}$ decrece al menos en ε. La razón es que una puja realizada por la persona i, o bien decrece v_i por al menos ε o deja v_i sin cambios porque hay más de un objeto alcanzando el máximo. Sin embargo, en este último caso, el precio del objeto j_i que ha recibido la oferta se va a incrementar al menos en ε, y el objeto j_i no va a recibir otra puja por la persona i hasta que v_i decrezca al menos ε. La conclusión es que si una persona i apuesta un número infinito de veces, v_i va a decrecer hasta $-\infty$.

Proposición 2.2. *Si existe al menos una asignación factible, que satisface* (2.4) *junto a un vector de precios, el algoritmo de subasta termina con una asignación que está a $n\varepsilon$ de ser óptima, siendo óptima si los datos del problema son enteros y $\varepsilon < \frac{1}{n}$. Además el vector de precios está a $n\varepsilon$ de ser una solución óptima del problema dual.*

Demostración. Se va a argumentar por contradicción. Si la terminación del algoritmo no ocurre, el subconjunto J^∞ de objetos que recibe un número infinito de pujas y el subconjunto de personas I^∞ que hacen ofertas un número infinito de veces son ambos no vacíos.

Los precios de los objetos en J^∞ van a tender a ∞ y, por las consideraciones previas, los valores v_i de todas las personas $i \in I^\infty$ van a tender a $-\infty$. Este hecho implica que,

$$A(i) \subset J^\infty, \quad \forall i \in I^\infty. \tag{2.8}$$

Las condiciones ε-HC establecen que $a_{ij} - p_j \geq v_i - \varepsilon$ para todo par asignado (i, j); entonces, después de un número finito de iteraciones, cada objeto de J^∞ solo puede ser asignado a una persona de I^∞. Además, después de un número finito de iteraciones, al menos una persona de I^∞ va a ser desasignada al principio de cada iteración, de esto sigue que el número de personas que pertenecen a I^∞ es estrictamente mayor que el número de objetos pertenecientes a J^∞. Este hecho contradice la existencia de una asignación factible, ya que por (2.8), las personas de I^∞ solo pueden ser asignadas a objetos de J^∞. Hemos llegado a contradicción y por tanto, el algoritmo sí termina. Por la proposición 2.1, la asignación final conserva ε-HC.

Vamos a ver ahora que dicha asignación está a $n\varepsilon$ de ser óptima. Sea A^* el óptimo beneficio total,

$$A^* = \max \sum_{i=1}^{n} a_{ik_i}$$

y sea D^* el costo óptimo dual.

$$D^* = \text{mín}\left\{\sum_{i=1}^{n} \max_{j\in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j\right\}.$$

Si $\{(i, j_i) \mid i = 1,\ldots,n\}$ es la asignación final satisfaciendo las condiciones ε-HC junto con un vector de precios $\bar{p}$, tendremos

$$max_{j\in A(i)}\{a_{ij} - \bar{p}_j\} - \varepsilon \leq a_{ij_i} - \bar{p}_{j_i}.$$

Sumando para todos los i,

$$D^* \leq \sum_{i=1}^{n}\left(\max_{j\in A(i)} \{a_{ij} - \bar{p}_j\} + \bar{p}_{j_i}\right) \leq \sum_{i=1}^{n} a_{ij_i} + n\varepsilon \leq A^* + n\varepsilon.$$

Por el Teorema 1.3, sabemos que $A^* = D^*$, así el beneficio total $\sum_{i=1}^{n} a_{ij_i}$ está a $n\varepsilon$ de A^*, mientras que el costo dual está a $n\varepsilon$ de ser el costo dual óptimo. Además, si todos los beneficios a_{ij} son enteros, el beneficio total de cualquier asignación es entero, y si $n\varepsilon < 1$, cualquier asignación completa que está a $n\varepsilon$ de ser óptima es óptima. □

Respecto a la velocidad de convergencia del algoritmo, vamos a estudiar una acotación del número de iteraciones t necesarias para que el algoritmo termine. Asumiendo que los a_{ij} son enteros, escogemos

$$M = \max\{a_{ij} \mid (i, j) \in A\} > 0. \tag{2.9}$$

Asumir además que $\varepsilon = \frac{1}{k}$ para algún entero k. Entonces es claro que, como máximo, serán necesarias kM iteraciones antes de que el precio de cada nodo con oferta alcance o supere el valor M. Como el número de precios no nulos no pueden exceder n, obtenemos la estimación:

$$t \leq kMn. \tag{2.10}$$

Si $\varepsilon = \frac{1}{n+1}$ hemos garantizado en la Proposición 2.2 que la asignación final será óptima (si los datos son enteros), entonces tenemos:

$$t \leq M(n^2 + n). \tag{2.11}$$

Es cierto que esta cota superior del número de iteraciones es válida, pero en muchos de los casos resulta demasiado amplia y el algoritmo termina en un número de pasos que es mucho menor que $M(n^2 + n)$.

Consideramos ahora el caso de un problema de asignación no factible. El algoritmo de subasta no puede terminar, seguirá haciendo incrementos de al menos ε a los precios de algunos objetos. Además, habrá personas presentando ofertas indefinidamente, y los correspondientes valores v_i irán decreciendo a $-\infty$. La forma más sencilla de evitar la infactibilidad es convertir el problema de asignación en un problema "forzadamente" factible antes de iniciar el algoritmo. Se añaden arcos con beneficios extremadamente bajos. Dichos arcos solo habrán sido utilizados en la asignación óptima en el caso de que el problema inicial sin modificaciones sea infactible.

Capítulo 3

Algoritmo de subasta para el problema de flujo a costo mínimo.

En el presente capítulo se va a generalizar la idea del algoritmo general de subasta y se va a aplicar al problema de flujo a costo mínimo presentado en (1.3). Lo primero que debemos hacer es introducir una serie de definiciones y resultados previos que aparecerán a lo largo del desarrollo del algoritmo de subasta.

3.1. Conceptos previos.

Teniendo en cuenta cómo era la forma de las condiciones de holgura complementaria (1.11) para el problema de flujo a costo mínimo, las condiciones ε-HC se pueden generalizar para dicho problema en la forma siguiente. Para un vector factible de flujo x y un vector de precios p:

$$p_i - p_j \leq a_{ij} + \varepsilon \qquad \forall (i,j) \in A \text{ con } x_{ij} < c_{ij}, \tag{3.1a}$$

$$p_i - p_j \geq a_{ij} - \varepsilon \qquad \forall (i,j) \in A \text{ con } 0 < x_{ij}. \tag{3.1b}$$

Nos referiremos a estas dos ecuaciones anteriores como condiciones ε-HCM, vemos una representación de las mismas en la Figura 3.1 Análogamente a lo que ocurría para el problema de asignación con las condiciones ε-HC, las condiciones ε-HCM para un vector de flujo factible x y un vector de precios se mantendrán a lo largo de toda la ejecución del algoritmo.

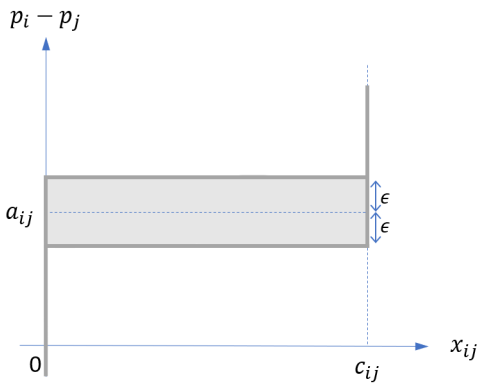

Figura 3.1: Condiciones ε-HCM. Para cada arco (i,j), el par $(x_{ij}, p_i - p_j)$ debe encontrarse en la gráfica, bien sobre las líneas o bien en el área sombreada entre estas.

Estas condiciones resultan de gran importancia algorítmica y dan lugar al siguiente resultado sobre optimalidad del problema.

Proposición 3.1. *Si $\varepsilon < \frac{1}{N}$, donde N es el número de nodos del problema, x es factible, y x y p satisfacen ε-HCM, entonces x es un vector de flujo óptimo para el problema de flujo a costo mínimo*[1].

Demostración. Si x es un vector de flujo no óptimo, por la Proposición A.1 del Apéndice A existe un ciclo simple Y, que tiene costo negativo; es decir,

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} < 0, \tag{3.2}$$

y está desbloqueado con respecto x, por lo que tenemos,

$$x_{ij} < c_{ij}, \forall (i,j) \in Y^+, \qquad 0 < x_{ij}, \forall (i,j) \in Y^-.$$

Por las condiciones (3.1) ε-HCM, las condiciones anteriores implican que

$$\begin{aligned} p_i &\leq p_j + a_{ij} + \varepsilon, \qquad \forall (i,j) \in Y^+, \\ p_j &\leq p_i - a_{ij} + \varepsilon, \qquad \forall (i,j) \in Y^-. \end{aligned}$$

Sumanado estas relaciones sobre todos los arcos de Y, cuyo número no es mayor que N, y por la hipótesis de que $\varepsilon < \frac{1}{N}$, obtenemos

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} \geq -N\varepsilon > -1.$$

Y esto es una contradicción con la ecuación (3.2). Luego x es un vector de flujo óptimo. □

Vamos ahora a definir terminología de operaciones. Cada una de estas definiciones asume que (x,p) es un vector de flujo-precio satisfaciendo ε-HCM.

Definición 2. Para un vector de flujo x, g_i se define como la diferencia del flujo total que entra en i y el flujo total que sale de i.

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

Si esta cantidad es positiva diremos que el nodo i tiene un *exceso* de g_i, mientras que si es negativa, diremos que dicho nodo tiene un *déficit* de g_i.

Definición 3. Un arco (i,j) se denomina ε^+- *desbloqueado* si

$$p_i = p_j + a_{ij} + \varepsilon \quad \text{y} \quad x_{ij} < c_{ij}. \tag{3.3}$$

Un arco (j,i) se denomina ε^-- *desbloqueado* si

$$p_i = p_j - a_{ji} + \varepsilon \quad \text{y} \quad 0 < x_{ji}. \tag{3.4}$$

La *lista "push"* de un nodo i es el conjunto de arcos salientes (i,j) que son ε^+- desbloqueados junto con los arcos entrantes (j,i) que son ε^-- desbloqueados.

En el algoritmo que se va a describir, únicamente se permite un aumento de flujo a través de los arcos ε^+ y una disminución de flujo a través de los arcos ε^-. Las siguientes dos definiciones especifican el tipo de cambios de flujo que se considerarán.

Definición 4. Para un arco (i,j) [ó un arco (j,i)] de la lista "push" del nodo i, sea δ un escalar tal que $0 < \delta \leq c_{ij} - x_{ij}$ ($0 < \delta \leq x_{ji}$, resepectivamente). Un δ-"push" del nodo i en el arco (i,j) [(j,i) respectivamente] consiste en incrementar el flujo x_{ij} en δ (disminuir el flujo x_{ji} en δ, respectivamente), dejando sin cambios todos los demás flujos, así como el vector de precios.

[1] Para realizar la demostración de esta Proposición son necesarios una serie de resultados y definiciones detallados en la Sección A.2 del Apéndice A.

En el contexto del algoritmo de subasta para el problema de asignación un δ-push (con $\delta = 1$) corresponde a asignar una persona no asignada a un objeto, esto supone un aumento del flujo, que pasa de ser 0 a ser 1. La siguiente operación consiste en elevar los precios de un subconjunto de nodos por el máximo común incremento γ que no incumple las condiciones ε-HCM.

Definición 5. Un incremento de precios de un subconjunto no vacío de nodos $I \subseteq N$, es decir $I \neq \emptyset, I \neq N$, consiste en dejar el vector de flujo x y los precios de los nodos no pertenecientes a I sin cambios e incrementar los precios de los nodos de I en la cantidad γ, dada por

$$\gamma = \begin{cases} \text{mín}\{S^+, S^-\}, & \text{si } S^+ \cup S^- \neq \emptyset, \\ 0, & \text{si } S^+ \cup S^- = \emptyset, \end{cases}$$

donde S^+ y S^- son el conjunto de escalares dados por

$$S^+ = \{p_j + a_{ij} + \varepsilon - p_i \mid (i,j) \in A \text{ tal que } i \in I, j \notin I, x_{ij} < c_{ij}\}, \tag{3.5a}$$
$$S^- = \{p_j - a_{ji} + \varepsilon - p_i \mid (j,i) \in A \text{ tal que } i \in I, j \notin I, 0 < x_{ij}\}. \tag{3.5b}$$

En el caso de que el incremento de precio $\gamma = 0$, diremos que el incremento de precio es *trivial*. Un incremento de precio trivial no cambia nada, simplemente se introduce para facilitar el desarrollo del algoritmo. Sin embargo, si el incremento es positivo, lo llamaremos *propio*.

Notar que S^+ y S^- siempre son escalares no negativos por las condiciones ε-HCM. Así, $\gamma \geq 0$, por tanto trabajamos siempre con aumentos de precio.

En el caso de que el subconjunto I esté formado por un solo nodo i, el incremento se llama *incremento de precio del nodo i*. Un aumento de precio de un solo nodo i es propio si y solo si el conjunto $S^+ \cup S^-$ es no vacío pero la lista "push" del nodo es vacía.

El algoritmo genérico de subasta consiste en una secuencia de δ-"push" y operaciones de incremento de precio. La siguiente proposición enumera algunas propiedades de estas operaciones que son importantes en el desarrollo del algoritmo.

Proposición 3.2. *Sea (x,p) un vector flujo-precio satisfaciendo las condiciones ε-HCM.*

(a) El vector flujo-precio obtenido después de un δ-"push" o un incremento de precio satisface las condiciones ε-HCM.

(b) Sea I un subconjunto de nodos tal que $\sum_{i \in I} g_i > 0$. Entonces si el conjunto de escalares $S^+ = \emptyset$ y $S^- = \emptyset$, el problema no es factible.

Demostración. (a) Por las condiciones ε-HCM, el flujo de los arcos ε^+- desbloqueados y ε^-- desbloqueados puede tomar cualquier valor que cumpla la restricción de capacidad de flujo. Como un δ-"push" solo cambia el flujo de un arco ε^+- desbloqueado ó ε^-- desbloqueado, no se incumplen las condiciones ε-HCM.

Sean p y p' los vectores de precio antes y después del incremento de precios de un conjunto de nodos I, respectivamente. Para los arcos (i,j) con $i \in I$ y $j \in I$ o con $i \notin I$ y $j \notin I$, las condiciones ε-HCM se satisfacen para (x,p'), ya que se satisfacen para (x,p) y tenemos $p_i - p_j = p'_i - p'_j$. Para arcos (i,j) con $i \in I$, $j \notin I$ y $x_{ij} < c_{ij}$ tenemos, por las ecuaciones (3.5),

$$p'_i - p'_j = p_i - p_j + \gamma \leq p_i - p_j + (p_j + a_{ij} + \varepsilon - p_i) = a_{ij} + \varepsilon, \tag{3.6}$$

luego se satisface la condición de ε-HCM (3.1a). Para arcos (j,i) con $i \in I$ y $j \notin I$ y $x_{ji} > 0$ se satisface la condición de ε-HCM (3.1b), se prueba de manera análoga al caso anterior.

(b) Si $S^+ \cup S^-$ es vacío

$$x_{ij} = c_{ij}, \qquad \forall (i,j) \in A \text{ con } i \in I, j \notin I, \tag{3.7a}$$
$$x_{ji} = 0, \qquad \forall (j,i) \in A \text{ con } i \in I, j \notin I. \tag{3.7b}$$

Tenemos

$$0 < \sum_{i \in I} s_i = \sum_{i \in I} s_i - \sum_{\{(i,j) \in A | i \in I\, j \notin I\}} x_{ij} + \sum_{\{(j,i) \in A | i \in I\, j \notin I\}} x_{ji}. \tag{3.8}$$

y, combinando esta ecuación junto con (3.7), obtenemos

$$0 < \sum_{i \in I} s_i - \sum_{\{(i,j) \in A | i \in I\, j \notin I\}} c_{ij}.$$

Entonces, para cualquier vector factible, la relación anterior implica que el suministro de los nodos de I excede la capacidad del corte [I,N-I],lo que supone una contradicción[2]. □

Como consecuencia de esta proposición, podemos afirmar que el algoritmo preserva las condiciones ε-HCM durante toda su ejecución.

3.2. Algoritmo general.

Supongamos ahora que el problema lineal de flujo a costo mínimo es factible, y consideramos que el par (x, p) satisface las condiciones ε-HCM. Supongamos que para un nodo i tenemos $g_i > 0$. Existen dos posibilidades:

(a) La lista "push" de i es no vacía; en este caso es posible realizar un δ-"push" del nodo i.

(b) La lista "push" de i es vacía, en este caso el conjunto $S^+ \cup S^-$ correspondiente al conjunto $I = \{i\}$ es no vacío; por tanto el problema es factible por la proposicion anterior. Entonces, el aumento de precio del nodo i será propio.

Por tanto, si $g_i > 0$ para algún i y el problema es factible, bien un δ-"push" o un incremento de precio propio será posible en el nodo i. Además, después de un aumento de precio propio, la lista "push" de i dejará de estar vacía y por tanto, siempre que $g_i > 0$, será posible realizar un δ-"push".

Con estas observaciones previas podemos presentar un algoritmo, llamado algoritmo general, que comienza con un par (x, p) que satisface las condiciones ε-HCM y, como hemos mencionado anteriormente, genera una secuencia de δ-"pushes" y aumentos propios de precio. El algoritmo mantiene durante todo el tiempo un valor fijo positivo de ε y termina cuando $g_i \leq 0$ para todos los nodos i.

3.2.1. Descripción del algoritmo.

Existen muchas formas de implementar el algoritmo. De forma general podríamos definir una iteración del algoritmo como el proceso que cumple los siguientes items:

- Se define un conjunto $I \subseteq N$ con $g_i > 0$.

- Realiza secuencialmente un número finito de aumentos de flujo δ-"pushes" y aumentos propios de precio. En todas las iteraciones del algoritmo debe haber al menos un δ-"push". Sin embargo no es obligatorio un aumento de precios.

[2]Los teoremas de factibilidad de flujo en redes se estudiaron en la asignatura "Grafos y combinatoria" del primer curso del grado de matemáticas [2].

- Los aumentos de flujo se realizan sobre nodos i que poseen exceso. Es decir, dichos nodos cumplen que $g_i > 0$, y además tienen arcos ε^+ ó ε^-- desbloqueados en su lista "push", dónde se pueden realizar esos incrementos o disminuciones de flujo, respectivamente.

- Los aumentos de precio se darán sobre conjuntos I de nodos con $g_i \geq 0 \; \forall i \in I$ pero que tengan sus listas "push" vacías. Estos aumentos provocan que aparezca al menos un arco desbloqueado en la lista "push" del nodo, haciendo así posible un aumento en el flujo.

Proposición 3.3. *Asumimos que nos encontramos ante un problema de flujo a costo mínimo factible. Si el incremento δ de cada δ-push es entero, entonces el algoritmo general termina con un par (x,p) satisfaciendo las condicones ε-HCM. El vector de flujo x es factible, y es óptimo si $\varepsilon < \frac{1}{N}$.*

Demostración. Para el desarrollo de la demostración, debemos tener en cuenta las siguientes consideraciones previas

(a) Los precios de todos los nodos son no decrecientes durante todo el proceso algorítmico.

(b) Una vez que un nodo tiene un exceso no negativo, este permanece no negativo a partir de ese momento. La razon es que un δ-"push" en un nodo i no puede llevar al exceso de i por debajo de cero (ya que $\delta \leq g_i$), y no puede disminuir el exceso de nodos vecinos.

(c) Si en algún momento un nodo tiene déficit, su precio nunca debería haber aumentado hasta ese momento, debiendo ser igual a su precio incial. Esto es consecuencia de la observación anterior y de que solo los nodos con excedentes no negativos pueden verse involucrados en el aumento de precios.

Se va a argumentar por contradicción. Supongamos que el algoritmo no termina. Entonces, como en cada iteración se da al menos un δ-"push", se darían un número infinito de δ-"pushes" en algún nodo i y arco (i,j).

Dado que para cada δ-"push", δ es un número entero, se darán tabíen un número infitnito de δ-"pushes" en el nodo j y arco (i,j). Esto significa que el arco (i,j) será, alternativamente, ε^+- desbloqueado con $g_i > 0$ y ε^-- desbloqueado con $g_j > 0$, lo que implica que p_i y p_j se incrementan en cantidades de al menos 2ε un número infinito de veces. Así, tendremos que $p_i \to \infty$ y $p_j \to \infty$ mientras que $g_i > 0$ ó $g_j > 0$ al comienzo de un número infinito de δ-"pushes".

Sea N^∞ el conjunto de nodos cuyos precios ascienden a ∞. Para preservar las condiciones ε-HCM, debemos tener, despúes de un número suficiente de iteraciones,

$$x_{ij} = c_{ij} \qquad \forall (i,j) \in A \text{ con } i \in N^\infty, j \notin N^\infty, \tag{3.9a}$$

$$x_{ji} = 0 \qquad \forall (j,i) \in A \text{ con } i \in N^\infty, j \notin N^\infty. \tag{3.9b}$$

Después de alguna iteración, por la última consideración, cada nodo de N^∞ debe tener exceso no negativo, ya que la suma de excesos de los nodos de N^∞ debe ser positiva en el inicio de los δ-"pushes" donde o $g_i > 0$ o $g_j > 0$. Se sigue usando el argumento de la demostración de la Proposición 3.2:

$$0 < \sum_{i \in N^\infty} s_i - \sum_{\{(i,j) \in A | i \in N^\infty j \notin N^\infty\}} c_{ij}.$$

Para cualquier vector factible, esta relación implica que la suma de las divergencias de los nodos en N^∞ excede la capacidad del corte $[N^\infty, N - N^\infty]$, lo cual es imposible. Entonces, no hay vector de flujo factible, contradiciendo la hipótesis. Por tanto, el algoritmo sí termina. Como al finalizar tenemos $g_i \leq 0$ para todos los i y el problema se supone factible, se sigue que $g_i = 0$ para todo i. Por ello el vector de flujo final x es factible y satisface las condiciones ε-HCM junto con el vector de precios final. Por la Proposición 3.1, si $\varepsilon < \frac{1}{N}$, x es un vector de flujo óptimo. □

Usando estructuras de datos relativamente simples se puede demostrar que en el peor de los casos, el tiempo de ejecución del algoritmo es de[3] $O\left(N^3 + \frac{N^2 L}{\varepsilon}\right)$, donde L es la máxima longitud de todos los caminos, siendo la longitud de cada arco (i, j) el valor absoluto $\mid p_j + a_{ij} - p_i \mid$, con p el vector inicial de precios. La velocidad de convergencia depende fuertemente tanto del tamaño de L como del de ε.

La técnica[4] de ε-scaling es importante en el contexto del método de ε-relajación, que explicamos más adelante, ya que mejoran el tiempo de ejecución del algoritmo, siendo, si los datos son enteros $O(N^3 \log(NC))$, donde $C = \text{máx}_{(i,j)\in A} \mid a_{ij} \mid$.

Consideramos ahora el caso en el que el problema de flujo a costo mínimo no es factible. Asumimos que el algoritmo genérico opera de manera que en cada δ-"push", δ es un número entero. Se podrán dar tres situaciones de infactibilidad que se detectan de la siguiente forma:

- El algoritmo terminará con $g_i \leq 0$ para todo i y $g_i < 0$ para al menos un i.
- El algoritmo llevará a cabo un número infinito de iteraciones, y consecuentemente, un número infinito de δ-"pushes". En este último caso, por la proposición anterior, los precios de los nodos involucrados en el número infinito de δ-pushes divergerán a infinito.
- Durante el transcurso del algoritmo encontramos un subconjunto de nodos I tales que $\sum_{i\in I} g_i > 0$, y los conjuntos de escalares S^+ y S^- son vacíos.

Sin embargo, no podemos asegurar que encontraremos dichas condiciones en el desarrollo del algoritmo de un problema infactible. Una manera más sencilla para detectar infactibilidad es resolver un problema de máximo flujo con el cual se puede determinar de forma muy rápida la factibilidad o no del problema.

3.3. Implementación con el método de ε-relajación.

El esquema anterior deja mucha libertad a la hora de diseñar una implementación del algoritmo, en esta sección se presenta una versión denominada "ε-Relajación", caso particular de la forma general de iteración presentada en la Sección 3.2.1. Su principal diferencia es que en cada iteración, todos los δ-"pushes" y los aumentos de precio implican a un solo nodo i.

Asumimos que el problema es factible y fijamos un valor positivo de ε. Además, consideramos que tanto los flujos como los suministros y el límite superior de flujo son números enteros. El algoritmo comienza a iterar con un vector flujo-precio (x, p) satisfaciendo las condiciones ε-HCM, descritas en (3.1). Se selecciona un nodo i con exceso $g_i > 0$, y en el caso de que no existan dichos nodos al comienzo de una iteración, el algoritmo finaliza.

3.3.1. Descripción de una iteración del método.

Análogamente a lo que ocurría en el algoritmo general, cada iteración consiste en un número finito de aumentos de flujo, a los que llamábamos δ-"pushes", y un número finito de aumentos de precio propios sobre nodos que poseen excedente ($g_i > 0$).

Al igual que en el algoritmo general, en una iteración se puede dar el caso de que no haya ningún aumento de precio. Sin embargo, siempre habrá al menos un δ-"push". Los aumentos de flujo serán

[3] Esta estimación queda justidicada detalladamente en [6].

[4] No realizaremos la exposición del método de ε-Scaling debido a la limitación de espacio y tiempo. Se puede encontrar un desarrollo completo del método en [7]. Esta técnica consiste en aplicar el algoritmo varias veces, comenzando con un valor de ε grande que se va reduciendo sucesivamente hasta que es inferior a algún valor crítico ($\frac{1}{n}$ cuando los datos son enteros). Únicamente destacamos la notable mejora de la convergencia del método gracias a ε-Scaling, según manifiesta su autor.

siempre enteros (δ entero) y nuncá superarán el exceso del nodo ($\delta \leq g_i$).

Los pasos de una iteración genérica del algoritmo se muestran en el Algoritmo 2.

Algorithm 2: Iteración ε-Relajación.

```
1 G = {i ∈ N | g_i > 0};
  Seleccionar i ∈ G ;
  while lista−"push" ≠ ∅ AND g_i > 0 do
      if existe (i, j) ε+ desbloqueado then
          δ = mín{g_i, c_ij − x_ij};
          x_ij ← x_ij + δ;
          g_i ← g_i − δ;
          g_j ← g_j + δ;
      end
      if existe (j, i) ε- desbloqueado then
          δ = mín{g_i, x_ji};
          x_ji ← x_ji − δ;
          g_i ← g_i − δ;
          g_j ← g_j + δ;
      end
  end
  γ = ∞;
  for (i, j) ∈ A do
      if (p_j − p_i + a_ij + ε < γ) AND (x_ij > c_ij) then γ = mín{γ, p_j − p_i + a_ij + ε} ;
  end
  for (j, i) ∈ A do
      if (p_j − p_i + ε − a_ji < γ) AND (0 < x_ji) then γ = mín{γ, p_j − p_i − a_ij + ε} ;
  end
  p_i = p_i + γ;
  if g_i == 0 then Ir a 1;
```

Cabe notar que tenemos $g_i > 0$ al inicio de la iteración y $g_i = 0$ al final, por lo que como mínimo se dará un δ-"push" en cada iteración.

En cuanto a los aumentos de precios, se observa que el paso 3 puede alcanzarse en dos situaciones:

(a) La lista push de i está vacía y $g_i > 0$, en cuyo caso el aumento de precio será propio, ya que hemos asumido que el problema es factible y por la Proposición 3.2, $S^+ \cup S^- \neq \emptyset$. Tras el aumento, la iteración volverá al primer bucle y la lista 'push" del nodo i tendrá al menos un nuevo arco a.

(b) $g_i = 0$. En este caso la iteración finalizará después de un aumento, quizás trivial.

Por lo tanto, todas las aumentos de precios se dan en nodos con exceso como en el algoritmo genérico. Como después de cada aumento propio de precio se realiza al menos un δ-'push", se sigue que el número de aumentos de precio por iteración también es finito.

De las observaciones anteriores se desprende que, si el problema es factible, el método de relajación es un caso especial del algoritmo genérico y satisface también la Proposición 3.3. Por lo tanto, debe terminar con un vector factible de flujo, que es óptimo si $\varepsilon < \frac{1}{N}$. En el Apéndice B se muestra la resolución, realizada manualmente, de un sencillo problema de flujo a costo mínimo.

Capítulo 4

Estudio computacional.

El algoritmo de ε- Relajación presentado en el capítulo anterior ha sido implementado en Fortran y se proporciona en el Apéndice C. En este capítulo vamos a estudiar su comportamiento resolviendo una colección de 434 problemas creados con un generador de redes que aparece también en dicho apéndice. Se han creado tres configuraciones de redes con el objetivo de estudiar el comportamiento del algoritmo al incrementar el número de arcos, al incrementar la dispersión de la oferta y también cuando se consideran redes con arcos de mayor o menor capacidad. Las características de cada configuración se muestran en la siguiente tabla. Por cada fila de la tabla se han generado 31 problemas con esas características.

Config.	Nodos	Arcos	Of.y Dem.	Capacidad
1.1	50	600	10	U(600, 1200)
	50	1000	10	U(1000, 2000)
	50	1500	10	U(1500,3000)
1.2	100	2000	20	U(2000, 4000)
	100	3400	20	U(3400, 6800)
	100	5400	20	U(5400,10800)
1.3	500	4500	100	U(4500,9000)
	500	7000	100	U(7000,14000)
	500	9500	100	U(9500,19000)
2	500	7000	25	U(7000,14000)
	500	7000	100	U(7000,14000)
	500	7000	175	U(7000,14000)
3	500	7000	100	U(0,14000)
	500	7000	100	U(14000,28000)

Observar que en las configuraciones 1.1, 1.2 y 1.3, solo varía el número de arcos para cada número de nodos fijo (todo lo demás viene fijado por el número de arcos), en la cofiguración 2 solo varía la oferta y la demanda, y en la configuración 3 solo varía la capacidad de los arcos.

A continuación, comentamos brevemente los resultados obtenidos para cada configuración. Se han construido diagramas de caja asociando el tiempo de ejecución y agrupando los datos en función de la característica estudiada: número de arcos, número de ofertas o capacidad de arco.

4.1. Configuración 1.

El propósito de esta configuración es estudiar el comportamiento por la influencia del número de arcos. Hemos construido diagramas de caja para estudiar el comportamiento del tiempo de ejecución en función de si la red es más o menos densa.

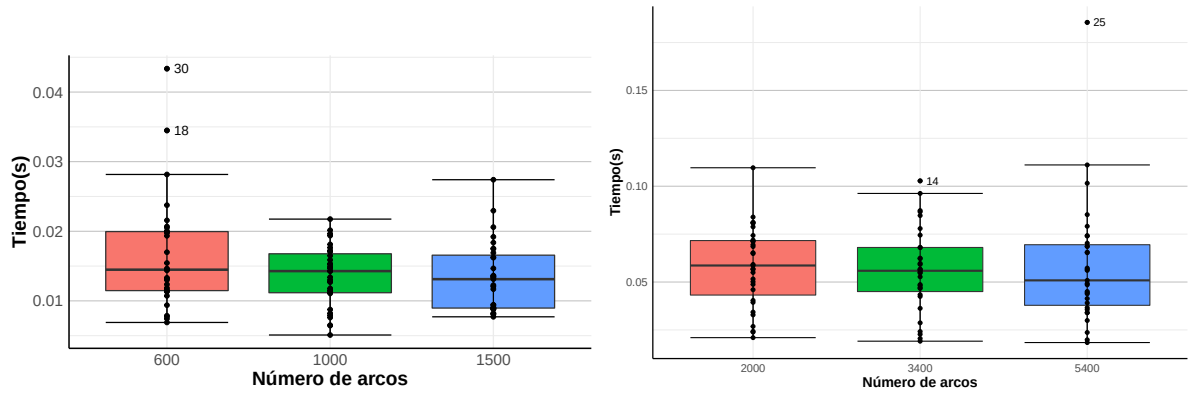

Figura 4.1: Tiempo de ejecución en segundos en función del número de arcos en problemas de 50 nodos (Izda.) y 100 nodos (Dcha.)

Observando el valor de la mediana y la situación de las cajas para el caso de 50 nodos, Figura 4.1 (Izda), se aprecia una pequeña disminución del tiempo cuando se incrementa el número de arcos. Además haciendo el Test Anova en función del número de arcos, obtenemos un p-value=0.082, y por tanto no hay diferencias significativas entre los tiempos medios de ejecución ($\alpha = 0{,}05$.).

Algo similar ocurre en el caso de 100 nodos, lo vemos en la Figura 4.1 (Dcha.), y el valor del p-value en el test Anova es de 0.947. Todavía no se puede afirmar diferencia significativa en las medias.

Finalmente, como se intuía en el diagrama de caja de la Figura 4.2, al considerar la red de 500 nodos aparecen diferencias estadísticamente significativas (p-value = 3.35e-16). En este caso se obtiene que las redes de 7000 y 9500 arcos presentan menor tiempo de ejecución que las redes de 4500 arcos.

Este comportamiento es esperable debido a que al incrementar el número de arcos, las listas-"push" de los arcos son más grandes con lo que el bucle de aumentos de flujo tiende a finalizar con nodos con exceso 0. Por lo tanto no es necesario realizar sucesivas actualizaciones de aumento de precio para generar nuevos arcos con los que disminuir el exceso de los nodos. En el caso de redes pequeñas este efecto no se detecta debido a su pequeño tamaño. Pero cuando las redes aumentan su dimensión, estas diferencias llegan a ser significativas, como se ve en el ejemplo de 500 nodos.

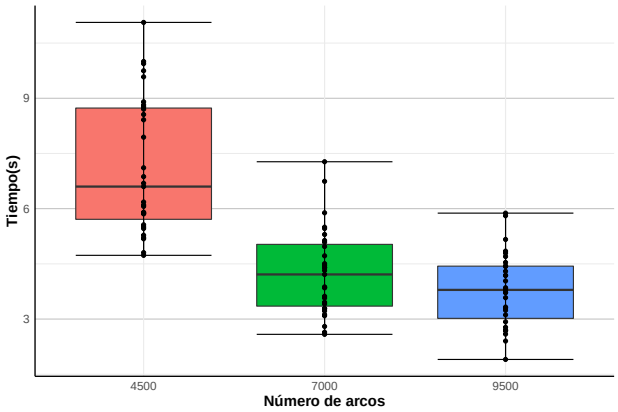

Figura 4.2: Tiempo de ejecución en segundos en función del número de arcos en problemas de 500 nodos.

4.2. Configuración 2.

El objetivo de esta configuración, en la que se varían el número de ofertas y demandas, es estudiar qué ocurre cuando la oferta y la demanda está más o menos distribuida. En este caso en los diagramas de cajas se muestra el tiempo de ejecución en función del número de ofertas y demandas.

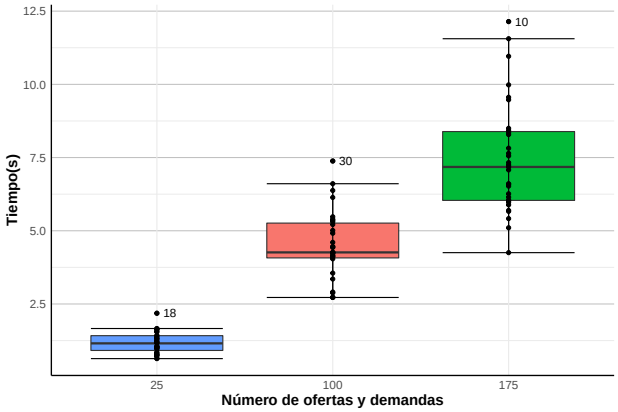

Figura 4.3: Tiempo de ejecución en segundos en función del número de ofertas y demandas del problema.

Observando el diagrama de cajas 4.3, parece claro que el tiempo aumenta cuanto mayor es el número de ofertas y demandas, y así lo confirma el test Anova con un p-value de 2e-16. Este comportamiento es lógico ya que cuando la oferta está inicialmente más concentrada, tenemos un menor número de nodos con oferta, y dado que la oferta total es constante, se produce una mayor saturación de los arcos cercanos a los nodos con oferta y esto obliga a realizar más aumentos de precio para conseguir nuevos arcos desbloqueados y así redistribuir esa oferta.

4.3. Configuración 3.

El objetivo aquí es ver cómo se comporta la red cuando aumentamos la capacidad máxima, pasamos de capacidades uniformes en (0,14000) a (14000,28000) .

En el diagrama de cajas se observan que las redes con arcos de mayor capacidad presentan menores tiempos de ejecución (pvalue= 1.04e-09), lo cual tiene sentido ya que al disponer de arcos con mayor capacidad, los nodos con exceso se vacían más rapido porque los altos desbloqueados tiene mayores capacidades de envío.

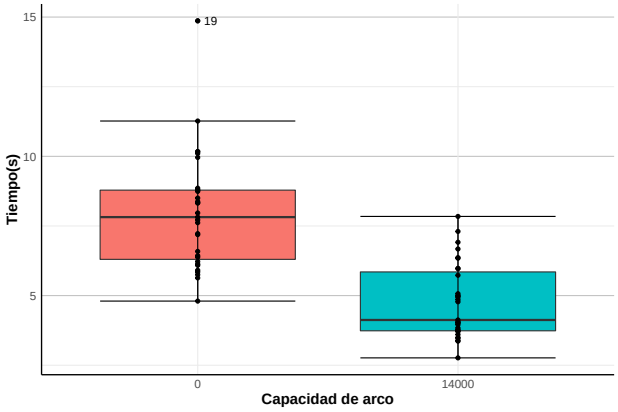

Figura 4.4: Variación del tiempo en función de la capacidad de los arcos.

Bibliografía

[1] PEDRO M. MATEO, *Apuntes de la asignatura Investigación operativa*, Universidad de Zaragoza, 2019.

[2] ALFREDO GARCÍA OLAVERRI, *Teoría de Grafos*, Universidad de Zaragoza, 2018.

[3] DIMITRI P. BERTSEKAS, *Linear Nerwork Optimization*, Massachusetts Institute of Technology, 1991.

[4] DIMITRI P. BERTSEKAS, *A distributed algorithm for the assignment problem*, Laboratory for Information and Decision Systems Working Paper, Massachusetts Institute of Technology, 1979.

[5] DIMITRI P. BERTSEKAS, *Auction Algorithms for Network Flow Problems: A Tutorial introduction*, Computational optimization and applications, 1992, vol. 1, no 1, p. 7-66.

[6] DIMITRI P. BERTSEKAS, *Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems.*, Laboratory for Information and Decision Systems Report P-1606, M.I.T.,1986.

[7] DIMITRI P. BERTSEKAS, J.ECKSTEIN, *Dual Coordinate Step Methods for Linear Network Flow Problems.*, Dual Coordinate Step Methods for Linear Network Flow Problems, Math. Programming, Series B, Vol. 42, pp. 203-243, 1988.

[8] DIMACSFORMAT `http://dimacs.rutgers.edu/programs/challenge/`, accedido Septiembre 2022.

Apéndice A

Demostraciones.

A.1. Demostración Teorema 1.3.

Si una asignación factible y un conjunto de precios $p_j, j = 1,\dots,n$ para el problema (1.1) *satisfacen las condiciones de holgura complementaria* (1.8) *para todas las personas i, entonces la asignación es óptima y dichos precios $p_j, j = 1,\dots,n$ son una solución óptima del problema dual. Además el beneficio de la asignación óptima y el costo dual óptimo son iguales.*

Demostración. El coste total de cualquier asignación factible $\{(i,k_i) \mid i = 1,\dots,n\}$ satisface[1]

$$\sum_{i=1}^{n} a_{ik_i} \leq \sum_{i=1}^{n} \max_{j\in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j, \tag{A.1}$$

para cualquier conjunto de precios $\{p_j \mid i = 1,\dots,n\}$. El primer término del lado derecho de la ecuación anterior no es menor que

$$\sum_{i=1}^{n} (a_{ik_i} - p_{k_i}),$$

mientras que el segundo término es igual a $\sum_{i=1}^{n} p_{k_i}$. Por otro lado, la asignación dada junto con el conjunto de precios, denotados como $\{(i,j_i) \mid i = 1,\dots,n\}$ y $\{\bar{p}_j \mid j = 1,\dots,n\}$ respectivamente, satisfacen las condiciciones de holgura complementaria (1.8); por tanto tenemos

$$a_{ij_i} - \bar{p}_{j_i} = \max_{j\in A(i)} \{a_{ij} - \bar{p}_j\} \qquad i = 1,\dots,n.$$

Al sumar esta relación sobre todos los i, vemos que

$$\sum_{i=1}^{n} a_{ij_i} = \sum_{i=1}^{n} \left(\max_{j\in A(i)} \{a_{ij} - \bar{p}_j\} + p_{j_i} \right).$$

Entonces, la asignación $\{(i,j_i) \mid i = 1,\dots,n\}$ alcanza el máximo del lado izquierdo de la ecuación (A.1) y es óptima para el problema primal (1.1), mientras que los precios $\{\bar{p}_j \mid j = 1,\dots,n\}$ alcanzan el mínimo del lado derecho y son óptimos para el problema dual (1.9). Además, estos valores son iguales.

□

[1] Esta afirmación se debe al Toerema de la dualidad débil, el cual ha sido estudiado en la asignatura del grado "Investigación Operativa" y podemos encontrar en [1]

A.2. Proposición 3.1.

A continuación, mostramos una serie de definiciones y resultados necesarios para la comprensión de la demostración de la Proposición 3.1.

Definición 6. Un *camino C* de un grafo dirigido es una secuencia de nodos $(n_1, n_2, \ldots, n_k)$ con $k \geq 2$ y una correspondiente secuencia de $k-1$ arcos tales que el i-ésimo arco puede ser o bien (n_i, n_{i+1}), en cuyo caso diremos que es un *arco hacia delante*, o bien (n_{i+1}, n_i), en cuyo caso diremos que es *un arco hacia atrás.*

El conjunto de arcos hacia delante se denotan por C^+, mientras que el conjunto de arcos hacia atrás se denotan por C^- .

Definición 7. Un *ciclo simple* es un camino en el que no aparecen ejes repetidos. Además el nodo inicial del camino coincide con el final, siendo este el único nodo del camino que aparece repetido.

Definición 8. El *costo de un ciclo simple C* es la suma de los costos de los arcos hacia delante, menos la suma de los costos de los arcos hacia detrás de C. Esto es,

$$\sum_{(i,j)\in C^+} a_{ij} - \sum_{(i,j)\in C^-} a_{ij}.$$

Definición 9. Decimos que un camino C está *desbloqueado* con respecto a x si $x_{ij} < c_{ij}$ para todos los arcos $(i,j) \in C^+$ y $0 < x_{ij}$ para todos los arcos $(i,j) \in C^-$.

Proposición A.1. *Considerar el problema de flujo a costo mínimo. Sea x un vector de flujo factible que no es óptimo. Entonces, existe un ciclo de flujo simple que sumado a x, produce un vector de flujo factible con menor costo que x, el correspondiente ciclo está desbloqueado con respecto a x y tiene costo negativo. (Queda demostrado en [3], pag 24)*

Apéndice B

Ejemplo de resolución de un problema mediante el Algoritmo ε-Relajación.

A continuación, procedemos a resolver un problema sencillo de flujo a costo mínimo mediante el método de ε-relajación, que ha sido creado "forzadamente" factible, enlazando las ofertas y las demandas con arcos de costo muy superior al costo de todas las demás conexiones.

El problema de flujo que vamos a resolver consta de cinco nodos, dos de ellos son nodos con oferta y otros dos de ellos son demadas. El grafo queda representado tal y como vemos en la Figura B.1.

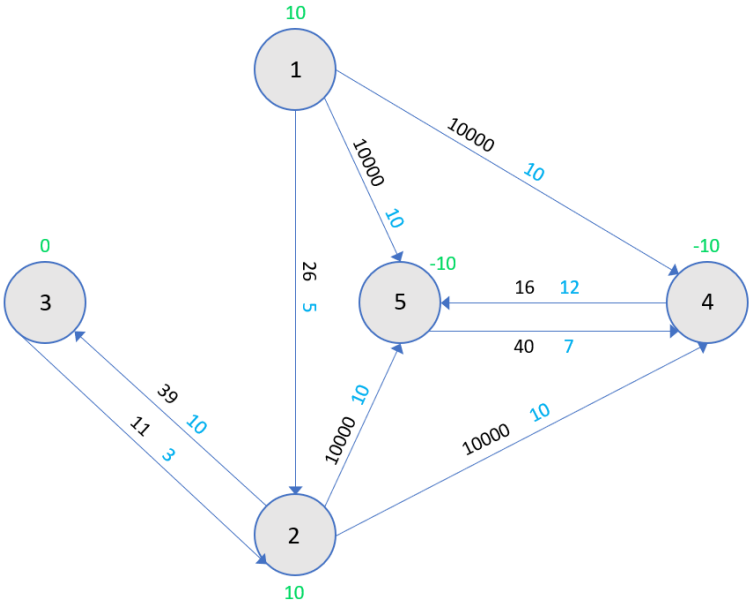

Figura B.1: Grafo dirigido $(5,9)$. Sobre cada arco (i,j): la cifra en negro representa el costo a_{ij} por unidad de flujo x_{ij}; la cifra en azul la capacidad superior c_{ij}. Sobre los nodos i, la cifra en verde representa el exceso ó deficit g_i de cada i.

Antes de comenzar con las iteraciones que nos llevarán a la resolución, debemos inicializar el proceso algorítmico con un vector flujo-precio (x,p) que satisfaga las condiciones ε-HCM. Fijamos un valor de ε que nos asegure la optimalidad de la solución, teniendo en cuenta el resultado de la Proposición 3.1, escogemos $\varepsilon = \frac{1}{N+1} = \frac{1}{6}$.

Trivialmente el vector de precios nulo $\vec{p} = 0$, junto con el vector de flujo nulo $\vec{x} = 0$ satisfacen las condiciones ε-HCM. Así pues, procedemos a realizar la primera iteración del algoritmo.

- PRIMERA ITERACIÓN.
  Seleccionamos el nodo 1, que tiene un exceso de $g_1 = 10$. Buscamos arcos no bloqueados, que pertenezcan a la lista "push" de i. Es trivial comprobar que dado el par flujo-precio inicial, el nodo 1 no tiene arcos en su lista "push".

  Vamos a aumentarle el precio. Veamos en qué cantidad calculando el valor mínimo del conjunto $S^+ = \{p_j + a_{ij} + \varepsilon - p_i \text{ tal que }, x_{ij} < c_{ij}\}$. Examinamos los arcos salientes del nodo 1.

$$\begin{aligned}
&\text{Arco (1,2):} \quad p_2 + a_{12} + \frac{1}{6} - p_1 = 0 + 26 + \frac{1}{6} - 0 = \frac{157}{6}.\\
&\text{Arco (1,4):} \quad p_4 + a_{14} + \frac{1}{6} - p_1 = 0 + 10000 + \frac{1}{6} - 0 = \frac{60001}{6}.\\
&\text{Arco (1,5):} \quad p_5 + a_{15} + \frac{1}{6} - p_1 = 0 + 10000 + \frac{1}{6} - 0 = \frac{60001}{6}.
\end{aligned}$$

  Luego $\gamma_1 = \text{mín}\{S^+\} = \text{mín}\{\frac{157}{6}, \frac{60001}{6}\} = \frac{157}{6}$. Aumentamos el precio del nodo i obteniendo,

$$p_1 = p_1 + \gamma_1 = 0 + \frac{157}{6} = \frac{157}{6}.$$

  Tras realizar este aumento de precio en el nodo 1 va a ser posible realizar un δ-"push" ya que vamos a tener el arco $(1,2)$ en la lista "push" del nodo 1 debido a que:

$$p_1 = p_2 + a_{12} + \varepsilon = \frac{157}{6} \quad \text{con} \quad 0 = x_{12} < c_{12} = 5.$$

  El valor de δ es el siguiente,

$$\delta = \text{mín}\{g_1, c_{12} - x_{12}\} = \text{mín}\{10, 5\} = 5.$$

  . Así pues, aumentamos en cinco unidades el valor del flujo del arco $(1,2)$,

$$x_{12} = x_{12} + 5 = 0 + 5 = 5.$$

$$g_1 = g_1 - 5 = 10 - 5 = 5.$$

$$g_2 = g_2 + 5 = 10 + 5 = 15.$$

  Como $g_1 \neq 0$, volvemos a examinar si existen arcos en su lista "push" y como vuelve a no a ver ninguno, aumentamos de nuevo el precio del nodo 1, siguiendo pasos totalmente análogos a los anteriores. Esta vez,

$$\gamma_2 = \text{mín}\{S^+\} = p_4 + a_{14} + \frac{1}{6} - p_1 = p_5 + a_{15} + \frac{1}{6} - p_1 = 0 + 10000 + \frac{1}{6} - \frac{157}{6} = 9974.$$

  Actualizamos el precio del nodo 1,

$$p_1 = p_1 + \gamma_2 = \frac{157}{6} + 9974 = \frac{60001}{6}.$$

  Por el razonamiento ya explicado, tendremos ahora los arcos $(1,4)$ y $(1,5)$ en la lista "push". Seleccionamos el arco $(1,4)$ y aumentamos su flujo realizando un δ-"push", donde δ es el valor,

$$\delta = \text{mín}\{g_1, c_{14} - x_{14}\} = \text{mín}\{5, 16\} = 5.$$

  Esto dará lugar a los siguientes cambios,

$$x_{14} = x_{14} + 5 = 0 + 5 = 5.$$

$$g_1 = g_1 - 5 = 5 - 5 = 0.$$

$$g_4 = g_4 + 5 = -10 + 5 = -5.$$

Como hemos llegado a que $g_1 = 0$ damos por finalizada esta primera iteración, ya que no se pueden realizar más aumentos de precio propios. Representamos en la Figura B.2 los cambios que ha habido tras ella respecto a la Figura B.1, que representaba el estado inicial del grafo.

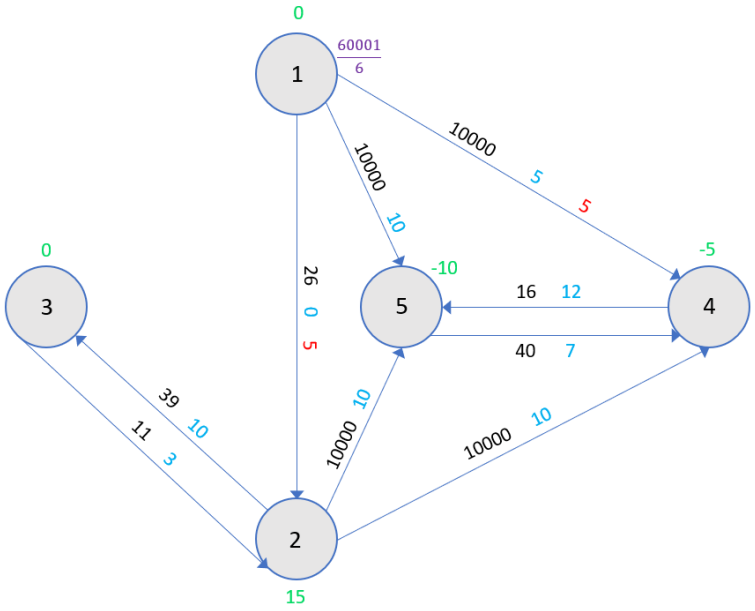

Figura B.2: Grafo dirigido $(5,9)$ tras la primera iteración del algoritmo. Si transcurre flujo por el arco (i,j) aparece la cifra en rojo y la capacidad que le queda en azul. Si el precio del nodo i es superior a 0, aparece la cifra en morado.

- SEGUNDA ITERACIÓN.
  Seguimos un procedimiento análogo a la iteración 1. Seleccionamos el único nodo con exceso que existe en este momento en el grafo, el nodo 2, que tiene un exceso $g_2 = 15$.

  Como no existen arcos en su lista "push" vamos a realizar un incremento de su precio. Calculamos el mínimo del conjunto $S^+ \cup S^- = \{p_j + a_{ij} + \varepsilon - p_i \text{ tal que }, x_{ij} < c_{ij}\} \cup \{p_j - a_{ji} + \varepsilon - p_i \text{ tal que }, 0 < x_{ji}\}$. Examinamos los arcos entrantes y salientes del nodo 2.

$$\text{Arco (2,3):} \quad p_3 + a_{23} + \frac{1}{6} - p_2 = 0 + 39 + \frac{1}{6} - 0 = \frac{235}{6}.$$
$$\text{Arco (2,4):} \quad p_4 + a_{24} + \frac{1}{6} - p_2 = 0 + 10000 + \frac{1}{6} - 0 = \frac{60001}{6}.$$
$$\text{Arco (2,5):} \quad p_5 + a_{25} + \frac{1}{6} - p_2 = 0 + 10000 + \frac{1}{6} - 0 = \frac{60001}{6}.$$
$$\text{Arco (1,2):} \quad p_1 - a_{23} + \frac{1}{6} - p_2 = \frac{60001}{6} - 26 + \frac{1}{6} - 0 = \frac{29923}{3}.$$

Luego $\gamma_1 = \text{mín}\{S^+ \cup S^-\} = \text{mín}\{\frac{235}{6}, \frac{60001}{6}, \frac{29923}{3}\} = \frac{235}{6}$. Aumentamos el precio del nodo 2 obteniendo,

$$p_2 = p_2 + \gamma_1 = 0 + \frac{235}{6} = \frac{235}{6}.$$

Tras realizar este aumento de precio, el nodo 2 va tener el arco $(2,3)$ en su lista "push" debido a que

$$p_2 = p_3 + a_{23} + \varepsilon = \frac{235}{6} \quad \text{con} \quad 0 = x_{23} < c_{23} = 10.$$

El valor de δ es el siguiente,

$$\delta = \text{mín}\{g_2, c_{23} - x_{23}\} = \text{mín}\{15, 10\} = 10.$$

. Así pues, aumentamos en diez unidades el valor del flujo del arco $(2,3)$,

$$x_{23} = x_{23} + 10 = 0 + 10 = 10.$$

$$g_2 = g_2 - 10 = 15 - 10 = 5.$$

$$g_3 = g_3 + 10 = 0 + 10 = 10.$$

Como $g_2 \neq 0$, volvemos a examinar si existen arcos en su lista "push" y como vuelve a no a ver ninguno, aumentamos su precio, siguiendo los mismos pasos.

$$\begin{aligned}
&\text{Arco (2,4):} \quad p_4 + a_{24} + \frac{1}{6} - p_2 = 0 + 10000 + \frac{1}{6} - \frac{235}{6} = 9961. \\
&\text{Arco (2,5):} \quad p_5 + a_{25} + \frac{1}{6} - p_2 = 0 + 10000 + \frac{1}{6} - \frac{235}{6} = 9961. \\
&\text{Arco (1,2):} \quad p_1 - a_{23} + \frac{1}{6} - p_2 = \frac{60001}{6} - 26 + \frac{1}{6} - \frac{235}{6} = \frac{59611}{6}.
\end{aligned}$$

$$\gamma_2 = \text{mín}\{S^+ \cup S^+\} = \text{mín}\{9961, \frac{59611}{6}\} = \frac{59611}{6}.$$

Actualizamos el precio del nodo 2,

$$p_2 = p_2 + \gamma_2 = \frac{235}{6} + \frac{59611}{6} = \frac{29923}{3}.$$

Por el razonamiento ya explicado, tendremos ahora el arco $(1,2)$ en la lista "push", será un arco ε^--desbloqueado y por tanto, realizamos una disminución de flujo δ-"push", donde δ es el valor,

$$\delta = \text{mín}\{g_2, x_{12}\} = \text{mín}\{5, 5\} = 5.$$

Esto dará lugar a los siguientes cambios,

$$x_{12} = x_{12} - 5 = 5 - 5 = 0.$$

$$g_1 = g_1 + 5 = 0 + 5 = 5.$$

$$g_2 = g_2 - 5 = 5 - 5 = 0.$$

Hemos llegado a que $g_2 = 0$. Vamos a ver si existe algún aumento propio de precio y seguidamente pasaremos a la siguiente iteración.

$$\delta_3 = \text{mín}\{S^+\} = p_4 + a_{24} + \frac{1}{6} - p_2 = p_5 + a_{25} + \frac{1}{6} - p_2 = 0 + 10000 + \frac{1}{6} - \frac{29923}{3}.$$

Actualizamos el precio del nodo 2.

$$p_2 = p_2 + \delta_3 = \frac{29923}{3} + 10000 + \frac{1}{6} - \frac{29923}{3} = \frac{60001}{6}.$$

Representamos en la Figura B.3 los cambios que ha habido respecto el final de la primera iteración, representada en la Figura B.2.

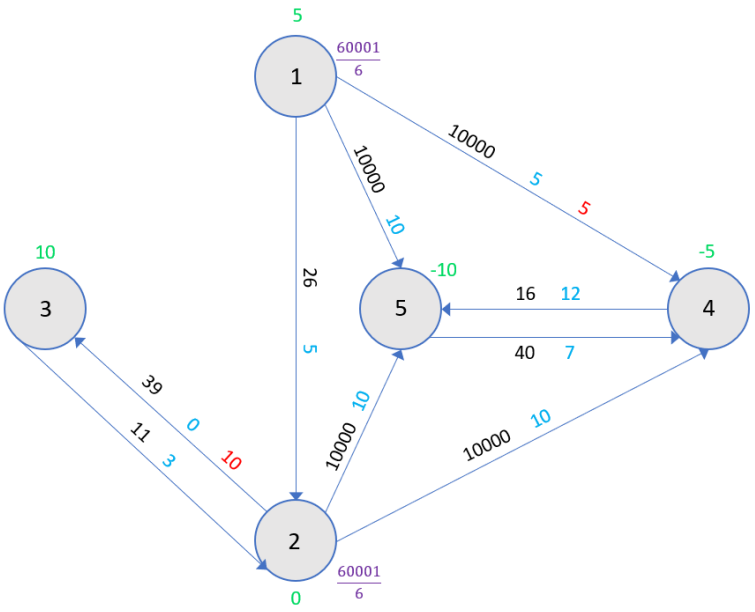

Figura B.3: Grafo dirigido $(5,9)$ tras la segunda iteración del algoritmo.

- TERCERA ITERACIÓN. Elegimos el nodo 1 que tiene un exceso de $g_1 = 5$. Esta vez, al inciar la iteración tenemos dos arcos pertenecientes a la lista "push" del nodo dos. Están el arco $(1,4)$ y el arco $(1,5)$, ya que

$$p_1 = p_4 + a_{14} + \frac{1}{6} = \frac{60001}{6}, \quad 5 = x_{14} > c_{14} = 10,$$
$$p_1 = p_5 + a_{15} + \frac{1}{6} = \frac{60001}{6}, \quad 0 = x_{15} > c_{14} = 10.$$

Seleccionamos el arco $(1,4)$ y realizamos en él un δ-"push" con,

$$\delta = \text{mín}\{g_1, c_{14} - x_{14}\} = \text{mín}\{5, 10-5\} = 5.$$

Aumentamos en cinco unidades el flujo del arco $(1,4)$ y actualizamos los siguientes valores:

$$x_{14} = x_{14} + 5 = 5 + 5 = 10.$$

$$g_1 = g_1 - 5 = 5 - 5 = 0.$$

$$g_4 = g_4 + 5 = -5 + 5 = 0.$$

Como $g_1 = 0$ y no existen más aumentos propios de precio, pasamos a la siguiente iteración. Como en las iteraciones anteriores, mostramos en la Figura B.4 la situación actual del problema.

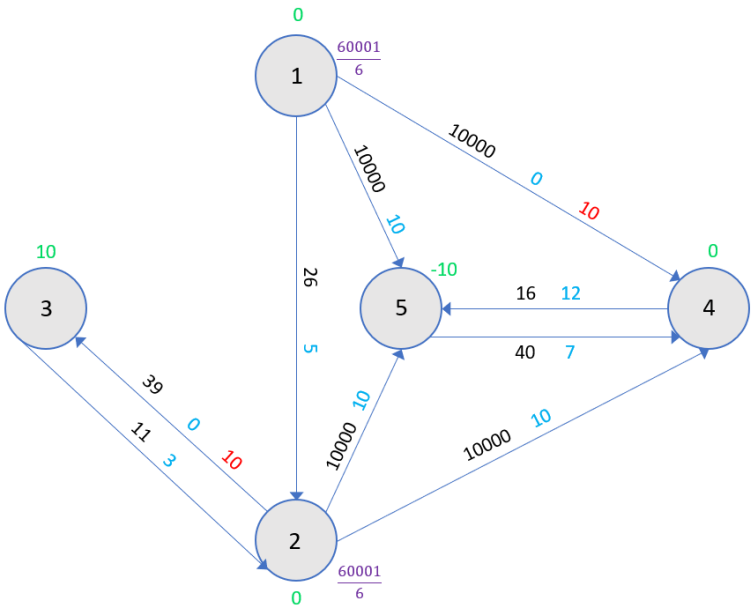

Figura B.4: Grafo dirigido $(5,9)$ tras la tercera iteración del algoritmo.

- CUARTA ITERACIÓN.
  Seleccionamos el único nodo que queda con exceso positivo, $g_3 = 0$. El nodo 3 no tiene elementos en su lista "push" así que vamos a realizar en él un aumento de precio. Para ello calculamos el valor mínimo del conjunto

$$\{S^+ \cup S^-\} = \{p_j + a_{ij} + \varepsilon - p_i \text{ tal que } x_{ij} < c_{ij}\} \cup \{p_j - a_{ji} + \varepsilon - p_i \text{ tal que } 0 < x_{ji}\}.$$

  . Examinamos el arco entrante y el arco saliente del nodo 3:

$$\text{Arco (3,2):} \quad p_2 + a_{32} + \frac{1}{6} - p_3 = \frac{60001}{6} + 11 + \frac{1}{6} - 0 = \frac{30034}{3}.$$
$$\text{Arco (2,3):} \quad p_2 - a_{23} + \frac{1}{6} - p_3 = +\frac{60001}{6} - 39 + \frac{1}{6} - 0 = \frac{29884}{3}.$$

  Luego $\gamma_1 = \text{mín}\{S^+ \cup S^-\} = \text{mín}\{\frac{30034}{3}, \frac{29884}{3}\} = \frac{29884}{3}$. Aumentamos el precio del nodo 3 en esta cantidad:

$$p_3 = p_3 + \gamma_1 = \frac{29884}{3}.$$

  Ahora tenemos el arco $(2,3)$ en la lista-"push" de 3 y realizamos una disminución δ-push del flujo en este arco,

$$\delta = \text{mín}\{g_3, x_{23}\} = \text{mín}\{10, 10\} = 10.$$

  Esto dará lugar a los siguientes cambios,

$$x_{23} = x_{23} - 10 = 10 - 10 = 0.$$

$$g_2 = g_2 + 10 = 0 + 10 = 10.$$

$$g_3 = g_3 - 10 = 10 - 10 = 0.$$

  Hemos llegado a que $g_3 = 0$ y todavía podemos hacer un último aumento del precio del nodo 3,

$$\gamma_2 = S^+ = p_2 + a_{32} + \frac{1}{6} - p_3 = 50.$$

Luego,

$$p_3 = p_3 + \gamma_2 = \frac{29884}{3} + 50 = \frac{30034}{3}.$$

Tras este último aumento finalizamos esta iteración. Así que, representamos los cambios en la Figura B.5 y pasamos a la siguiente

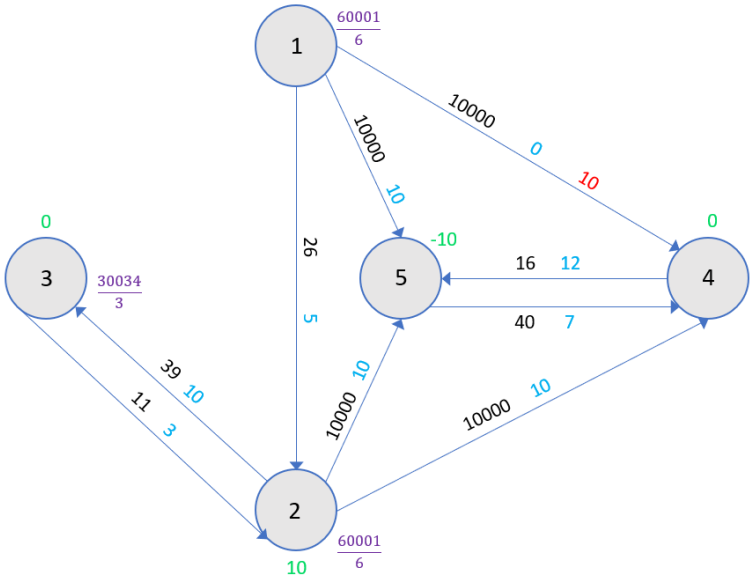

Figura B.5: Grafo dirigido $(5,9)$ tras la cuarta iteración del algoritmo.

- QUINTA ITERACIÓN.
  El único nodo con exceso positivo es el nodo 2, $g_2 = 10$. Este tiene dos arcos desbloqueados en su lista"push", el arco $(2,4)$ y el arco $(2,5)$ ya que

$$p_2 = p_4 + a_{24} + \frac{1}{6} = \frac{60001}{6}, \quad 0 = x_{14} > c_{14} = 10,$$
$$p_2 = p_5 + a_{25} + \frac{1}{6} = \frac{60001}{6}, \quad 0 = x_{15} > c_{24} = 10.$$

Seleccionamos el arco $(2,4)$ y realizamos un aumento de flujo δ-"push". El valor de δ es el siguiente,

$$\delta = \text{mín}\{g_2, c_{24} - x_{24}\} = \text{mín}\,10, 10 = 10$$

Aumentamos el flujo del arco $(2,4)$ en unidades. Se producen los siguientes cambios:

$$x_{24} = x_{24} + 10 = 0 + 10 = 10.$$

$$g_2 = g_2 - 10 = 10 - 10 = 0.$$

$$g_4 = g_4 + 10 = 0 + 10 = 10.$$

Hemos llegado a que $g_2 = 0$ y no se pueden realizar más aumentos propios de precio. Representamos la situación en la Figura B.6 y pasamos a la siguiente iteración

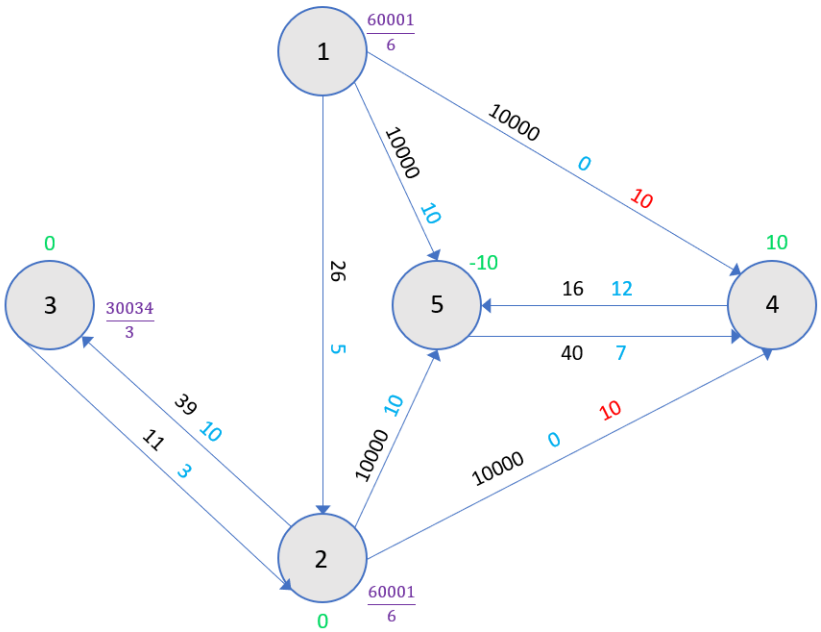

Figura B.6: Grafo dirigido $(5,9)$ tras la quinta iteración del algoritmo.

- SEXTA ITERACIÓN.
  El único nodo con exceso es el nodo 4, que no tiene arcos desbloqueados en su lista "push". Aumetamos el precio del mismo. Calculamos el valor

$$\text{mín}\{S^+ \cup S^-\} = \cdots = \text{mín}\{\frac{97}{6}, \frac{1}{3}\} = \frac{1}{3}$$

  Aumentamos el precio del nodo 4,

$$p_4 = p_4 + \frac{1}{3} = 3.$$

  Realizamos un δ-"push" en el arco $(1,4)$,

$$\text{mín}\{g_4, x_{14}\} = \text{mín}\{10,10\} = 10.$$

  Y actualizamos ,

$$x_{14} = x_{14} - 10 = 10 - 10 = 0.$$

$$g_4 = g_4 - 10 = 10 - 10 = 0.$$

$$g_1 = g_1 + 10 = 0 + 10 = 10.$$

  Hemos llegado a que $g_4 = 0$ y además no se pueden hacer más aumentos de precio propios sobre este nodo 4. Luego actualizamos la representación y pasamos a la siguiente y última iteración.

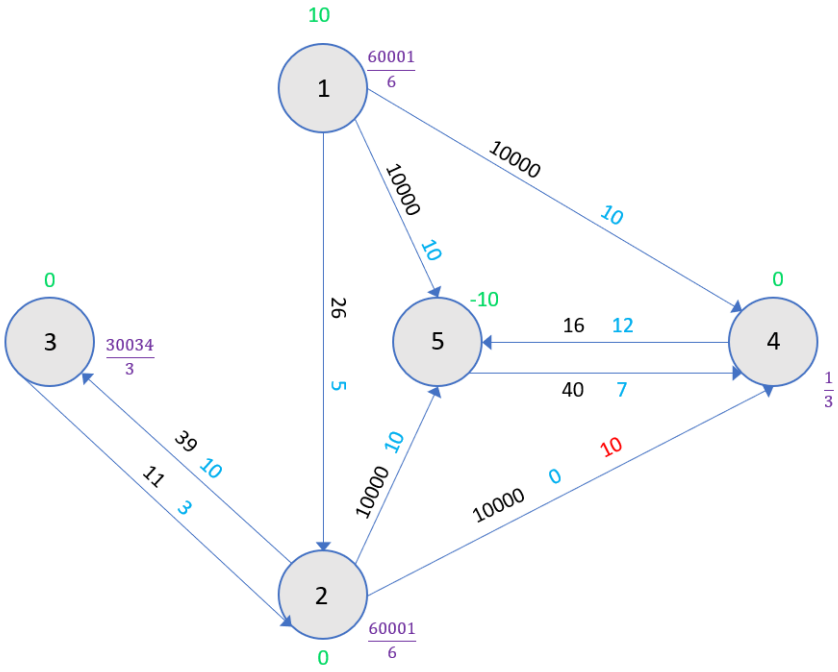

Figura B.7: Grafo dirigido $(5,9)$ tras la sexta iteración el algoritmo.

- SÉPTIMA ITERACIÓN. El único nodo que queda con exceso es el 1, $g_1 = 10$. El arco $(1,2)$ pertenece a la lista "push" de 1, veamos cual es el valor de δ,

$$\delta = \text{mín}\{g_1, c_{ij} - x_{ij}\} = \text{mín}\{10,10\} = 10.$$

Aumentamos el flujo en esta cantidad, y actualizamos los excesos y el flujo,

$$x_{15} = x_{15} + 10 = 0 + 10 = 10.$$

$$g_1 = g_1 - 10 = 10 - 10 = 0.$$

$$g_4 = g_{51} + 10 = -10 + 10 = 0.$$

Hemos llegado a una situacion en la que todos los $g_i = 0$. Luego el proceso algorítmico se da por finalizado. Además la solución es óptima, ya que hemos partido de un vector flujo precio (x,p) factible y un $\varepsilon > \frac{1}{N}$.

La solución del problema de flujo a costo mínimo mediante el algoritmo ε-Relajación queda representada en B.8.

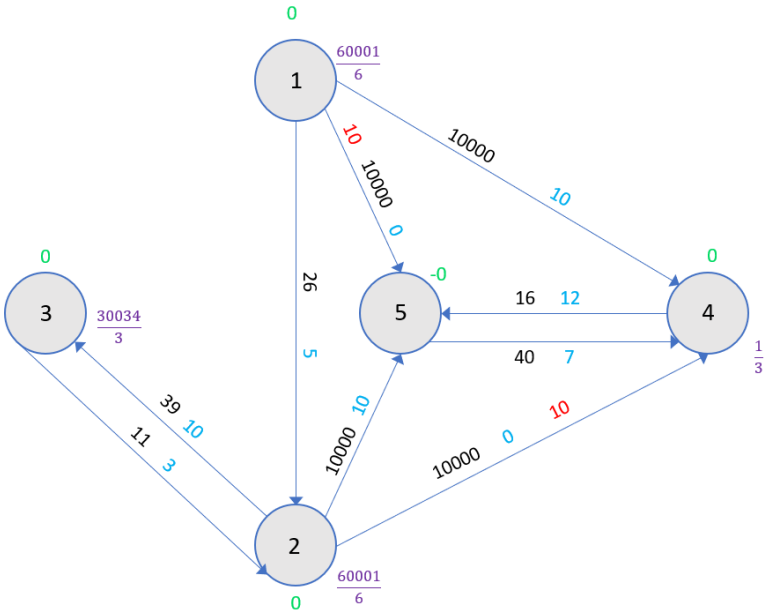

Figura B.8: Representación de la solución óptima.

Sustituyendo los valores de la solución en la función objetivo de nuestro problema obtenemos tenemos el valor

$$Z = a_{15} * x_{14} + a_{24} * x_{24} = 20000.$$

Apéndice C

Códigos.

En esta sección, vamos a ver la implementación del algoritmo ε-Relajación para resolver problemas de flujo a costo mínimo.

Presentamos dos códigos: el primero es un generador de problemas de flujo y el segundo es un optimizador. Este último, primero realiza una estructuración de los datos de los problemas generados, después desarrolla el propio algoritmo ε-relajación y por último almacena los resultados en un fichero de datos.

C.1. Generador de Problemas

Este generador es una simplificación del PGRIDGEN desarrollado en [3]. A continuación realizamos una pequeña explicación de cómo construye la red de arcos y nodos.

Primero crea un esqueleto que asegura que el problema que se va a generar es factible. El número de ofertas y demandas coinciden, así que el esqueleto del algoritmo une un nodo oferta con un nodo demanda. El costo de estos arcos es 1000 veces la cota superior de los arcos que no pertenecen al esqueleto factible (que se ha establecido previamente en 100 unidades) y su capacidad superior es la oferta total del problema.

El resto de los arcos se generan sorteando su nodo inicial i y su nodo final j, comprobando previamente que dichos nodos no son iguales y que no existe ya dicho arco (i, j). El costo y la capacidad se sortean. El costo es un número aleatorio entre 1 y la cota superior de costo (fijada como ya hemos mencionado en el párrafo anterior a valor 100) y la capacidad es un número aleatorio entre c1 y c2, números que el mismo algoritmo pide en su inicio.

Tras generar la información esta es almacenada en un fichero cuyo nombre se proporciona en el generador. Los datos se almacenan en formato DIMACS [8], un formato estándar para el almacenamiento de información de redes.

```
program generador

integer :: c1, c2, n, nof, nd, m, ofertatotal, oferta, demanda, aux
integer, dimension (1000,1000) :: costo !costo por unidad de flujo
integer, dimension(1000,1000):: capsuperior !capacidad superior de flujo
integer :: i, j, seed
character(100) :: cmdArgs
character(100) :: nameFileOut
integer :: cotaCostoArco

!LECTURA DE LINEA DE COMANDOS
```

```
if(command_argument_count().NE.9) then
   write(*,*) 'Argumentos insuficientes (',command_argument_count(), ' en
lugar de 9)'
   write(*,*) 'num. nodos, num. ofertas, num demandas, of. total, cap min,
ccap max, ', &
    'num. arcs, semilla, nombre fichero resultados'
   stop
endif

call get_command_argument(1,cmdArgs)
read(cmdArgs,*) n
write(*,*) 'n=',n
call get_command_argument(2,cmdArgs)
read(cmdArgs,*) nof
write(*,*) 'nof=',nof
call get_command_argument(3,cmdArgs)
read(cmdArgs,*) nd
write(*,*) 'nd=',nd
call get_command_argument(4,cmdArgs)
read(cmdArgs,*) ofertatotal
write(*,*) 'ofertatotal=',ofertatotal
call get_command_argument(5,cmdArgs)
read(cmdArgs,*) c1
write(*,*) 'c1=',c1
call get_command_argument(6,cmdArgs)
read(cmdArgs,*) c2
write(*,*) 'c2=',c2
call get_command_argument(7,cmdArgs)
read(cmdArgs,*) m
write(*,*) 'm=',m
call get_command_argument(8,cmdArgs)
read(cmdArgs,*) seed
write(*,*) 'semilla=',seed
call get_command_argument(9,nameFileOut)

if(nof.NE.nd) then
    write(*,*) 'El generador esta disegnado para el mismo numero de ofertas
 y demandas'
    return
end if

cotaCostoArco=100

!ESTABLECIENDO SEMILLA PARA DIGITOS ALEATORIOS

call srand(seed)
open(unit=10, file=nameFileOut)

write(10,*) ' c Fichero generado por',n,'nodos, ',nof,'son ofertas',nd,'son
demandas, y',ofertatotal,'es la oferta total.'
write(10,*) ' c La factibilidad se asegura con',nd*nof, 'conexiones entre
ofertas y demandas con costo ',m*c2
write(10,*) ' c Adems el problema genera', m-nd*nof,'arcos aleatorios.'
write(10,*) ' c El costo por unidad de flujo enviada por cada arco aleatorio
 oscila entre',c1,'y',c2
write(10,*) ' c En ningn caso existen 2 arcos con el mismo nodo inicial y
el mismo nodo final'
write(10,*) 'p min' , n, m
write(10,*) 'o ',nof
write(10,*) 'k ',c1, c2

oferta=ofertatotal/nof
demanda= ofertatotal/nd
```

```
    if((oferta*nof.NE.ofertatotal).OR.((demanda*nd.NE.ofertatotal)))then
      write(*,*) 'La oferta total debe ser mltiplo del nmero de ofertas y
    demandas'
      return
    end if

    do i=1,nof !escritura de los nodos con oferta
        write(10,*) 'n', i, oferta
    end do

    do i= n-nd+1,n !escritura de los nodos con demanda
        write(10,*) 'n', i, -demanda
    end do

    do i=nof+1, n-nd !escritura de los nodos de transbordo
        write(10,*) 'n', i, 0
    end do

    do i=1,n
     do j=1,n
        capsuperior(i,j)= -1
     end do
    end do

    !CONSTRUCCI N DEL ESQUELETO FACTIBLE
    do i=1,nof   !nodos con oferta
            costo(i,n-i+1)= cotaCostoArco*1000
            capsuperior(i,n-i+1)= ofertatotal   !Asegura la factibilidad
    end do
    numarc=nof
1   do while(numarc< m)
        i=rand()*n + 1
       j=rand()*n + 1
       if((i==j).OR.(capsuperior(i,j).NE.-1))then
        go to 1
       end if
       if  (capsuperior(i,j)==-1) then
        costo(i,j)= rand()*cotaCostoArco
        capsuperior(i,j)=c1+rand()*c2
        numarc=numarc+1
       end if
    end do

      do i=1,n
       do j=1,n
        if ( capsuperior(i,j)/=-1 ) then
            write(10,*) 'a',i,j,costo(i,j), capsuperior(i,j)
        end if
       end do
      end do
    close(unit=10)
end program
```

C.2. Algoritmo ε-Relajación para el problema de flujo a costo mínimo.

```
program lectoryoptimizador
implicit none

integer N,A,j,iN,iA,costo,capacidad,salir,l,iii,i,numOfertas,c1
integer, external :: token
```

```
integer, allocatable  :: startA(:), endA(:),  nextIn(:), nextOut(:), currentIn
    (:), currentOut(:),  capacidadA(:),s(:)
integer (kind=8),allocatable:: costoA(:)
integer::  dpushA
integer (kind=8):: totalCosto, totalFlujo
integer, allocatable ::  flujoA(:)
integer, allocatable ::firstIn(:), firstOut(:)

real(kind=8) , allocatable :: p(:)
real(kind=8)::gamma
real(kind=8) :: eps
real:: t1,t2

character (len=200) :: linea
character (100) :: name
character(100) :: cmdArgs
character(100) :: nameFileOut

call get_command_argument(1,nameFileOut)
    if(command_argument_count().NE.1) then
        write(*,*) 'Argumentos insuficientes (',command_argument_count(), ' en
    lugar de 1)'
        write(*,*) 'Nombre del fichero de datos'
    stop
    end if

open(unit=1, file= nameFileOut, STATUS='OLD')

!LECTURA DEL FICHERO
    do while(5.GT.4) !
        read(1,'(A)',END=17) linea
        if(linea(1:1).ne.'c')then
            call StripSpaces(linea)
        if(linea(1:1)=='p') then
            N=token(linea,3)
            A=token(linea,4)
            allocate (capacidadA(A))
            allocate (costoA(A))
            allocate (startA(A))
            allocate (endA(A))
            allocate  (nextIn(A))
            allocate (nextOut(A))
            allocate (currentIn(A))
            allocate (currentOut(A))
            allocate(flujoA(A))
            allocate (firstIn(N))
            allocate(firstOut(N))
            allocate (S(N))
            allocate(p(N))
            nextIn=-1
            nextOut=-1
            currentIn=-1
            currentOut=-1
            firstIn=-1
            firstOut=-1
            flujoA=0
            S=0
            iA=0
            iN=0
            p=0
        end if
      if(linea(1:1).eq.'o') then
            numOfertas=token(linea,2)
```

```
            end if
      if(linea(1:1).eq.'k') then
            c1=token(linea,2)
            end if
        if(linea(1:1).eq.'n') then
            S(token(linea,2))=token(linea,3)
            iN=iN+1
            end if
        if(linea(1:1).eq.'a') then
            i=token(linea,2)
            j=token(linea,3)
            costo=token(linea,4)
            capacidad=token(linea,5)
            iA=iA+1
            startA(iA)=i
            endA(iA)=j
            costoA(iA)=costo*(n+1)
            capacidadA(iA)=capacidad
            flujoA(iA)=0
            if (firstIn(j)==-1) then
                firstIn(j)=iA
                currentIn(j)=iA
                else
                    nextIn(currentIn(j))=iA
                    currentIn(j)=iA
            end if
            if (firstOut(i)==-1) then
                firstOut(i)=iA
                currentOut(i)=iA
                else
                    nextOut(currentOut(i))=iA
                    currentOut(i)=iA
            end if
           end if
        end if
    end do
17  close(unit=1)
    write(*,*)'Finalizada carga de datos'
!FIN DE LECTURA DE FICHERO
!ALGORITMO EPS-RELAJACION
    call cpu_time(t1)
!Establecemos eps, inicializamos p()
eps=1
    do i=1,n
        p(i)=0.0
    end do

!Inicializacin de flujos
   do ia=1,A
        if(p(startA(ia))-p(endA(ia))-costoA(ia)<= eps)then
          flujoA(ia)=0
        if(p(startA(ia))-p(endA(ia))-costoA(ia)>= -eps) then
           flujoA(ia)=capacidadA(ia)
           s(startA(ia))= s(startA(ia))-capacidadA(ia)
           s(endA(ia))= s(endA(ia))+capacidadA(ia)
        end if
        end if
    end do
!Fin de inicializacion
    do i=1,N
        iA = firstOut(i)
        do while (ia.ne.-1)
            !Write(*,*) startA(iA), endA(iA), costoA(iA), flujoA(iA),capacidadA(
```

```
    iA), p(startA(iA)), p(endA(iA))!, nextout(iA), nextin(iA)
             ia=nextOut(iA)
         end do
      end do
      do i=1,N
       !  write(*,*) 'nodo ',i,' exceso ', s(i)
      end do

     salir=0
     do while (salir.EQ.0)
         i=0
         l=0
         do j=1,n
             if(s(j)>0)then
     !         write(*,*)'exceso de nodo',j,' al comienzo de iter',s(j)
               i=i+s(j)
             end if
             if(s(j)<0)then
      !         write(*,*)'exceso de nodo',j,' al comienzo de iter',s(j)
               l=l+s(j)
             end if
         end do
       !  write(*,*)'exceso total al principio de la iteracion
    ------------------->',i,l
         salir=1
         i=-1

!Buscamos nodo con exceso.

         do j=1,n
            if(s(j).GT.0) then
              i=j
              salir=0
        !      write(*,*)'Nodo con exceso positivo seleccionado',i, 'exceso =',s(
    i)
              go to 10
            end if
          end do
          !Para detener el algoritmo porque no hay nodos con exceso positivo
10       if(i.EQ.-1)then
             salir=1
             cycle
         end if
2          if (salir==0) then !Tenemos un nodo con exceso positivo. buscamos
    componentes de la lista push
                 iA= firstOut(i) !Buscamos arcos no bloqueados (i,j)
                 do while (iA.NE.-1)
                     if ((p(startA(iA))== p(endA(iA))+costoA(iA)+eps).AND.(flujoA
    (iA).LT.capacidadA(iA))) then !e+desbloqueados
                         dpushA=min(s(i), capacidadA(iA)-flujoA(iA)) !aumento el
    flujo del arco en esta cantidad
                         flujoA(iA)=flujoA(iA)+dpushA
                         s(i)=s(i)-dpushA
                         s(endA(iA))=s(endA(iA))+dpushA
                             if(s(i)==0) then
                                   go to 25
                             endif
                     end if
                     iA=nextOut(iA)
                 end do
                 iA= firstIn(i) !'Buscando arcos no bloqueados (j,i)'
                 do while (iA.NE.-1)
                     if ((p(endA(iA))== p(startA(iA))-costoA(iA)+eps).AND.(0.LT.
```

```
    flujoA(iA))) then  !e -desbloqueados
                         dpushA=min(s(i), flujoA(iA))
                         flujoA(iA)=flujoA(iA)-dpushA
                         s(i)=s(i)-dpushA
                         s(startA(iA))=s(startA(iA))+dpushA
                             if(s(i).EQ.0) then
                                 go to 25
                             endif
                     end if
                     iA=nextIn(iA)
                 end do

!Actualizacin de precios
 25              Gamma=99999999.0
                 iA  = firstOut(i) !arcos(i,j))
                 do while (iA.NE.-1)
                     if ((p(endA(iA))+costoA(iA)+eps-p(startA(iA)).LT.gamma).AND
    .(flujoA(iA).LT.capacidadA(iA))) then
                          gamma=p(endA(iA))+costoA(iA)+eps-p(startA(iA))
                     end if
                    iA= nextout(iA)
                 end do
                 iA=firstIn(i) !arcos (j,i)
                 do while (iA.NE.-1)
                     !write(*,*) iA
                     if ((p(startA(iA))-costoA(iA)+eps-p(endA(iA)).LT.gamma).AND
    .(0.LT.flujoA(iA))) then  !cual esta bien de los dos??? revisar tb memoria
                           gamma= p(startA(iA))-costoA(iA)+eps-p(endA(iA))
                     end if
                     iA= nextIn(iA)
                 end do
                 p(i)=p(i)+gamma !Actualizamos p
                 !Si hay exceso realizamos d-pushes y si no se sigue el flujo del
     programa y comienza una
                 !nueva iteracin selccionando otro nodo con exceso (si existe)
                 if(s(i)>0)then
                     go to 2
                 end if
         end if
     end do

     call cpu_time (t2)

!Calculamos el flujo total y el costo total del problema.
    totalCosto=0
    totalFlujo=0
    do l= 1,A
     if(flujoA(l)>0) then
         totalCosto=totalCosto+flujoA(l)*(costoA(l)/(n+1))
         totalFlujo=totalFlujo+flujoA(l)
     end if
    end do
!FINALIZADA RESOLUCI N DEL ALGORITMO
    write(*,*)'Z=',totalCosto
    write(*,*) t2-t1
!ALMACENAMIENTO DE RESULTADOS EN FICHERO DE DATOS
    open (unit=14, file='resultados.txt', STATUS= 'UNKNOWN', access='SEQUENTIAL',
      POSITION='APPEND')
     write(14,14) nameFileOut,N,A, numOfertas,c1, totalcosto,totalflujo, t2-t1
14   FORMAT (A25,3X,I3,3X,I5,3X,I4,3X,I6,3X,   I12,3X,I12,3X,F10.6)
    close(unit=14)
end
```

```
!FUNCION AUXILIAR PARA CADENAS
integer function  token(cadena,pos)

    implicit none
    CHARACTER(len=*) :: cadena
    integer pos
    character(len=10)::resultado
    integer longitud
    integer i
    integer contador
    do i=1,10
        resultado(i:i)=' '
    end do

    longitud =len(cadena)
    contador=1
    i=1
    do while (contador.ne.pos )
        do while (cadena(i:i).ne.' ')
          i=i+1;
        end do
        i=i+1
        contador=contador+1
    end do
  !  write(*,*)'contador=',contador, 'i=',i
    contador=1

    do while(cadena(i:i).ne.' ')
        resultado(contador:contador) = cadena(i:i)
        contador=contador+1
        i=i+1
    end do
    read(resultado,'(I10)') token
end function token

!SUBRUTINA AUXILIAR PARA ELIMINAR ESPACIOS EN BLANCO
subroutine StripSpaces(string)
    implicit none
    character(len=*) :: string
    character(len=len_trim(string))::resultado
    integer :: stringLen
    integer :: last, actual
    integer :: i

    stringLen = len (string)
 !   write(*,*) stringLen,'**',string,'**',resultado,'**'
    last=1
    do while  (string(last:last)==' ')
 !     write(*,*) 'Last=',last,'->',string(last:last)
       last=last+1
    end do

    actual = 2
    resultado(1:1)=string(last:last)
    do i=last+1,stringLen
        if (string(i:i)==' ') then
            if(string(i-1:i-1).ne.' ') then
            resultado(actual:actual)=string(i:i)
            actual=actual+1
            end if
        else
            resultado(actual:actual)=string(i:i)
            actual=actual+1
```

```
        end if
    end do
!   write(*,*) 'Resultado=', resultado
    string=resultado

end
```