Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

# Aplicaciones de MATLAB a comunidades modeladas por grafos

Autor: Francisco Javier Nozal Serrano
Tutor: Manuel Ordóñez Sánchez

**Dpto. Matemática Aplicada II**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2024

DEPARTAMENTO DE MATEMÁTICA APLICADA II

Proyecto Fin de Carrera
Ingeniería de Tecnologías Industriales

# Aplicaciones de MATLAB a comunidades modeladas por grafos

Autor:
Francisco Javier Nozal Serrano

Tutor:
Manuel Ordóñez Sánchez
Profesor titular

Dpto. de Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Proyecto Fin de Carrera: Aplicaciones de MATLAB a comunidades modeladas por grafos

Autor: Francisco Javier Nozal Serrano

Tutor: Manuel Ordóñez Sánchez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

*A mi familia, cuánto ha costado, y cuánto que agradeceros.*

*A mis amigos, cuánto compartido y cuánto que agradeceros.*

*A mis profesores, cuánto que agradecer y que aprender.*

# Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas y entidades que han contribuido en el camino hasta a la realización de este Trabajo de Fin de Grado, que corona esta etapa de formación.

En primer lugar, agradezco a mi tutor, don Manuel Ordóñez Sánchez, por su guía, paciencia y valiosos consejos a lo largo de este proyecto. Su apoyo y conocimientos han sido fundamentales para la culminación de este trabajo.

También quiero agradecer a mis profesores, a todo el personal de administración y servicios y a mis compañeros de la Escuela de Ingenieros de la Universidad de Sevilla, quienes me han brindado su apoyo y han compartido conmigo generosamente su sabiduría, profesionalidad y ejemplo.

A mi familia y todos mis queridos amigos, gracias por vuestro apoyo incondicional, por ser pacientes y por apoyarme en cada etapa de mi formación académica. Sin vuestro apoyo de toda una vida, este logro no habría sido posible.

A Maripaz, que con tanta simpatía y luz ha recorrido a mi lado este camino regalándome tanta ayuda.

*Francisco Javier Nozal Serrano*

*Ingeniero Industrial*

*Sevilla, 2024*

# Resumen

Este Trabajo de Fin de Grado explora el uso de MATLAB para analizar y modelar comunidades y sistemas industriales a través de grafos, mostrando la utilidad y capacidad de MATLAB para el procesamiento de datos y el modelado matemático, que resulta muy valioso en diversas aplicaciones industriales.

En este estudio, se presentan las bases teóricas de los grafos y su relevancia en el modelado de comunidades y sistemas industriales. Se describen diversas técnicas y algoritmos implementados en MATLAB para analizar propiedades de grafos, como la centralidad, la detección de comunidades y la visualización de redes. Además, se discuten ejemplos prácticos y estudios de caso que demuestran la aplicabilidad de estas técnicas en contextos sociales, comunicacionales e industriales.

Los resultados obtenidos demuestran que el uso de MATLAB facilita el análisis y la comprensión de estructuras complejas representadas por grafos, proporcionando una herramienta poderosa para ingenieros y profesionales en diversas áreas.

# Abstract

This Final Degree Project explores the use of MATLAB to analyze and model communities and industrial systems through graphs, demonstrating MATLAB's utility and capability for data processing and mathematical modeling, which is highly valuable in various industrial applications.

This study presents the theoretical foundations of graphs and their relevance in the modeling of communities and industrial systems. It describes various techniques and algorithms implemented in MATLAB to analyze graph properties such as centrality, community detection, and network visualization. Additionally, practical examples and case studies are discussed to demonstrate the applicability of these techniques in social, communicational, and industrial contexts.

The results obtained show that the use of MATLAB facilitates the analysis and understanding of complex structures represented by graphs, providing a powerful tool for engineers and professionals in various fields.

# Índice

# ÍNDICE DE FIGURAS

# 1 INTRODUCCIÓN A LA TEORÍA DE GRAFOS

Un **grafo** puede modelarse como una **serie de nodos o vértices** unidos por una **serie de aristas**. De esta manera, un ejemplo para desarrollar la explicación puede ser el siguiente:

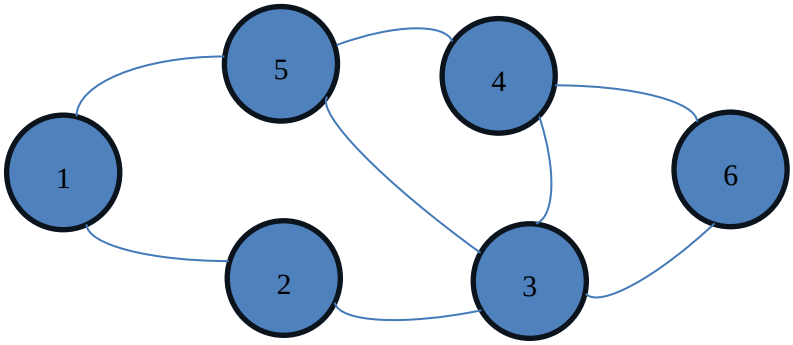



Figura 1 Grafo simple

En la Figura 1, el grafo *G(V, E)*, dónde *V*={1, 2, 3, 4, 5, 6} son los vértices o nodos y *E*={(1,2), (1,5), (2,3), (3,4), (3,5), (3,6), (4,5), (4,6)} son las aristas.

## 1.1 Tipos de grafos

En general, los grafos pueden clasificarse como **multigrafos** en caso de que tengan una arista comunicando un nodo consigo mismo o dos grafos comunicados por más de 1 camino (Figura 2); o como **grafos simples** en caso contrario (Figura 1).

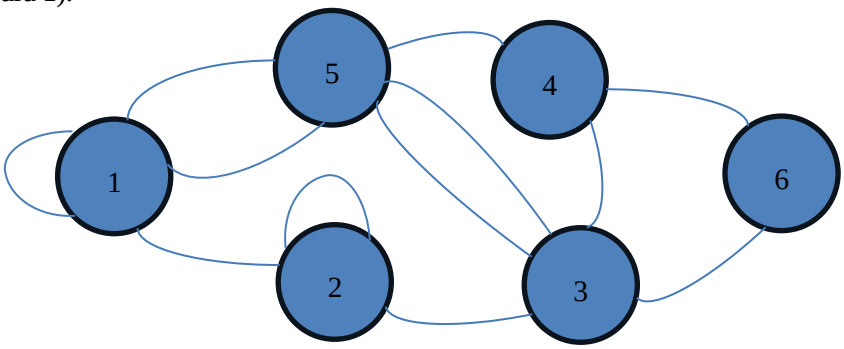



Figura 2 Multigrafo

Se denomina **grafo dirigido** aquel en el que las aristas son pares ordenados *(u, v)*, *u* y *v* pertenecientes al grafo, de manera que *(u, v)* es distinto que *(v, u)*, como se muestra en la figura 3. Un ejemplo es una llamada telefónica de un usuario 1 a un usuario 2: no es lo mismo que 1 llame a 2 que que 2 llame a 1.

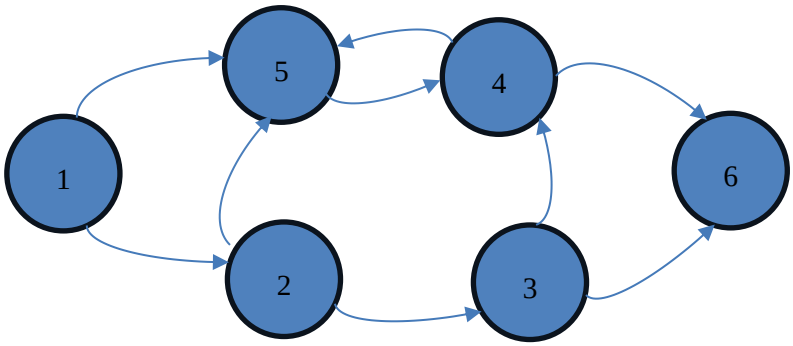



Figura 3 Grafo dirigido

Los grafos dirigidos también suelen referirse como **digrafos**. Por convenio la arista *(u,v)* apunta hacia *v*. Si en un grafo existen las aristas *(u, v)* y *(v, u)*, se dice que *u* y *v* son nodos mutuos.

Dado un grafo *G*, un grafo *G'(V', E')* es un **subgrafo inducido** de G si *V'* y *E'* están respectivamente contenidos en *V* y *E*, y *E'* representa a las aristas que unen los nodos *V'*. Siguiendo con el ejemplo de la Figura 1, el grafo inducido por los nodos *V'*={1, 4, 5} se ve como en la Figura 4.

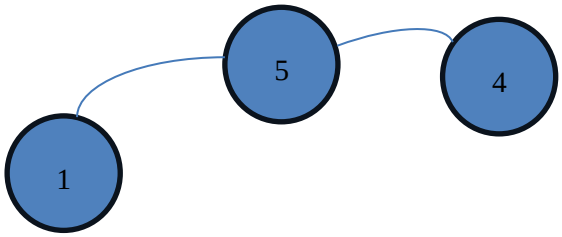



Figura 4 Subgrafo inducido por *V'*={1, 4, 5}

En caso de que las aristas tengan un valor numérico, el gráfico se denomina **grafo con pesos**; puede observarse en la Figura 5. La etiqueta determina el peso de la arista.

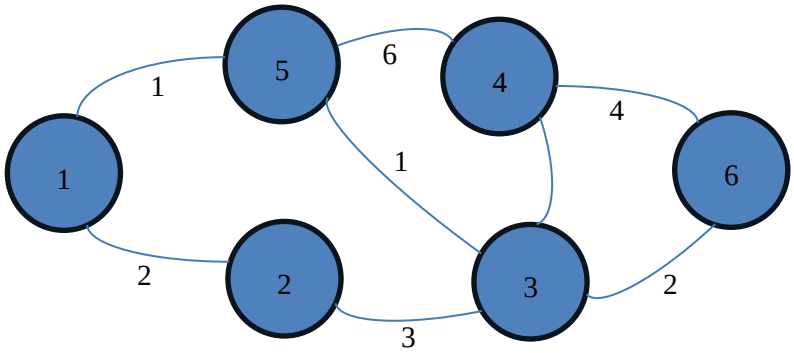



Figura 5 Grafo con pesos

Algunos ejemplos de representaciones de redes mediante grafos son los siguientes:

Tabla 1 Representación habitual de redes

| RED | TIPO DE GRAFO |
|---|---|
| **WWW** | Multigrafo dirigido con bucles, sin pesos |
| **AMISTADES EN FACEBOOK** | No dirigido, sin pesos |
| **RED DE CITAS** | Dirigido, sin pesos, acíclico* |
| **RED COLABORATIVA** | No dirigido, sin pesos |
| **LLAMADAS DE MÓVILES** | Dirigido, con pesos |
| **INTERACCIONES EN PROTEÍNAS** | Multigrafo no dirigido con bucles, sin pesos |

## 1.2 Conectividad

Para entender la conectividad de una red es necesario conocer alguna terminología básica.

### 1.2.1 Adyacencia

Los vértices *u* y *v* pertenecientes a *V* son adyacentes si están unidos por una arista perteneciente a *E*.
Las aristas $e_1$ y $e_2$ pertenecientes a *E* son adyacentes si ambas comparten un extremo conectado un mismo vértice

de *V*.
En el ejemplo de la Figura 1, los nodos 1 y 5 son adyacentes, así como lo son las aristas (1, 5) y (1, 2). No lo son los nodos 1 y 3, ni las aristas (1, 5) y (2, 3).

### 1.2.2 Grado

La **incidencia** es una propiedad de las aristas con respecto a los vértices sobre los que inciden. La arista genérica *(u, v)* es incidente en los vértices *u* y *v*.
El **grado** $d_v$ de un vértice *v* se define como número de aristas que inciden sobre él. En la Figura 6, encima de cada vértice se muestra su grado. Por ejemplo, $d_1$=2 y $d_5$=3. Un grado alto habla del grado de importancia o influencia del vértice dentro del grafo.

Los valores del grado de los vértices pueden ir desde 0 a |*V*|-1.

Se cumple que el sumatorio del grado de todos los vértices es igual a dos veces el número de aristas del grafo, tal que

$$\sum_{v=1}^{|V|} d_v = 2\,|E|$$

Así, el número de vértices con grado impar es par.

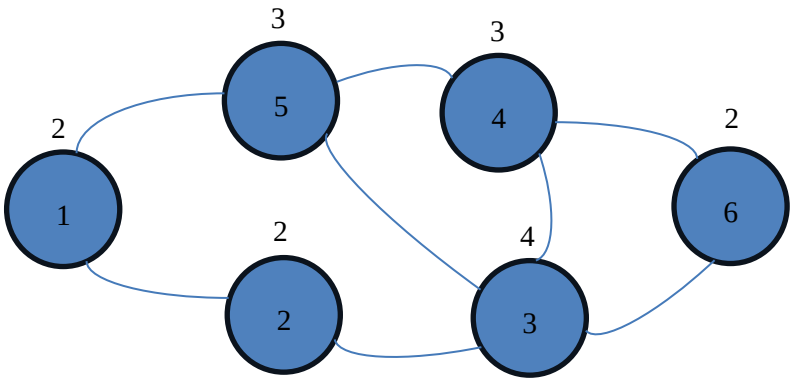



Figura 6 Grados de un grafo

En dígrafos, los vértices tienen un **grado de entrada** (nº de aristas que entran al vértice) y un **grado de salida** (nº de aristas que salen del vértice). De esta manera, por ejemplo, $d_1^{in}$=0, $d_1^{out}$=2 y $d_5^{in}$=3, $d_5^{out}$=1.

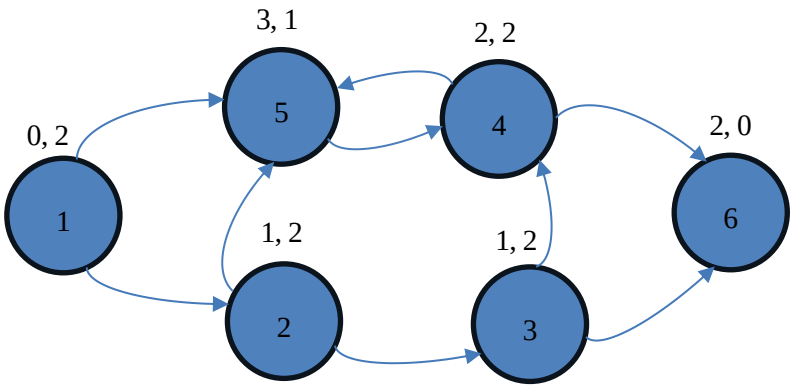



Figura 7 Grado de entrada y salida

Una aplicación real del grado de entrada y salida en relaciones puede ser la jerarquía dentro de una institución. Al nuevo empleado normalmente le llegan tareas por parte de un superior; el gerente adjudica la tarea al empleado a la vez que le llegan directrices de la dirección de la empresa, la dirección gestiona al gerente y le orienta en la actividad a desarrollar.

### 1.2.3 Entorno de un vértice

El entorno $N_i$ de un vértice *i* se define como el conjunto de sus vértices adyacentes. Por ejemplo, $N_5$={1, 3, 4}

(Figura 6). En general, $|N_i| = d_i$.

### 1.2.4 Movimiento en un grafo

Un **camino** de longitud $l$ es una secuencia de vértices diferentes de forma que dos consecutivos son adyacentes, con $l$+1 vértices ($v_0$,…, $v_l$).

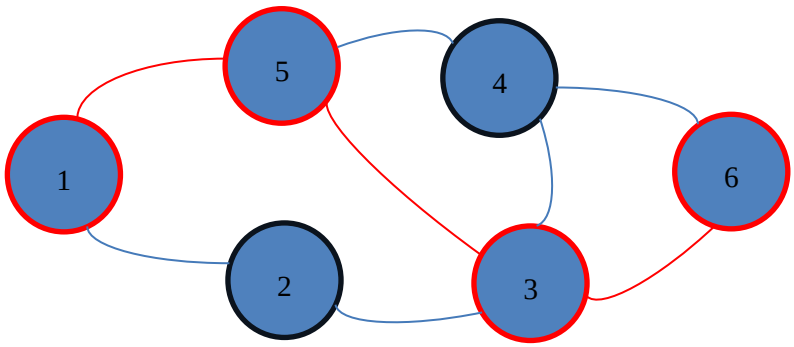



Figura 8 Camino

Un **paseo** es un camino en el que se puede pasar más de una vez por el mismo vértice.
Un **paseo cerrado** ($v_0 = v_l$) se denomina **circuito**.
Un **ciclo** es un camino que empieza y acaba en el mismo vértice (camino cerrado).

### 1.2.5 Distancias en un grafo

La **longitud** de un camino es la suma de los pesos de sus aristas.
La **distancia** entre dos nodos se define como la longitud del camino más corto entre ellos. Si no hay un camino que una a dos vértices, se dice que la distancia entre ellos es infinita. El **diámetro** de un grafo es la longitud mayor entre 2 de sus vértices.
Existen algoritmos que permiten calcular distancias en grafos, como son Dijkstra, Floyd-Warshall o Johnson.

### 1.2.6 Conectividad en un grafo

Un vértice $u$ es **alcanzable** desde otro $v$ si existe un camino entre ellos.
Un grafo es **conexo** si dados dos vértices cualesquiera, éstos son mutuamente alcanzables.

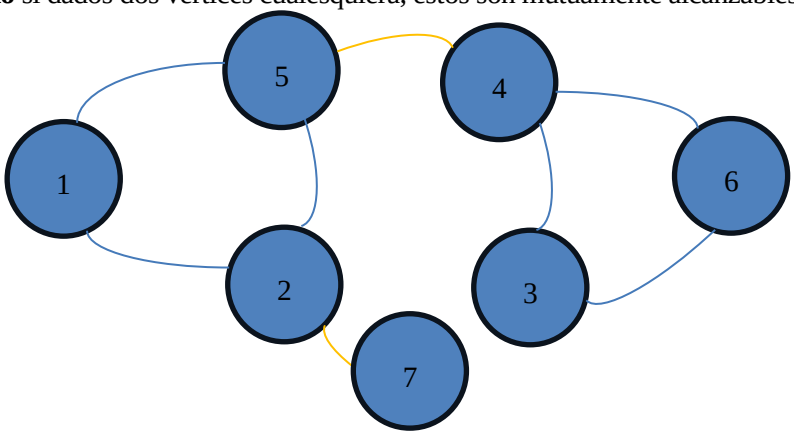



Figura 9 Grafo conexo con 2 puentes

Se denominan **puentes** aquellas aristas que hacen que un grafo sea conexo; si no existieran el grafo no sería conexo, como es el caso de las aristas (2, 6) y (5, 4) en la Figura 9.
Una **componente** es un subgrafo conexo maximal, lo cual significa que no está contenido en ningún subgrafo conexo.

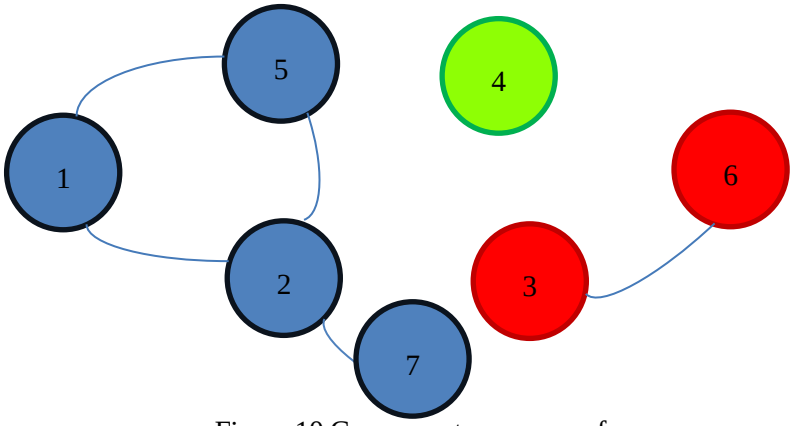


Figura 10 Componentes en un grafo

En la Figura 10, podemos observar cómo las componentes son {1, 2, 5, 7}, {3, 6} y {4}. Sin embargo, el subgrafo {3, 4, 6} no es conexo, o el {1, 2, 5} no es maximal, por lo que ninguno de ellos es una componente.
Un grafo no conexo tiene al menos 2 componentes. La componente mayor (con mayor número de nodos) de un grafo se denomina componente gigante. Un ejemplo de este caso es el de las grandes redes en línea (redes sociales), que suelen presentar una componente gigante, o las relaciones afectivas en el ámbito de la escuela secundaria en EE. UU., estudiadas por Peter Bearman en "Chains of Affection: the Structure of Adolescent Romantic and Sexual Networks" (establece la cadena de quién ha mantenido una relación con quién y analiza lo observado).
En grafos dirigidos el estudio de la conectividad se hace más sutil. Para esta evaluación es importante definir el significado de dos términos:

- Un grafo dirigido está **fuertemente conectado** si para cada par de vértices *u*, *v* del grafo *V*, existe un camino para llegar a *u* desde *v* y viceversa.
- Un grafo dirigido está **débilmente conectado** si omitiendo la dirección establecida por las aristas, existirían caminos para conectar cada par de vértices *u*, *v* del grafo *V*.

Una conexión fuerte implica conexión débil, no así al revés.

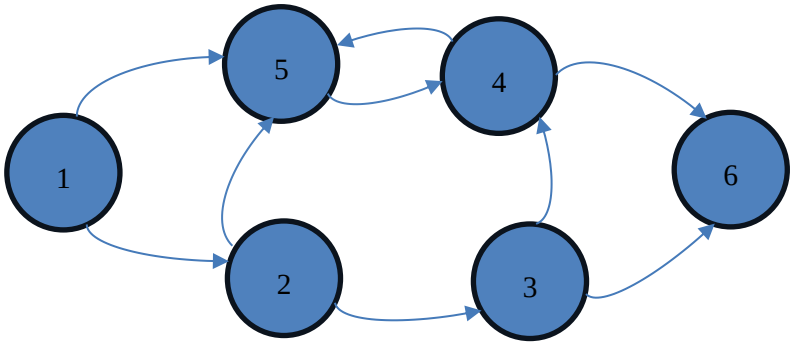


Figura 11 Grafo dirigido débilmente conexo

Un **grafo completo** $K_n$ de orden *n* tiene todas las aristas posibles entre sus vértices. Un grafo completo $K_n$ tiene el mismo número de aristas que pares de vértices, es decir:

$$Nº\ de\ aristas\ de\ K_n = Nº\ de\ pares\ de\ vértices = \frac{n * (n-1)}{2}$$

Algunos ejemplos de grafos completos se muestran a continuación:

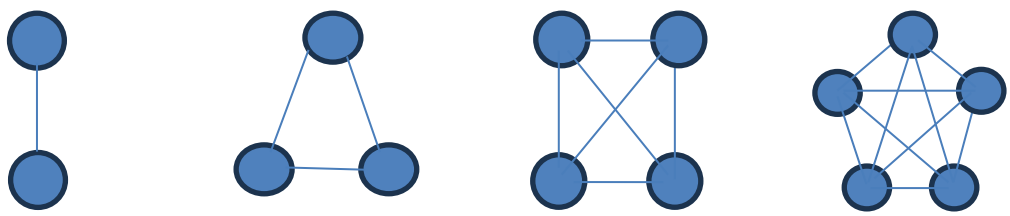
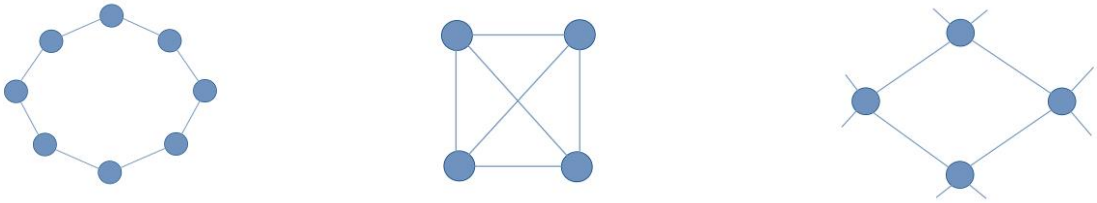


Figura 12 Ejemplos de grafos completos

Estos ejemplos corresponden con los grafos $K_1$, $K_2$, $K_3$, $K_4$ y $K_5$, respectivamente.

Es de interés conocer qué es un **clique**, que se define como un subconjunto de vértices en el cual existe una arista entre cada par de vértices. Por tanto un clique es un **subgrafo completo**, puesto que hay conexión directa entre todos los nodos. Su tamaño está determinado por el número de nodos que contiene. El clique más grande se denomina **clique máximo** y su identificación dentro de un grafo o red es un problema conocido en ciencias de computación. En el ámbito de las redes sociales sirve para identificar grupos de personas fuertemente conectadas; más allá de esto, también es la base de estudios en el campo de la bioinformática y el diseño de redes.

Un **grafo d-regular** cumple que todos sus nodos tienen el mismo grado *d*. Un grafo completo $K_n$ completo es (n-1)-regular. En el caso de un **ciclo** (ver apartado **1.2.4 Movimiento en un grafo**), se trata siempre de un subgrafo 2-regular.

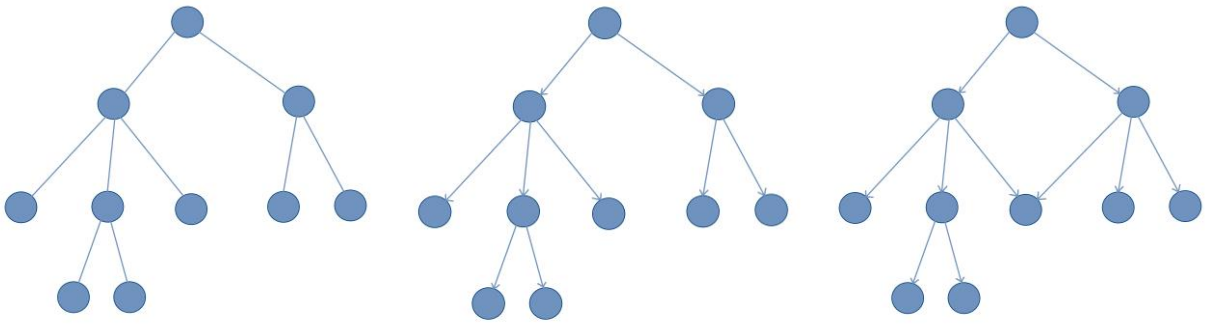


Ciclo (grafo 2-regular) Grafo 3- regular Subgrafo 4-regular

Figura 13 Ejemplos de grafos regulares

Los grafos regulares son frecuentes en el modelado de estructuras cristalinas en física y química, en modelos de adyacencia de pixeles en procesamiento de imágenes y en ciclos de información.

Un **árbol** es un grafo que no tiene ciclos. Un **conjunto de árboles** se denomina **bosque**. Obsérvese a continuación.



Árbol Árbol dirigido Grafo dirigido acíclico

Figura 14 Ejemplos de árboles y DAG

Se dice que un grafo tiene **raíz** en un nodo *u* si desde dicho nodo surgen las conexiones al resto de nodos. De esta manera también se definen los términos padre/s, hijo/s, antecesor, descendiente, hoja…

En un **grafo dirigido acíclico (DAG**, del inglés**)**, un vértice puede tener múltiples padres, mientras que en un árbol cada vértice tiene un solo padre. El grafo subyacente de un DAG (que significa ignorar las direcciones marcadas por las aristas) no tiene por qué ser un árbol.

Se llama grafo **bipartito** a un grafo *G(V, E)* cuando ocurre que el conjunto de los vértices *V* puede ser dividido en dos conjuntos separados y cada arista del conjunto *E* tiene un extremo en el subconjunto $V_1$ y otro en $V_2$. Este tipo de grafo es habitual en la modelización redes de membresía o suscripción, siendo los nodos en $V_1$ las personas suscritas y $V_2$ el propio club o asociación.

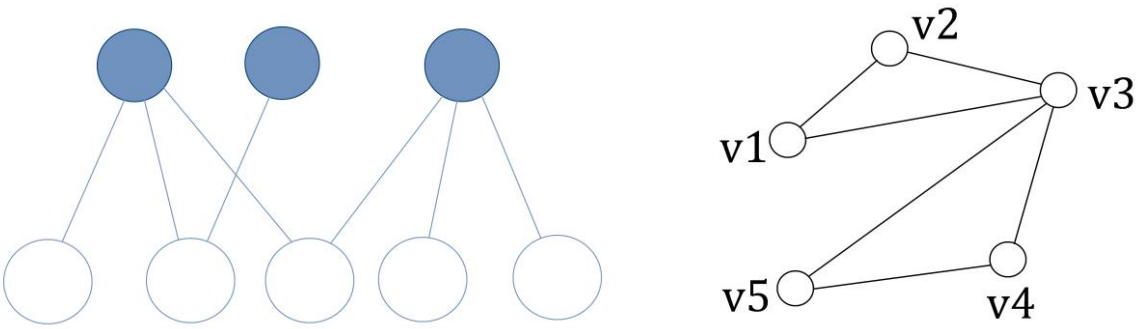



Figura 15 Ejemplo de grafo bipartito (izquierda) y no bipartito (derecha)

En este ejemplo, identificando el grupo azul como $V_1$ y el blanco como $V_2$ se aprecia como no existen aristas entre nodos pertenecientes al mismo grupo.

Comparativamente, puede observarse cómo en la representación de la derecha en la Figura 15, sí que existen aristas conectando vértices del mismo subgrupo (por ejemplo, miembros del mismo club).

# 2 CIERRE TRIÁDRICO Y HOMOFILIA

El cierre triádico y la homofilia son conceptos que ayudan a entender cómo se forman y mantienen las amistades y las redes sociales. Estudiarlos revela por qué tendemos a conectarnos con personas similares a nosotros en nuestras redes sociales. A continuación, este capítulo desarrolla conceptos básicos que permiten su estudio.

## 2.1 Matriz de adyacencia

La teoría de grafos algebraica emplea representaciones matriciales de grafos, de manera que consigue visualizar los grafos gráficamente.

La conectividad de un grafo *G(V, E)* puede expresarse de manera matricial por medio de una matriz binaria simétrica $A \in \{0, 1\}^{|V|\times|V|}$ con valores de $A_{ij}$ tales que:

$$A_{ij} = \begin{cases} 1, & si\ (i, j)\ \in\ E \\ 0, & en\ otro\ caso \end{cases}$$

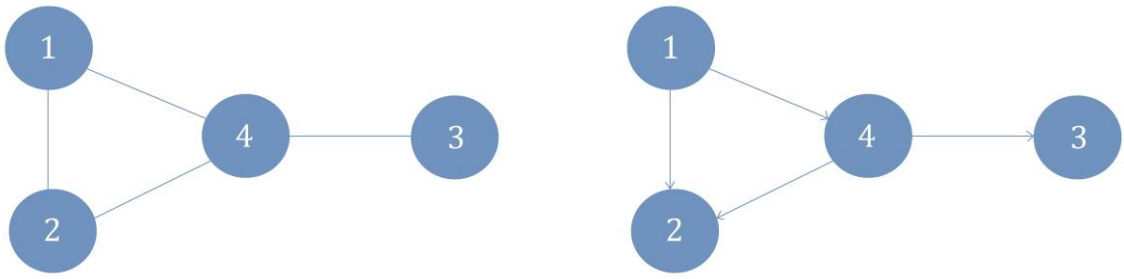
Cada vértice del grafo se le asigna un número entero único, comenzando desde 1 hasta el número total de vértices en el grafo. Si el grafo tiene $n$ vértices, estos vértices se numeran del 1 al $n$. Aquí, $|V|$ representa el número total de vértices en el grafo.

Por ejemplo, si un grafo tiene 5 vértices, estos vértices se notarán como 1, 2, 3, 4 y 5. Esta notación se usa para referenciar los vértices en la matriz de adyacencia, donde la fila y la columna correspondientes a cada índice representan la conexión (o no) entre los vértices.

La matriz de adyacencia *A* es una forma de representar un grafo utilizando una matriz cuadrada, donde las filas y las columnas representan los vértices del grafo. En esta matriz:

Cada elemento $A_{ij}$ corresponde a una pareja de vértices (i, j). Si existe una arista entre dichos nodos en el grafo *G(V, E)*, entonces $A_{ij}$=1. En caso contrario, $A_{ij}$=0.

A continuación, en los siguientes ejemplos puede comprenderse lo explicado con mayor claridad.




$$A_{no\,dirigida} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad A_{dirigida} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Figura 16 Ejemplos de matrices de adyacencia para grafos dirigido y no dirigido

En caso de que haya pesos, el valor de la componente de la matriz en las posiciones que correspondan a una arista existente, en lugar de ser unitario, será el del peso mencionado.

De la misma manera que la matriz de adyacencia permite almacenar información sobre la estructura del grafo, también permite la operación matemática sobre el grafo *G* por medio de su forma matricial *A*.

### 2.1.1 Grados

En la matriz de adyacencia $A_{ij}$, los grados de los vértices pueden calcularse como la suma de los elementos de cada fila. Esto se debe, como puede deducirse de lo explicado previamente, a que en la representación de la matriz de adyacencia, cada fila representa un vértice y cada columna representa una arista posible hacia otros vértices. Así, la suma de los elementos de la fila *i* de la matriz $A_{ij}$ nos da el grado del nodo *i*, $d_i$. La expresión matemática correspondiente es:

$$d_i = \Sigma_{j=1} A_{ij}$$

Como puede observarse en la Figura 16, la matriz de adyacencia de los grafos dirigidos no es simétrica, por lo que la suma de las filas y columnas no es igual. En este caso, la suma de la fila *i=1* da a conocer el grado de entrada $d_{1,in}$, mientras que la suma de la columna *j=1* da el valor del grado de salida $d_{1,out}$.

### 2.1.2 Regularidad

La regularidad de un grafo puede ser estudiada en base a su matriz de adyacencia.

Un grafo *G* es d-regular si y solo si **1** es autovector de *A*, tal que: $\boldsymbol{A1} = \boldsymbol{d1}$.

El vector **1** es un vector columna unitario de las dimensiones del grafo. La multiplicación de la matriz de adyacencia $A_{ij}$ correspondiente por el vector **1** da como resultado un nuevo vector que cumple que sus elementos son la suma de los nodos vecinos al cada nodo, es decir, que son el grado de cada nodo.

Si los valores de las componentes de dicho vector son iguales, supóngase d, entonces sabremos que el grafo *G* es d-regular.

### 2.1.3 Paseos, caminos y ciclos

Mediante el estudio de la matriz de adyacencia también puede obtenerse información sobre los paseos, caminos y ciclos en un grafo *G*.

Llamamos $A^r$ a la r-ésima potencia de *A*, con componentes $A_{ij}^r$. La entrada $A_{ij}^r$ de la matriz $A^r$ representa el número de caminos de longitud *r* que existen entre los vértices *i* y *j*.

Por ejemplo: $A_{ij}^2 = \sum_{k=1}^{n} A_{ik} * A_{kj}$ → Esta expresión cuenta el número de caminos de longitud 2 entre los nodos *i* y *j*.

Por otro lado, mediante el uso de la traza también puede obtenerse información de la matriz de adyacencia, en concordancia con las expresiones siguientes que se explican:

$$\frac{tr(A^2)}{2} = |E| = n^{\underline{o}}\ de\ aristas\ en\ el\ grafo\ G$$

$$\frac{tr(A^3)}{6} = \#\Delta = n^{\underline{o}}\ de\ ciclos\ de\ longitud\ 3\ en\ el\ grafo\ G$$

La traza es la suma de los elementos diagonales de una matriz. Las expresiones anteriores se explican porque $A^2$ cuenta dos veces cada arista, y $A^3$ cuenta cada ciclo de longitud 3 seis veces.

## 2.2 Matriz de incidencia

Otra forma de representación de grafos de la que puede hacerse uso es la de la matriz de incidencia *B*, de dimensiones |*V*| x |*E*|. Si |V|=|E|, entonces *B* será cuadrada, pero no lo será en ningún otro caso, no es una matriz cuadrada en general.

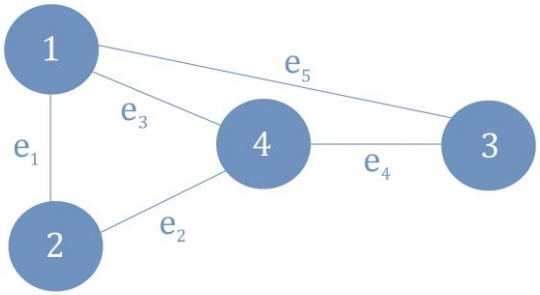
En grafos no dirigidos, las componentes de la matriz *B* tendrán los valores:

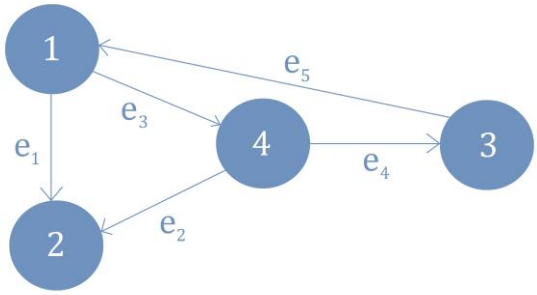
$$B_{ij} = \begin{cases} 1, & si\ el\ nodo\ i\ es\ incidido\ por\ la\ arista\ j \\ 0, & en\ otro\ caso \end{cases}$$

En dígrafos, también se tiene en cuenta la dirección de la arista, reflejado en el signo positivo o negativo de cada componente. En este caso:

$$B_{ij} = \begin{cases} 1, & si\ la\ arista\ j\ llega\ al\ nodo\ i\ (k,i) \\ -1, & si\ la\ arista\ j\ sale\ del\ nodo\ i\ (i,k) \\ 0, & en\ otro\ caso \end{cases}$$

Los siguientes ejemplos pueden ayudar a entender lo explicado.



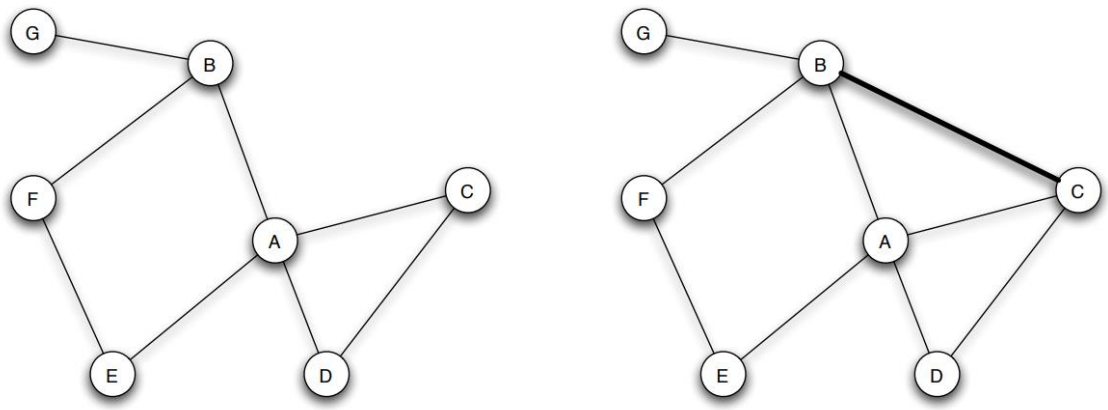

$$B_{no\,dirigida} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \qquad B_{dirigida} = \begin{pmatrix} -1 & 0 & -1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 & 0 \end{pmatrix}$$

De la misma manera que en la matriz de adyacencia, en el caso de grafos con pesos, el valor unitario será cambiado por el del peso de la arista correspondiente.

## 2.3 Cierre triádico

Rara vez las redes son estructuras estáticas, sino que evolucionan con el tiempo. La formación de aristas en estas redes sigue patrones universales, y uno de estos patrones es el cierre triádico.

El cierre triádico se basa en la idea de que si dos personas en una red social tienen un amigo en común, es más probable que se conviertan en amigos en el futuro. Este fenómeno es una característica universal en la evolución de redes. [1]




(a) *Before B-C edge forms.* (b) *After B-C edge forms.*

Figura 17 Ejemplo de formación de una relación entre dos individuos a partir de la relación que ambos tenían con una tercera persona [1]

El cierre triádico es un fenómeno muy natural que puede explicarse por varias razones:

- Oportunidad: Si B y C tienen un amigo en común (A), existe una mayor probabilidad de que B y C se encuentren, ya sea en eventos sociales, actividades organizadas por A, o a través de recomendaciones mutuas.
- Confianza: Si A es amigo tanto de B como de C, es probable que B y C confíen más el uno en el otro debido a su conexión mutua con A. La confianza en redes sociales se fortalece a través de vínculos comunes.
- Incentivo: A puede tener un incentivo para que B y C se hagan amigos. Por ejemplo, podría ser más fácil para A organizar actividades o eventos con ambos presentes si todos son amigos, o A podría beneficiarse de la cooperación entre B y C.

## 2.4 Homofilia

El término homofilia se refiere a la tendencia de las personas a relacionarse con otros individuos con los que tienen algo en común, en aspectos diferentes como raza, edad, intereses, creencias, opiniones, nivel cultural, educacional o socioeconómico.

La formación de redes sociales puede verse influenciada tanto por efectos contextuales como intrínsecos:

- Contextuales: debido a circunstancias externas que agrupan a las personas en un mismo entorno. Por ejemplo, las personas pueden convertirse en amigos porque asisten a la misma escuela, trabajan en la misma empresa, o viven en el mismo vecindario.
- Intrínsecos: debido a conexiones personales previas. Por ejemplo, dos personas pueden hacerse amigas porque un amigo en común las presenta, creando una conexión basada en una relación previamente existente. [2]

En el caso del ejemplo de la Figura 17, los sujetos *B* y *C* tienen alta probabilidad de establecer un vínculo personal como resultado de su común relación con *A*, incluso sin saber que ambos conocen a *A* (en distintas redes sociales existen los perfiles sugeridos o amigos que tienen dos usuarios en común).

Sin embargo, puede surgir la pregunta de si en un grafo y la formación de vínculos entre nodos existe homofilia como característica intrínseca de la red o es sólo un resultado de la forma en que la red ha sido construida. La respuesta a esta pregunta puede darse mediante un estudio o prueba de homofilia.

A continuación se desarrolla un ejemplo que puede ayudar a entender mejor este análisis. Considérese un grupo de 9 niñas y niños, *g=3/9* y *b=6/9*, respectivamente. Si las aristas entre ellos son agnósticas (no hacen distinción entre sexos), la proporción de aristas entre sexos distintos es igual a dos veces el numero de niñas por el numero de niños, 2*g*b. Llevando a cabo la prueba de homofilia, se diría que si la fracción de aristas entre sexos distintos es significativamente menos que ***2*g*b=2*(3/9)*(6/9)= 4/9***, entonces cabría apuntar que la homofilia existe. Siguiendo el ejemplo que se muestra:

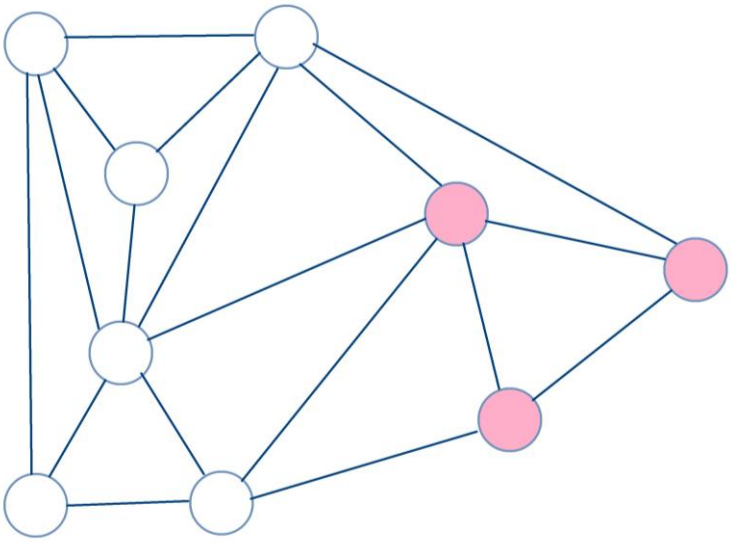


Figura 18 Grafo en representación de grupo de niñas y niños

En el grafo de la figura 18 se representa en forma de grafo el conjunto de niños mencionado en el párrafo previo. En él, existen 18 aristas, de las cuales 5 son entre sexos distintos. En este caso, comparando ***5/18 = 0.27 < 4/9 = 0.44***. En este caso puede sugerirse una homofilia leve, debido a que la fracción observada de aristas entre sexos distintos es menor que la esperada.

$Si\ \frac{aristas\ entre\ sexos\ distintos}{total\ de\ aristas} < 2 * g * b$ → Hay evidencia de homofilia. Esto significa que hay menos aristas entre sexos distintos de las que se esperarían si las aristas fueran agnósticas al sexo de los individuos.

$Si\ \frac{aristas\ entre\ sexos\ distintos}{total\ de\ aristas} \geq 2 * g * b$ → No hay evidencia significativa de homofilia, lo que sugiere que la formación de aristas podría ser independiente del sexo de los individuos.

Usando el ejemplo de una red laboral, según el estudio "Fuerza de los Vínculos Débiles" llevado a cabo por Mark Granovetter en 1973, personas que habían cambiado de trabajo fueron entrevistadas. Se evidenció que la mayoría conocieron de esas oportunidades laborales a través de conocidos, más que por amigos o personas más cercanas. La explicación de esta realidad se basa en propiedades locales de la red (las conexiones individuales de una persona) así como en la estructura global de la red, refiriéndose en este último caso a cómo dichas conexiones se integran en una red más amplia.

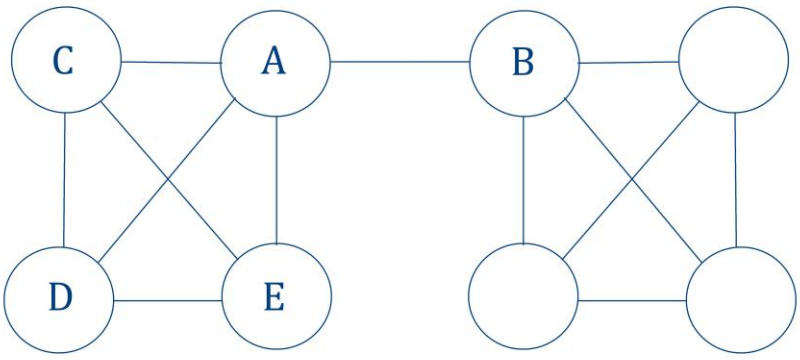



Figura 19 Representación de red en el ámbito laboral

En el caso de las propiedades locales, del grupo cercano, los amigos de una persona, como en el caso de A con sus amigos E, C y D, suelen formar un grupo muy unido. En estos grupos, la información tiende a circular rápidamente y es probable que todos ya conozcan la misma información.

En el de la estructura global, el alcance de la información varía. Un conocido como B, que no está tan integrado en el círculo cercano de A, puede conectarse a una parte diferente de la red. Esto permite que A acceda a nueva

información que no está disponible en su grupo cercano. Asimismo, en la red, el vínculo entre A y B puede actuar como un *puente*. Si se elimina la conexión (A, B), la red podría desconectarse, lo que indica que esta conexión es crucial para mantener la cohesión de la red. Sin embargo, los puentes verdaderos son raros en las redes del mundo real.

A este respecto, Granovetter concluyó que estos *conocidos* ya mencionados (*vínculos débiles*) son valiosos por motivo de esa conexión a distintas partes de la red, dando lugar a una renovación de la información para cada individuo y a una propagación más eficiente de las oportunidades laborales a través de la red.

Acercando el modelo de la Figura 19 a otro más próximo a la realidad, lo habitual es que además del vínculo directo entre A y B, haya entre ellos muchos caminos indirectos diferentes a través de terceros. De esta manera, A es un nexo con B para su red de contactos cercanos, pero su eliminación no supone la desconexión total, sino que aumenta la *distancia* entre estos contactos y B. Esta realidad hace de la arista (A, B) un **puente local**, cuya condición de ser es no formar parte de un triángulo. Dicho de otra manera, los caminos alternativos a (A, B) están a una distancia mayor que 2. Obsérvese en la Figura 20.

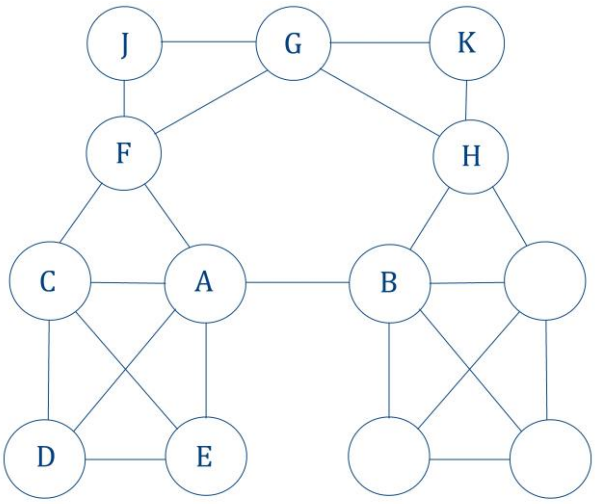



Figura 20 Representación de red laboral cercana a la realidad

Los **puentes verdaderos** (cuya eliminación supone la total desconexión de la red) son raros en redes reales, pero los puentes locales son más comunes y juegan un papel crucial en mantener la cohesión de la red y la conexión ante la eliminación de aristas.

Un grupo unido de amigos, como E, C y D para A, tienen casi la misma información debido a su cercanía y constante comunicación. Este grupo es muy eficiente para compartir información que ya poseen. Sin embargo, un conocido como B, que no es parte del mismo círculo cercano, puede traer información nueva desde una parte diferente de la red. Si la conexión directa entre A y B (que es un puente local) se elimina, A perdería acceso directo a la nueva información que B podría proporcionar.

Existe otro concepto relevante como es el Cierre Triádrico Fuerte, que es una propiedad de las redes sociales que se centra en la relación entre los vínculos fuertes y los vínculos débiles dentro de la red, sugiriendo que si una persona tiene dos amigos cercanos, es probable que estos amigos también se conozcan. Así, una persona A viola el Cierre Triádico Fuerte si tiene vínculos fuertes con dos otras personas B y C, y no hay ningún vínculo (ni fuerte ni débil) entre B y C.

Se definen los vínculos fuertes (S) como la conexión entre amigos o relaciones muy cercanas, mientras que los vínculos débiles (W) corresponden a conocidos o relaciones más superficiales. Por ejemplo, consideremos la red de la Figura 21, donde A tiene una relación fuerte con B (por ejemplo, son mejores amigos) y también tiene una relación fuerte con C (otro mejor amigo). Si no hay ninguna relación (ni fuerte ni débil) entre B y C, entonces A viola el principio de Cierre Triádico Fuerte.

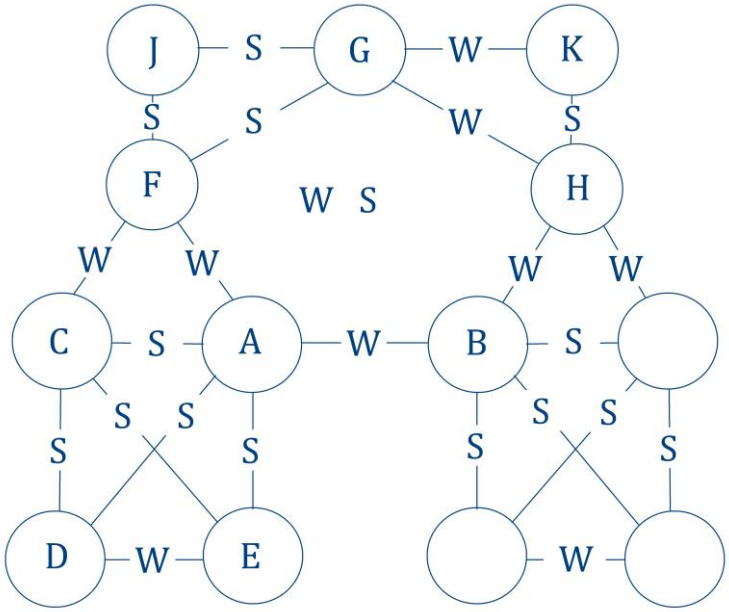



Figura 21 Red laboral con vínculos débiles y fuertes

Con una importancia relevante en el contexto del Cierre Triádrico Fuerte, se distinguen dos características fundamentales: la fuerza de los vínculos y la propiedad de puente. La fuerza del vínculo se refiere a la intensidad, profundidad o nivel de confianza de la arista entre dos nodos, mientras que la propiedad de puente se observa como característica estructural de la red, que describe cómo un vínculo conecta diferentes partes.

Centrándonos en la Figura 22 (página siguiente), si A satisface el Cierre Triádico Fuerte y está involucrado en al menos dos vínculos fuertes, entonces cualquier puente local en el que A esté involucrado debe ser un vínculo débil. Esto se debe a que los vínculos fuertes entre A, B y C implican la existencia de conexiones adicionales que impiden que estos vínculos actúen como puentes locales.

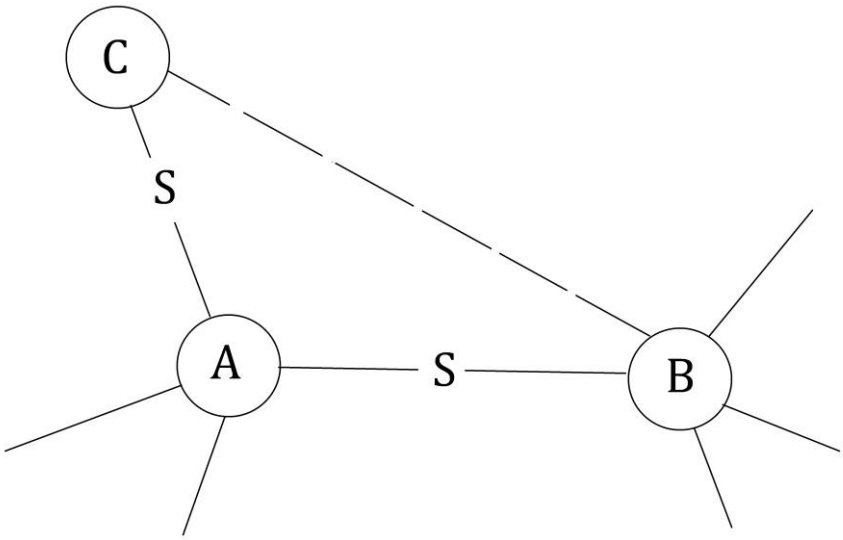



Figura 22 Subgrafo con cierre triádrico

Los conocidos son fuentes naturales de nueva información en una red social debido a su capacidad para conectar diferentes subgrupos. Los modelos que hacen suposiciones estrictas y derivan conclusiones de primer orden (que son una simplificación de una realidad compleja y transversal) ayudan a entender este fenómeno y proporcionan una base para la investigación empírica. Estas características hacen que los vínculos débiles sean esenciales para la dinámica y la cohesión de las redes sociales.

# 3 MEDIDAS DE CENTRALIDAD

LA noción de centralidad en teoría de grafos comprende la identificación de los nodos más importantes en un grafo dentro de la topología de éste, no basada en la naturaleza de un nodo en concreto. Dentro de este análisis de relevancia de nodos, existen diferentes enfoques o maneras de significar el término "importancia", acorde a diferentes **medidas**, llamadas **de centralidad**. En este capítulo se desarrollan la centralidad **de grado, de cercanía, de intermediación y** [3, 4, 5, 6, 7, 8, 9, 10, 11, 12] [13] **de vector propio.** Todas ellas dan lugar a una clasificación de centralidad por nodos dentro del grafo. Estas medidas con ampliamente empleadas en marketing dirigido, en análisis de vulnerabilidad de redes y de dinámicas del poder en redes de intercambio, en control epidemiológico, entre otros muchos campos.

## 3.1 Centralidad de grado

Esta medida analiza la importancia local de un nodo en un grafo, mediante la suma de los pesos de sus aristas incidentes. Se basa en el número de conexiones directas que tiene un nodo con otros nodos. Esta medida es relevante para identificar los nodos que tienen mayor cantidad de enlaces directos en una red, proporcionando una visión básica pero significativa de su relevancia estructural.

La fórmula general para la centralidad de grado ($C_D$) de un nodo *i* es la suma de los pesos de las aristas incidentes, como se muestra:

$$C_D(i) = \sum_{j|(i,j) \in E} A_{ij}$$

donde $A_{ij}$ representa el peso de la arista entre los nodos *i* y *j*, y *E* es el conjunto de todas las aristas del grafo. Esta suma de pesos proporciona una medida cuantitativa de cómo de conectado está un nodo en comparación con los demás.

Un nodo con un alto valor de centralidad de grado tiene un gran número de vecinos (conexiones directas). En grafos de similitud ponderada (grafos con pesos), un nodo con alta centralidad de grado está estrechamente relacionado con sus vecinos debido a los altos pesos de las aristas incidentes. Esto lo convierte en un nodo crucial para la comunicación y difusión de información dentro de la red.

Para redes dirigidas, es importante considerar tanto la centralidad de entrada como la centralidad de salida (véase *1.2.2 Grado*).

Sin embargo, la centralidad de grado tiene la limitación de no capturar los *efectos de cascada* en la red, es decir, no considerar que un nodo puede ser más importante si sus vecinos también son importantes. Este tipo de influencia se refleja mejor en otras medidas de centralidad, como la centralidad de vector propio, que toma en cuenta la importancia de los vecinos de un vértice.

## 3.2 Centralidad de cercanía

Esta medida analiza la velocidad de propagación de la información desde un nodo al resto de nodos de la red, mediante la inversa de la suma de los caminos más cortos a otros nodos ($s_G$), según la expresión que sigue:

$$C_C(i) = \left( \sum_{j \in V} s_G(i,j) \right)^{-1}$$

donde *V* es el conjunto de todos los nodos del grafo.

Los nodos que están cerca de otros en promedio tienen alta centralidad de cercanía. Estos nodos pueden tener una influencia más directa sobre los demás, ya que pueden comunicarse rápidamente con cualquier otro nodo en la red. Esta característica los hace especialmente importantes para funciones como la difusión de información

y la coordinación dentro de la red, aumentando su valor estratégico en la estructura del grafo.

Un ejemplo ilustrativo de la centralidad de cercanía se encuentra en la red de actores de cine, donde dos actores están conectados si han trabajado juntos en una película. En esta red, Christopher Lee tiene la centralidad de cercanía más alta (0.4143), lo que le permitió entrar en el Libro Guinness de los Récords en 2007 por tener el mayor número de créditos en pantalla. Por otro lado, Leia Zanganeh, una actriz iraní de cine y teatro, tiene una centralidad de cercanía baja (0.1154), indicando que está menos conectada en esta red específica.

En este caso, se destacan dos limitaciones de la centralidad de cercanía.

Primero, abarca un rango dinámico muy pequeño, lo que puede limitar su capacidad para distinguir claramente entre la importancia relativa de los nodos. Esto significa que, frecuentemente, los valores de centralidad de cercanía para diferentes nodos están muy cerca unos de otros, dando lugar a una difícil ordenación de los nodos según su importancia relativa. Incluso si un nodo es ligeramente más central que otro, la diferencia en sus valores de centralidad de cercanía puede ser tan pequeña que no se aprecie una diferencia significativa en términos prácticos.

Segundo, en grafos no conexos (grafos en los que hay algún nodo aislado), la centralidad de cercanía es cero para todos los nodos, ya que no existen caminos que conecten todos los nodos. Esto limita su aplicabilidad en este tipo de redes, porque la centralidad de cercanía se calcula como la suma de las distancias más cortas desde un nodo a todos los demás, y si no existe una ruta entre ciertos nodos, la distancia entre ellos es infinita (la suma de distancias infinitas elevadas a -1 da lugar a división entre infinito, resultando en 0).

Se concluye por tanto que la medición de la centralidad de cercanía solo es útil en redes conexas.

## 3.3 Centralidad de intermediación

Esta medida analiza la influencia que tiene un nodo sobre el flujo total de información que se transmite a segundo nodo dentro de la misma red, mediante la identificación de los nodos que se encuentran en los caminos más cortos entre muchos pares de nodos, haciéndolos influyentes en la comunicación y el flujo de datos en la red.

La expresión empleada que se muestra a continuación realiza la medición calculando la proporción de caminos que llegan a un nodo *k* desde un nodo *j* pasando por un nodo *i* en comparación con todos los caminos que llegan a *k* desde *j*. Es decir, en qué medida los caminos que llegan a *k* están precedidos por un paso por *i*:

$$C_B(i) = \sum_{\substack{j,k \in V \\ j \neq i \neq k}} \frac{\sigma_{jk}(i)}{\sigma_{jk}}$$

donde $\sigma_{jk}$ representa el número de caminos más cortos que van de *j* a *k* y $\sigma_{jk}(i)$ al número de caminos más cortos de *j* a *k* que pasan por *i*.

El siguiente ejemplo puede ayudar a afianzar el concepto de centralidad de intermediación. En la red de actores de cine, un personaje destacado es el actor español Fernando Rey, quien tiene la centralidad de intermediación más alta, con un valor de $7.47 * 10^8$. Esto es así porque Fernando Rey participó en muchas películas francesas, americanas y españolas, conectando así diferentes subredes de actores, lo cual le hizo adquirir una centralidad de intermediación muy alta y servir como un puente crítico en la red.

Como contrapunto, cabe destacar que la centralidad de intermediación no es una medida de la conectividad general de un nodo, sino más bien de su peso sobre el total de flujo de información de la red: un nodo puede tener alta intermediación, aunque no cuente con muchos vecinos.

## 3.4 Interpretación de resultados a partir de la centralidad de grado, cercanía e intermediación

La centralidad de grado en una red social revelaría quién tiene más conexiones, es decir, quién tiene más amigos en la red. Esto hace de los nodos con alta centralidad de grado nodos potencialmente superiores en influencia

social (nótese el término potencialmente: es un nodo con capacidad de llegar a muchos otros pero que no tiene por qué hacerlo; podría ejemplificarse en una persona con muchos amigos, pero con carácter tímido o sin interés por difundir información, por el motivo que sea).

La centralidad de cercanía indica quién puede llegar rápidamente a todos los demás en la red, directa o indirectamente. Esto es relevante en la difusión rápida de la información en la red social. Los nodos con alta centralidad de cercanía destacan por su eficiencia para la distribución de la información rápidamente a través de la red.

La centralidad de intermediación da una idea sobre el poder de transmisión o la influencia en el flujo de información a través de una red, actuando como un puente entre diferentes grupos. Un nodo con alta centralidad de intermediación cobra especial relevancia en la comunicación entre grupos separados dentro de la red, por ser esenciales en la cohesión social. De la misma manera, en una red social, revela los "puntos de corte" del grafo, es decir, los nodos cuya eliminación causaría la división de la red en múltiples componentes.

La centralidad de *grado* revela la *popularidad*, la centralidad de *cercanía* indica la *eficiencia en la difusión de información*, y la centralidad de *intermediación* señala la *influencia en la comunicación entre grupos*.

## 3.5 Centralidad de vector propio

Esta medida tiene su base en la combinación lineal de la centralidad del conjunto de nodos que rodean a otro nodo (centralidad de los vecinos). Unos pocos vecinos con mucha importancia pueden tener más influencia que otro conjunto mayor de vecinos con menos importancia. De esta forma, un nodo es considerado central si está conectado a otros nodos centrales. Esto se concreta en la siguiente expresión:

$$C_E(i) = \frac{1}{\lambda} \sum_{(i,j) \in E} A_{ij} C_E(j)$$

donde $\lambda$ es el valor propio correspondiente. En la expresión lo que queda dicho es que la centralidad de un nodo es proporcional a la suma de la centralidad de sus vecinos. Si eres amigo de muchas personas cuya influencia en la red es pequeña o nula, tu centralidad no es potenciada a consecuencia de tu vínculo con ellos.

Esta definición lleva a un problema de autovectores, en el que $C_E$ es el autovector dominante de la matriz de adyacencia, la cual, recordemos, representa las conexiones entre nodos de la red.

## 3.6 Analisis de la centralidad y el poder en redes: influencia de la Familia Medici en Florencia en el siglo XV

Después de lo ya desarrollado, es fácil entender que la centralidad en las redes otorga un poder significativo, el caso de los Medici puede servir para ilustrar este análisis. Su alta centralidad de intermediación les permitió dominar las estructuras políticas y sociales de Florencia, convirtiéndose en la familia más influyente de su tiempo. Una respuesta razonable a por qué los Medici fueron la familia más influyente en la Florencia del siglo XV puede encontrarse en la estructura política y de amistad de la época. Según el estudio “Centrality and network flow” de Padgett y Ansell en 1993, la familia Medici alcanzó un alto nivel de influencia debido a su alta centralidad en la red social y política de Florencia.

La familia Medici se encontraba ligada a muchas otras tanto en lo familiar como en lo político y social. La red de Florencia en ese momento estaba compuesta por diversas familias nobles que establecían alianzas políticas y sociales a través de matrimonios y acuerdos comerciales. Los Medici lograron posicionarse estratégicamente en esta red, formando conexiones clave que les permitieron acceder y controlar la información y los recursos. [14]

# 4 APLICACIÓN 1ª: TEORÍA DE GRAFOS CON MATLAB

En esta primera aplicación vamos a explicar cómo dar los primeros pasos en la Teoría de Grafos utilizando el programa MATLAB. Para ello comenzaremos explicando cómo se pueden definir grafos, hacer sus representaciones gráficas y solicitar información relevante sobre los vértices. A continuación, estudiaremos el problema de la conexión en grafos tanto orientados como no orientados.

## 4.1 Primeros pasos con Matlab

### 4.1.1 Definir grafos

Comencemos explicando cómo podemos definir un grafo. MatLab nos permite definir grafos de dos maneras, de forma matricial y de forma vectorial. A continuación explicamos cómo hacerlo en cada uno de los casos.

#### 4.1.1.1 Definir un grafo de forma matricial

Comenzamos definiendo una matriz cuadrada A, de manera que su dimensión coincida con el número de vértices del grafo. Si queremos definir simplemente un grafo y no una red, la matriz A es justamente la matriz de adyacencia asociada al grafo, de manera que para cada par de vértices *i, j*:

$$a_{ij} = \begin{cases} 0 \ \ si\ i \to j \\ 1\ si\ i \nrightarrow j \end{cases}$$

Ahora bien, si queremos definir una red (esto es, un grafo en el que cada arco tenga asignado un valor numérico), la matriz *A* será la matriz de adyacencia con pesos. Esto significa que el valor $a_{ij}$ será 0 si el arco $i \to j$ no existe, y tomará un valor real no nulo si el arco existe y tiene asignado dicho peso.

Una vez definida la matriz de adyacencia con pesos, debemos tener en cuenta si el grafo es orientado o no orientado. En el caso de grafos orientados, utilizaremos la instrucción:

$$\gg G = digraph(A)$$

Por otra parte, si el grafo es no orientado, la matriz de incidencia *A* debería de ser simétrica ($a_{ij} = a_{ji}$). En este caso, utilizaremos la instrucción:

$$\gg G = graph(A)$$

Si la matriz introducida es no simétrica y utilizamos la instrucción *graph*, MATLAB dará un error, indicando que para construir un grafo no orientado es necesario utilizar una matriz simétrica.

**Ejemplo 4.1.** *Consideremos el grafo orientado G y su grafo no orientado asociado que denotaremos por F, definidos en la Figura 23. Como podemos ver, cada arco tiene asociado un peso. En el caso de los arcos 2→3 y 3→2, ambos tienen asociado el mismo peso 2.*

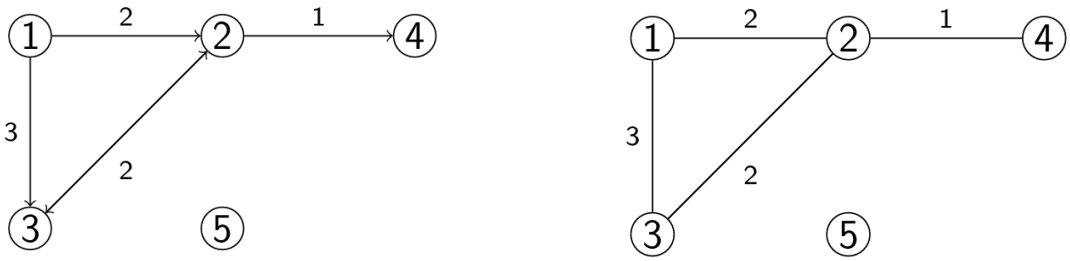



Figura 23 Grafo orientado *G* (izquierda) y su grafo no orientado asociado (derecha), denotado por *F*. Para el grafo orientado, el peso de los arcos 2→3 y 3→2 es en ambos casos de 2.

*Las matrices de adyacencia con pesos de estos grafos, que denotaremos por $A_1$y $A_2$, vienen dadas por:*

$$A_1 = \begin{pmatrix} 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}; \; A_2 = \begin{pmatrix} 0 & 2 & 3 & 0 & 0 \\ 2 & 0 & 2 & 1 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

*Como podemos observar, la matriz $A_2$, al estar asociada a un grafo no orientado, es simétrica. Para introducirlas en MatLab, usamos las siguientes órdenes:*

$$>> \; A1 = [0\ 2\ 3\ 0\ 0; 0\ 0\ 2\ 1\ 0; 0\ 2\ 0\ 0\ 0; 0\ 0\ 0\ 0\ 0; 0\ 0\ 0\ 0\ 0];$$

$$>> \; A2 = [0\ 2\ 3\ 0\ 0; 2\ 0\ 2\ 1\ 0; 3\ 2\ 0\ 0\ 0; 0\ 1\ 0\ 0\ 0; 0\ 0\ 0\ 0\ 0];$$

*Una vez introducidas las matrices de incidencia con pesos, podemos de definir los grafos orientados y no orientados, respectivamente, mediante:*

$>> \; G = digraph(A1)$

$G =$

$digraph\ with\ properties$:

$Edges$: [$5x2\ table$]

$Nodes$: [$5x0\ table$]

$>> \; G = graph(A2)$

$F =$

$graph\ with\ properties$:

$Edges$: [$4x2\ table$]

$Nodes$: [$5x0\ table$]

*Como vemos, la salida nos informa de que el grafo orientado G tiene 5 arcos y 5 nodos, mientras que el grafo no orientado F sigue teniendo 5 vértices, pero solamente 4 aristas (los arcos 2→3 y 3→2 dan lugar a la misma arista).*

A la hora de de definir un grafo, podemos estar interesados en dar un nombre a cada uno de los vértices. Por ejemplo, si cada vértice representa una ciudad, una localización, . . . Para introducir dichos nombres, simplemente necesitamos definir un vector de caracteres que contenga los nombres de cada vértice. Por supuesto, el tamaño del vector deberá de coincidir con el número de vértices del grafo. Para un grafo con tres nodos la instrucción sería:

$>> \; Nodos = \{`Nodo\ 1', `Nodo\ 2', `Nodo\ 3'\};$

$>> \; G = digraph(A, Nodos)$

En el caso de un grafo no orientado, simplemente sustituiríamos la instrucción *digraph* por la instrucción *graph*.

Una vez definido el grafo, podemos **solicitar por pantalla la información referida a los vértices, a los arcos/aristas o a los pesos** mediante las instrucciones:

$>> G.Nodes$

$>> G.Edges$

$>> G.Edges.Weight$

**Ejemplo 4.2.** *Continuando con el Ejemplo 4.1, cada nodo hace referencia a un aeropuerto, mientras que los arcos hacen referencia a conexiones entre aeropuertos, y su peso será el tiempo de vuelo. En el caso del grafo $F$ no orientado, cada arista hará referencia a la presencia de una conexión entre ambos aeropuertos y en ambas direcciones, mientras que para el grafo $G$ orientado cada arco $i \rightarrow j$ hace referencia a la existencia de un vuelo desde el aeropuerto $i$ hasta el aeropuerto $j$. En este caso, podríamos introducir esta información de la siguiente manera:*

$>> A1 = [0\ 2\ 3\ 0\ 0; 0\ 0\ 2\ 1\ 0; 0\ 2\ 0\ 0\ 0; 0\ 0\ 0\ 0\ 0; 0\ 0\ 0\ 0\ 0];$

$>> Nodos = \{`Aeropuerto\ 1', `Aeropuerto\ 2', `Aeropuerto\ 3', `Aeropuerto\ 4', `Aeropuerto\ 5'\};$

$>> G = digraph(A1, Nodos)$

$G$

$digraph\ with\ properties:$

$Edges: [5x2\ table]$

$Nodes: [5x0\ table]$

$>> G.Nodes$

$ans =$

$Name$

_____

$`Aeropuerto\ 1'$

$`Aeropuerto\ 2'$

$`Aeropuerto\ 3'$

$`Aeropuerto\ 4'$

$`Aeropuerto\ 5'$

$>> G.Edges$

$ans =$

| $EndNodes$ | | $Weight$ |
|---|---|---|
| $`Aeropuerto\ 1'$ | $`Aeropuerto\ 2'$ | 2 |
| $`Aeropuerto\ 1'$ | $`Aeropuerto\ 3'$ | 3 |
| $`Aeropuerto\ 2'$ | $`Aeropuerto\ 3'$ | 2 |
| $`Aeropuerto\ 2'$ | $`Aeropuerto\ 4'$ | 1 |
| $`Aeropuerto\ 3'$ | $`Aeropuerto\ 2'$ | 2 |

*De esta manera disponemos de la información relativa tanto a los nodos como a los arcos. Análogamente podemos proceder con el grafo no orientado F.*

#### 4.1.1.2 Definir un grafo a través de los arcos

Para definir grafos usando arcos/aristas, distinguiremos una vez más entre grafos orientados y no orientados:

a) Grafos orientados:

En este caso, definimos dos vectores, un vector $s$ y un vector $t$, ambos con tamaño igual a $m$ (el número de arcos en el grafo), de manera que $s(i)$ y $t(i)$ indican los vértices inicial y final del $i$-ésimo arco, respectivamente. A continuación, definimos el grafo utilizando la siguiente instrucción:

$$\gg\ G = digraph(s,t)$$

Además, si queremos añadir pesos a los arcos, podemos definir un vector peso de manera que en la posición $i$ asigne el peso del arco $s(i)\rightarrow t(i)$. La instrucción a utilizar sería en este caso:

$$\gg\ G = digraph(s,t,peso)$$

b) Grafos no orientados: En este caso, volvemos a definir dos vectores *s* y *t*, ambos con tamaños igual a m (el número de aristas), de manera que *s(i)* y *t(i)* indican los vértices inicial y final de la *i*-ésima arista. En este caso, el orden en el que pongamos los vértices inicial y final es indiferente y solamente lo introduciremos una vez (por ejemplo, la arista 1 - 2 se puede introducir como 1 - 2 o 2 - 1, pero solamente se introducirla en una ocasión). La orden que debemos utilizar es:

$$\gg\ G = graph(s,t)$$

Al igual que en el caso de grafos orientados, si queremos añadir pesos, utilizaríamos la instrucción:

$$\gg\ G = graph(s,t,peso)$$

**Ejemplo 4.3.** *Continuemos con los grafos G y F del Ejemplo 5.1 definidos en la Figura 23. Si queremos definir el grafo G utilizando los arcos, debemos definir los vectores s, t y peso de manera que la componente i-ésima contenga los vértices inicial y final del arco i-ésimo y el peso de dicho arco:*

$>>\ \ s = [1\ 1\ 2\ 2\ 3];$

$>>\ \ t = [2\ 3\ 3\ 4\ 2];$

$>>\ \ peso = [2\ 3\ 2\ 1\ 2];$

*Con el orden introducido, estaríamos diciendo que el primer arco es 1→2 con un peso de 2, el segundo arco 1→3 con un peso de 3 y así sucesivamente. Por último, definiríamos el grafo G mediante la expresión:*

$>>\ G = digraph(s,t,peso)$

$G$

$digraph\ with\ properties$:

$Edges$: $[4x2\ table]$

$Nodes$: $[5x0\ table]$

*En el caso del grafo no orientado F, hemos de introducir en los vectores s y t los vértices de cada arista. Una posibilidad para hacerlo sería la siguiente:*

$>>\ s = [1\ 1\ 2\ 2];$

$>>\ t = [2\ 3\ 3\ 4];$

$>>\ peso = [2\ 3\ 2\ 1];$

*Y definimos el grafo no orientado mediante:*

$>>\ F = graph(s,t,peso)$

$F$

$graph\ with\ properties$:

$Edges$: $[4x2\ table]$

$Nodes$: $[4x0\ table]$

Al igual que hicimos en el caso de grafos definidos a través de la matriz de incidencia con pesos, cuando definimos el grafo a través de los arcos también podemos asignar nombres a los vértices. En este caso, debemos introducir los vectores *s* y *t* que definen los arcos/aristas del grafo, a continuación un vector de pesos y un último vector de caracteres con los nombres de los vértices. En el caso de un grafo orientado con 3 vértices, la expresión sería:

$>>\ Nodos = \{`Nodo\ 1', `Nodo\ 2', `Nodo\ 3'\};$

$>>\ G = digraph(s, t, peso, Nodos)$

En el caso de grafos no orientados, simplemente sustituiríamos la instrucción *digraph* por *graph*. Al igual que hicimos anteriormente, una vez definido el grafo podemos utilizar los comandos *G.Nodes* y *G.Edges* para obtener por pantalla la información sobre los vértices y los arcos/aristas, respectivamente.

Hemos visto cómo definir grafos usando o bien la matriz de incidencia o bien los arcos. En ambos casos, el grafo obtenido es el mismo, así que utilizaremos la manera que en cada caso nos resulte más cómoda.

### 4.1.2 Modificación de vértices y arcos/aristas

Una vez definido un grafo (tanto orientado como no orientado), podemos realizar ciertas operaciones como añadir o quitar vértices y/o arcos/aristas. Explicamos a continuación cómo realizar cada una de las operaciones:

a) **Añadir vértices.** Para añadir vértices a un grafo *G*, utilizaremos la instrucción *addnode*, donde debemos especificar el nombre del grafo creado y, o bien el número de vértices a crear, o bien un vector de caracteres con los nombres de los nuevos vértices. Si por ejemplo deseamos añadir dos nodos, podríamos utilizar una de las siguientes instrucciones:

$>>\ H = addnode(G, 2)$

$>>\ H = addnode(G, \{`Nodo\ 1', `Nodo\ 2'\})$

En el primer caso añadirá dos vértices que los numerará continuando con la numeración presente en *G*, mientras que con la segunda instrucción crearía dos vértices a los que llamaría *Nodo 1* y *Nodo 2*. En ambos casos, el nuevo grafo se almacenaría con el nombre *H*.

b) **Eliminar vértices.** La eliminación de vértices en un grafo *G* se hace mediante el comando *rmnode*. En dicho comando se debe de especificar el grafo al que hacemos referencia, así como los vértices a eliminar. Si éstos están simplemente numerados, se indicará el número correspondiente a dichos vértices; si por contra los vértices tienen un nombre en forma de caracteres, utilizaremos un vector de caracteres para indicar los vértices que deseamos eliminar. Por ejemplo, si se desea eliminar de un grafo *G* el vértice numerado como 3, usaremos la expresión:

$H = rmnode(G, 3)$

Hay que decir que al eliminar un vértice, inmediatamente se eliminan también todos los arcos

incidentes en él.

c) **Añadir arcos.** Para añadir arcos utilizaremos una expresión similar a la utilizada para definir los grafos a través de los arcos. El comando a utilizar es *addedge*, donde indicaremos el nombre del grafo y a continuación dos vectores *s* y *t* que contendrán, en el caso de grafos orientados, los vértices inicial y final de los arcos a añadir, respectivamente, mientras que en el caso de grafos no orientados *s* y *t* simplemente contendrán los vértices que definen los arcos así como un vector de pesos. Por ejemplo, si deseamos añadir los arcos *3→5* y *2→1*, con los pesos 4 y 5, respectivamente, usaremos las siguientes instrucciones:

$>>\ s = [3\ 2];$

$>>\ t = [5\ 1];$

$>>\ peso = [4\ 5];$

$>>\ H = addedge(G, s, t, peso)$

En caso de que los vértices hayan sido nombrados con caracteres, los vectores *s* y *t* serán también vectores de caracteres.

d) **Eliminar arcos.** Para eliminar arcos de un grafo usaremos la expresión *rmedge*. En dicha expresión debemos de indicar el nombre del grafo y, al igual que en el apartado anterior, los vectores *s* y *t* que definan los arcos/aristas a eliminar. Por ejemplo, si se desea eliminar los arcos *3→1* y *3→4*, se utilizarán los siguientes comandos:

$>>\ s = [3\ 3];$

$>>\ t = [1\ 4];$

$>>\ H = rmedge(G, s, t)$

Una vez más, si los vértices tienen un nombre en forma de caracter, los vectores *s* y *t* serán vectores de caracteres indicando los arcos/aristas a eliminar.

**Ejemplo 4.4.** *En el grafo orientado G definido en el Ejemplo 4.1, podemos observar que el Aeropuerto 5 no tiene conexiones con ninguno de los otros aeropuertos, así que podemos eliminarlo de nuestro grafo:*

$>>\ G2 = rmnode(G, \{`Aeropuerto\ 5'\})$

*Además, dado que el vuelo desde el Aeropuerto 1 al Aeropuerto 2 ha perdido un gran número de viajeros en los últimos meses, la compañía que operaba este vuelo lo ha cancelado. Hemos de eliminar por tanto este arco:*

$>>\ G3 = rmedge(G2, \{`Aeropuerto\ 1'\}, \{`Aeropuerto\ 2'\})$

*Sin embargo, para no perder su presencia en el Aeropuerto 1, la compañía ha decidido crear una nueva conexión tanto de ida como de vuelta desde el Aeropuerto 1 a un nuevo aeropuerto que llamaremos Aeropuerto 6. El tiempo de vuelo es de 1.5 horas, tanto para la ida y la vuelta:*

$>>\ G4 = addnode(G3, \{`Aeropuerto\ 6'\})$

$>>\ G5 =$
$addedge(G4, \{`Aeropuerto\ 1', `Aeropuerto\ 6'\}, \{`Aeropuerto\ 6', `Aeropuerto\ 1'\}, [1.5\ 1.5])$

*Con esto, el nuevo grafo obtenido (que finalmente ha sido llamado G5), se puede representar gráficamente como muestra la Figura 24.*

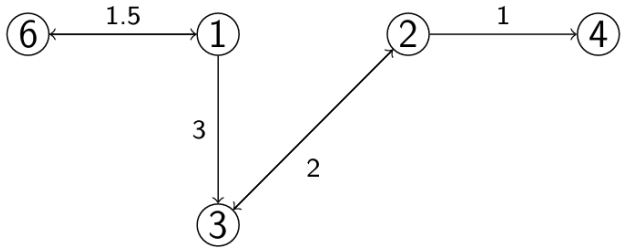



Figura 24 Grafo G5 modicado tras añadir y eliminar aeropuertos y conexiones.

Otros dos comandos que pueden ser de ayuda son *numnodes* y *numedges*, que nos indican el número de vértices y de arcos que hay el grafo, respectivamente. En el Ejemplo 4.4 anterior, si queremos saber el número de vértices y de arcos que hay en el grafo final obtenido (llamado *G5*), simplemente utilizaremos las siguientes instrucciones:

$>>\ numnodes(G5)$

5

$>>\ numedges(G5)$

6

### 4.1.3 Representación gráfica

Una vez definido un grafo en MATLAB, podemos hacer su representación gráfica. Esto se puede hacer de manera sencilla mediante el comando *plot*, indicando el nombre del grafo que estamos trabajando. Si el grafo se ha almacenado con el nombre *G*, utilizaremos la instrucción:

$>> plot(G)$

La instrucción plot permite hacer una primera representación del grafo en MATLAB. Sin embargo, tenemos varias opciones para mejorar el gráfico obtenido.

a) **Layout.** Podemos solicitar que MATLAB dibuje los vértices y arcos de manera diferente, utilizando el comando *layout* con algunas de las siguientes opciones: *'circle'*, *'force'*, *'layered'*, *'subspace'*. Por ejemplo:

$>> h =$
$plot(G)$ # $Almacenamos\ el\ gráfico\ con\ el\ nombre\ h\ para\ referirnos\ a$ él $a\ continuación.$

$>> layout(h, `force')$

b) **Labeledge.** En un gráfico podemos añadir un nombre a cada uno de los arcos. Por ejemplo, podemos estar interesados en añadir a cada arco su peso. En tal caso usaríamos el comando:

$>> labeledge(h, 1{:}numedges(G), peso)$

De esta manera, a cada uno de los arcos les asignará el valor del vector peso.

c) **Highlight.** Con este comando podemos remarcar nodos o arcos de un grafo no orientado, para dibujarlos en distinto color o tamaño. Por ejemplo, para un grafo no orientado cuya figura se almacenó bajo el nombre de *h*, para remarcar en rojo los vértices *1* y *2* en color rojo, utilizamos la instrucción:

$>> highlight(h, [1\ 2], `NodeColor', `r')$

Si por contra queremos remarcar varios arcos/aristas, indicaremos dos vectores *s* y *t* donde, al igual que hemos hecho en otras ocasiones, indicaremos los vértices inicial y final de los arcos/aristas. Por ejemplo, si en un grafo orientado queremos remarcar en rojo los arcos *1*→*2* y *2*→*3*, usaríamos la instrucción:

$>> highlight(h, [1\ 2], [2\ 3], `EdgeColor', `r')$

Una vez ejecutada la instrucción nos aparecerá una nueva pantalla con el gráfico solicitado.

**Ejemplo 4.5.** *Continuemos con el grafo modificado G5 que hemos construído en el Ejemplo 4.4 y que hemos representado gráficamente en la Figura 24. Si queremos que MatLab lo represente gráficamente, utilizamos la instrucción:*

$>> h = plot(G5)$

*Si por ejemplo queremos que en cada arco aparezca el tiempo de vuelo y además queremos remarcar en rojo las conexiones de ida y vuelta, es decir, los vuelos de ida y vuelta entre los aeropuertos 1 y 6 y entre los aeropuertos 2 y 3, usaremos la siguiente instrucción:*

$>> labeledge(h, 1{:}numedges(G5), peso)$

$>> highlight(h, \{`Aeropuerto\ 1', `Aeropuerto\ 6', `Aeropuerto\ 2', `Aeropuerto\ 3'\},$

$\{`Aeropuerto\ 6', `Aeropuerto\ 1', `Aeropuerto\ 3', `Aeropuerto\ 2'\}, `EdgeColor', 'r')$

*Obtendremos por pantalla la representación gráfica que podemos observar en la Figura 25.*

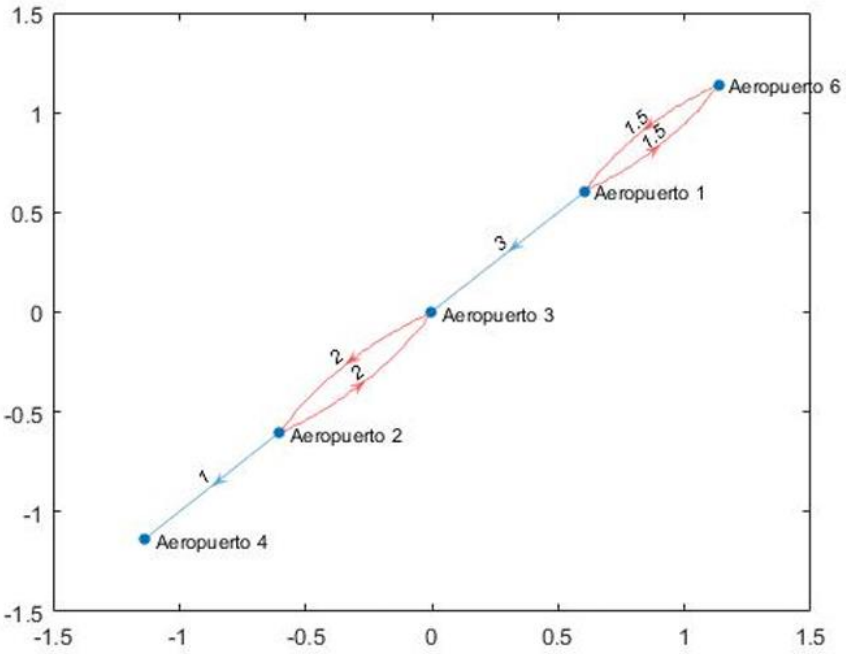



Figura 25 Representación gráfica de MatLab del grafo G5.

### 4.1.4 Información sobre el grafo

Cuando hayamos definido el grafo, podemos solicitar información sobre él. Distinguiremos los casos de grafos orientados y no orientados.

#### 4.1.4.1 Grafos orientados

Comencemos con un grafo orientado *G*. Sobre él podemos calcular los índices de incidencia de entrada y de salida, así como los antecesores y predecesores de cada vértice.

a) **Grado de incidencia de entrada.** Recordemos que, para cada vértice, este número indica el número de arcos que tiene a dicho vértice como vértice final. Para calcularlo, utilizaremos el comando *indegree*:

$>> indegree(G)$

b) **Grado de incidencia de salida.** El grado de incidencia de salida de un vértice dado es el número de arcos que tienen a dicho vértice como vértice de entrada. Para calcularlo utilizaremos la instrucción *outdegree*:

$\gg outdegree(G)$

c) **Predecesores de un vértice.** Los predecesores de un vértice *i* son los vértices *j* de manera que *j*→*i* es un arco. Para calcularlo utilizaremos la instrucción *predecessors*, indicando el nombre del grafo y el vértice sobre el que queremos calcular sus predecesores. Por ejemplo, para calcular los predecesores del vértice 1, usaremos la instrucción:

$>> predecessors(G, 1)$

Si los vértices vienen nombrados como caracteres, llamaremos al vértice usando su nombre

como caracter. Por ejemplo, para calcular los predecesores del vértice `*Nodo 1*':

$>> predecessors(G, 'Nodo\ 1')$

d) **Sucesores de un vértice.** Los sucesores de un vértice *i* son los vértices *j* de manera que *i*→*j* es un arco. Para calcularlo utilizaremos la instrucción *sucessors*, indicando el nombre del grafo y el vértice sobre el que deseamos trabajar. Al igual que en el apartado anterior, para indicar el nodo lo haremos a través de su número o de su nombre caracter, según venga expresado. Por ejemplo, para calcular los sucesores

del vértice 1, usaremos la instrucción:

$>> \ sucessors(G, 1)$

e) **Determinar si hay ciclos.** El grado En muchas ocasiones nos interesará saber si el grafo considerado es acíclico o si, por contra, tiene ciclos. Para ello, simplemente usaremos la instrucción *isdag*, indicando el nombre del grafo:

$>> isdag(G)$

**Ejemplo 4.6.** *Continuemos una vez más con el grafo que hemos modificado en el Ejemplo 4.4. Para calcular los grados de incidencia de entrada y de salida, podemos calcularlos mediante la instrucción:*

$>> \ [indegree(G5), outdegree(G5)]$

$ans =$

1 2

1 2

2 1

1 0

1 1

*Así, obtenemos los grados de incidencia de entrada (primera columna) y de salida (segunda columna) de cada uno de los vértices. En el lenguaje de nuestro ejemplo, los grados de incidencia de entrada y de salida de un aeropuerto i indican el número de aeropuertos desde los cuales se puede volar al aeropuerto i y el número de aeropuertos a los que se puede volar desde el aeropuerto i, respectivamente. Por otra parte, podemos calcular los predecesores y sucesores de cada vértice. En nuestro caso, por ejemplo, para el Aeropuerto 1, los predecesores son los aeropuertos desde los que se puede volar al Aeropuerto 1 mientras que los sucesores son los aeropuertos a los que se puede volar desde el Aeropuerto 1:*

$>> \ predecessors(G5, `Aeropuerto\ 1')$

$ans =$

$`Aeropuerto\ 6'$

$>> \ successors(G5, `Aeropuerto\ 1')$

$ans =$

$`Aeropuerto\ 3'$

$`Aeropuerto\ 6'$

*Concluimos por tanto que al Aeropuerto 1 solamente se puede volar desde el Aeropuerto 6, pero desde el Aeropuerto 1 se puede volar tanto al Aeropuerto 6 como al Aeropuerto 3.*

*Por último, nos interesa saber si en este grafo hay ciclos. Para ello usamos la instrucción:*

$>> \ isdag(G5)$

$ans \ =$

0

*Eso significa que el grafo no es acíclico al tener algún ciclo. Esto no es sorprendente, puesto que ya sabemos que hay un vuelo de ida y vuelta entre los aeropuertos 1 y 6.*

#### 4.1.4.2 Grafos no orientados

En el caso de grafos no orientados, podemos calcular tanto el grado de incidencia como los precedesores/sucesores (que en el caso de grafos no orientados coinciden).

a) **Grado de un vértice.** El grado de un vértice *i* es el número de vértices *j* de manera que *i*→*j* es una arista del grafo. En este caso, calcularemos los grados de los vértices de un grafo *G* mediante la

instrucción *degree*:

$>> degree(G)$

b) **Predecesores/Sucesores de un vértice.** Dado que estamos trabajando con grafos no orientados, los sucesores y predecesores son equivalentes. Para calcularlos, usaremos el comando *neighbors*, indicando el nombre del grafo y el vértice del que queremos calcular los predecesores/sucesores. Al igual que ocurría en el caso de grafos orientados, el vértice se indicará mediante número o carácter, en función de cómo estén definidos. Por ejemplo, para calcular los predecesores/sucesores del vértice 4, usaremos la instrucción:

$>> neighbors(G5,4)$

### 4.1.5 Representación matricial

Como hemos visto en las clases de teoría, un grafo se puede representar de manera matricial de varias formas. Las representaciones matriciales son muy útiles porque nos permiten simplificar los cálculos. Con MATLAB podemos calcular las matrices de adyacencia y de incidencia vértice-arco:

a) **Matriz de adyacencia.** Como vimos al inicio de este capítulo, es la matriz cuadrada que tiene tantas filas y columnas como vértices hay en el grafo, y definida por $a_{ij} = 1$ si $i \rightarrow j$ es un arco o $a_{ij} = 0$ en otro caso. Para calcular la matriz de adyacencia, usaremos el comando *adjacency*:

$>> adjacency(G)$

Usando esta expresión nos indicará los arcos que tiene el grafo. Si queremos que nos muestre la matriz completa, debemos añadir el comando *full*:

$>> full(adjacency(G))$

*b)* **Matriz de incidencia vértice-arco.** Esta matriz solamente se puede calcular para los grafos orientados. Esta matriz tiene tantas columnas como vértices y tantas filas como arcos, y en el elemento $m_{ie}$ toma el valor 1, si *i* es el vértice inicial del arco *e*; $m_{ie}$ toma el valor -1 si *i* es el vértice final del arco *e*, y toma el valor 0 en otro caso. Para calcularla, utilizaremos el comando *incidence*:

$>> incidence(G)$

Al igual que en el apartado anterior, por pantalla nos mostrará una lista con los arcos $(i, j)$ y nos indicará el valor 1 o -1 según el vértice inicial sea *i* o *j*, respectivamente. Una vez más, si queremos la matriz completa debemos utilizar la instrucción *full*:

$>> full(incidence(G))$

**Ejemplo 4.7.** *Continuemos con el grafo del Ejemplo 4.4. Vamos a calcular sus matrices de adyacencia y de incidencia vértice-arco:*

$>> adjacency(G5)$

$ans =$

(5,1) 1

(3,2) 1

(1,3) 1

(2,3) 1

(2,4) 1

(1,5) 1

$>> full(adjacency(G5))$

$ans =$

0 0 1 0 1

0 0 1 1 0

0 1 0 0 0

0 0 1 1 0

1 0 0 0 0

$>> incidence(G5)$

$ans =$

$(1,1) \ -1$

$(3,1) \quad 1$

$(1,2) \ -1$

$(5,2) \quad 1$

$(2,3) \ -1$

$(3,3) \quad 1$

$(2,4) \ -1$

$(4,4) \quad 1$

$(2,5) \quad 1$

$(3,5) \ -1$

$(1,6) \quad 1$

$(5,6) \ -1$

$>> full(incidence(G5))$

$ans =$

$$\begin{matrix} -1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{matrix}$$

*Concluimos que las matrices de adyacencia (A) y de incidencia vértices-arco (M) vienen dadas por:*

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}; \ M = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix}$$

### 4.1.6 Conexión

Una vez que ya hemos explicado cómo definir grafos y hacer operaciones básicas con ellos, podemos estudiar la conexión. Recordemos brevemente el significado de la conexión en grafos orientados y no orientados:

a) **Conexión en grafos no orientados.** Un grafo no orientado es conexo cuando existe una cadena entre cada par de vértices.

b) **Conexión débil en grafos orientados.** Un grafo orientado es débilmente conexo cuando su grafo no

orientado asociado es conexo.

c) **Conexión en grafos orientados.** Un grafo orientado es conexo cuando entre cada par de vértices *i, j* existe un camino de *i* a *j* y otro camino de *j* a *i*.

Cuando un grafo (orientado o no orientado) no es conexo (o débilmente conexo), es interesante conocer cuáles son las componentes conexas del grafo. Una componente conexa es un subgrafo conexo (o débilmente conexo) al que no es posible añadirle un vértice porque perdería esta propiedad.

Con el fin de conocer si un grafo es conexo, y conocer sus componentes conexas, utilizaremos la instrucción *conncomp*. Vamos a detallar su uso:

a) Si *G* es un grafo no orientado, utilizaremos la instrucción anterior en la que únicamente indicaremos el nombre del grafo:

$>> conncomp(G)$

Como resultado, obtendremos un vector con tantos valores como vértices, de manera que para el vértice *i* indica a qué componente conexa pertenece. Si solamente hay una componente conexa, el grafo *G* será conexo.

b) Si *G* es un grafo orientado y queremos estudiar su posible conexión, seguiremos los mismos pasos del apartado anterior.

c) Si *G* es un grafo orientado y queremos ver si es débilmente conexo, esto es, si su grafo no orientado asociado es conexo, usaremos la siguiente instrucción:

$>> conncomp(G, `Type', `weak')$

En este caso, indicamos que el tipo de conexión es *débil* (weak), y por pantalla nos volverá a sacar un vector indicando la componente conexa a la que pertenece cada vértice.

**Ejemplo 4.8.** *En el Ejemplo 4.4 teníamos un grafo orientado en el que teníamos varios aeropuertos e indicábamos las conexiones existentes entre ellos. Pasamos a continuación a estudiar la conexión en este grafo:*

$>> conncomp(G5)$

$ans =$

1 2 2 3 1

*De esta manera obtenemos que hay 3 componentes conexas en este grafo:*

- *Componente conexa 1: Aeropuerto 1, Aeropuerto 6.*
- *Componente conexa 2: Aeropuerto 2, Aeropuerto 3.*
- *Componente conexa 3: Aeropuerto 4.*

*Así vemos que se pueden hacer viajes de ida y vuelta entre los aeropuertos 1 y 6, y entre los aeropuertos 2 y 3. Estudiemos a continuación la conexión débil de este grafo:*

$>> conncomp(G5, 'Type', 'weak')$

$ans =$

1 1 1 1 1

*Esto significa que el grafo es débilmente conexo. El significado es que, si todo vuelo de ida tuviese un vuelo de vuelta, el grafo sería conexo, es decir, se podrían hacer vuelos de ida y vuelta entre cada par de aeropuertos.*

## 4.2 Ejemplo ilustrativo

Escogeremos las calles principales del barrio sevillano de Los Remedios para analizar la correcta interpretación del tráfico.


Figura 26 Distribución del tráfico en los Remedios.

1. Virgen de Luján
2. Virgen de las Montañas
3. Virgen de La Estrella
4. Virgen de Araceli
5. Virgen de Robledo
6. Padre Damián
7. Virgen de Monserrat
8. Virgen de Guaditoca
9. Adolfo Suárez
10. Virgen de la Oliva
11. Virgen del Águila
12. Virgen de la Antigua
13. Juan Ramón Jiménez
14. Virgen de la Cinta.

Hemos escogidos estas calles porque según Tráfico son las más conflictivas.

Vamos a crear matriz de incidencia dirigida. La notación ha de entenderse como *A(i,j)* refiriéndose a la conexión existente o no entre los vértices *i* y *j* en el sentido *i→j*. De esta forma, por usar un primer ejemplo, el primer término definido a continuación nos permite representar y notar que desde la calle Virgen de Luján (1) puede accederse directamente a la calle Virgen de Las Montañas (2):

$A(1,2) = 1; A(2,1) = 0;$

$A(1,3) = A(3,1) = 0;$

$A(1,4) = A(4,1) = 0;$

$A(1,5) = A(5,1) = 0;$

$A(1,6) = 0;\ A(6,1) = 1;$

$A(1,7) = 1;\ A(7,1) = 0;$

$A(1,8) = A(8,1) = 0;$

$A(1,9) = 1;\ A(9,1) = 0;$

$A(1,10) = 1;\ A(10,1) = 1;$

$A(1,11) = 0;\ A(11,1) = 1;$

$A(1,12) = A(12,1) = 0;$

$A(1,13) = 1;\ A(13,1) = 0;$

$A(1,14) = A(14,1) = 0;$

Hasta aquí, se han establecido las conexiones de la calle Virgen de Luján (1) con el resto de calles del barrio, en ambos sentidos.

La calle Virgen de las Montañas (2) sólo conecta con la calle Virgen de Luján (1), por lo que se define:

$A(2,k) = A(k,2) = 0;\ k = 3,..,14$

La calle Virgen de la Estrella (3) conecta con la calle Padre Damián (6) y Juan Ramón Jiménez (13)

$A(3,6) = 1;$

$A(3,13) = 1;$

$A(13,4) = A(4,6) = 1;\ resto\ cero.$

Desde Virgen de Araceli (4) sólo se puede acceder a Virgen de Robledo (5) (sólo se tienen en cuenta las calles que se han incluido en el estudio del grafo, por eso no se declara el acceso a/desde Asunción y Elcano):

$A(4,5) = 1;\ resto\ cero.$

Desde calle Adolfo Suárez (9) se puede acceder a Padre Damián (6), y desde Padre Damián se puede acceder a Virgen de la Antigua (12) y Virgen de la Cinta (14):

$A(9,6) = A(6,12) = A(6,14) = 1;\ resto\ cero$

Desde Virgen de Montserrat (7) se puede acceder a Virgen de Guaditoca (8) y Adolfo Suárez (9). Asimismo, desde Virgen de Guaditoca también se puede acceder a Adolfo Suárez:

$A(7,8) = A(7,9) = 1;$

$A(8,9) = 1;$

La calle Adolfo Suárez es accesible desde la calle Virgen de la Oliva (10), y también da acceso a Virgen del Águila (11). De igual manera, la calle Virgen de la Oliva da acceso a Virgen del Águila, y es accesible desde la calle Virgen de la Cinta (14). Desde Virgen del Águila se puede acceder a Virgen de la Antigua (12) y a Virgen de la Cinta (14).

$A(10,9) = A(9,11) = 1;$

$A(10,11) = A(14,10) = 1;$

$A(11,12) = A(11,14) = 1;$

La calle Juan Ramón Jiménez (13) da acceso a Virgen de la Antigua (12) y Virgen de la Cinta (14). Desde Virgen de la Antigua puede accederse a Virgen de la Cinta (incorrecto).

$A(13,12) = 1;$

$A(12,14) = 1;$

$A(13,14) = 1;$

Al ejecutar las siguientes órdenes de Matlab obtenemos las respuestas que se muestran:

$>>\ conncomp(G,'Type','weak')$

ans =

1 1 1 1 1 1 1 1 1 1 1 1 1 1

$>> conncomp(G)$

ans =

2 4 1 2 3 2 2 2 2 2 2 2 2 2

En estas respuestas se nos indica que el grafo es conexo (débilmente) pero que hay 4 componentes (4 subgrafos conexos maximales; recordatorio: significa que no están contenidos en ningún subgrafo conexo). En particular, si no existiesen los cruces segundo, tercero y quinto, no habría problemas de circulación para llegar a todos los puntos.

Vamos a calcular los predecesores y sucesores de dichos vértices para analizar la situación:

- Vértice 2 (Virgen de las Montañas): veamos sus sucesores y predecesores.

  Resulta:

  $>> successors(G, 2)$

  $ans =$

  $0 \times 1\ empty\ double\ column\ vector$

  $>> predecessors(G, 2)$

  $ans =$

  1

  Esta respuesta nos hace saber que desde Virgen de las Montañas no puede accederse a ninguna otra calle, y a ella sólo puede accederse desde Virgen de Luján (1). Luego hay una calle que nos lleva a la calle en cuestión. Por tanto, es claro que debemos dar a la calle Virgen de las Montañas (2) doble sentido. Es decir:

  $A(2,1) = 1$

- Vértice 3 (Virgen de la Estrella): Hacemos lo mismo. Esta calle tiene dos sucesores, Padre Damián (6) y Juan Ramón Jiménez (13), accesibles desde Virgen de la Estrella. Para hacer accesible esta última, Nos basta hacer que Padre Damián sea accesible desde Juan Ramón Jiménez y Virgen de la Estrella para asegurarnos una buena circulación, aunque bastaría con uno de los dos:

  $A(3,6) = A(13,6) = 1$

- Vértice 5 (Virgen de Robledo): tiene un sucesor que es el nodo correspondiente a Virgen de Araceli (4) y ningún predecesor, por lo que deberemos darle doble sentido estableciendo que:

  $A(5,4) = 1;$

Añadimos las direcciones correspondientes. Tenemos:

$\gg conncomp(G)$

$ans =$

1 1 1 1 1 1 1 1 1 1 1 1 1 1

Dado que en este ejemplo cada arco no tiene asociado un peso, con la matriz de adyacencia podemos definir el grafo en MATLAB:

$>> G = digraph(A)$

$G =$

$digraph\ with\ properties$:

$Edges$: $[18x2\ table]$

$Nodes$: $[13x0\ table]$

Nos indica que tenemos 13 vértices (cruces) y 18 arcos (direcciones). Este grafo se puede representar gráficamente con la instrucción:

$>>\ h = plot(G)$

Fruto de este modelaje, se obtiene de MATLAB la siguiente representación del grafo en la Figura 27.

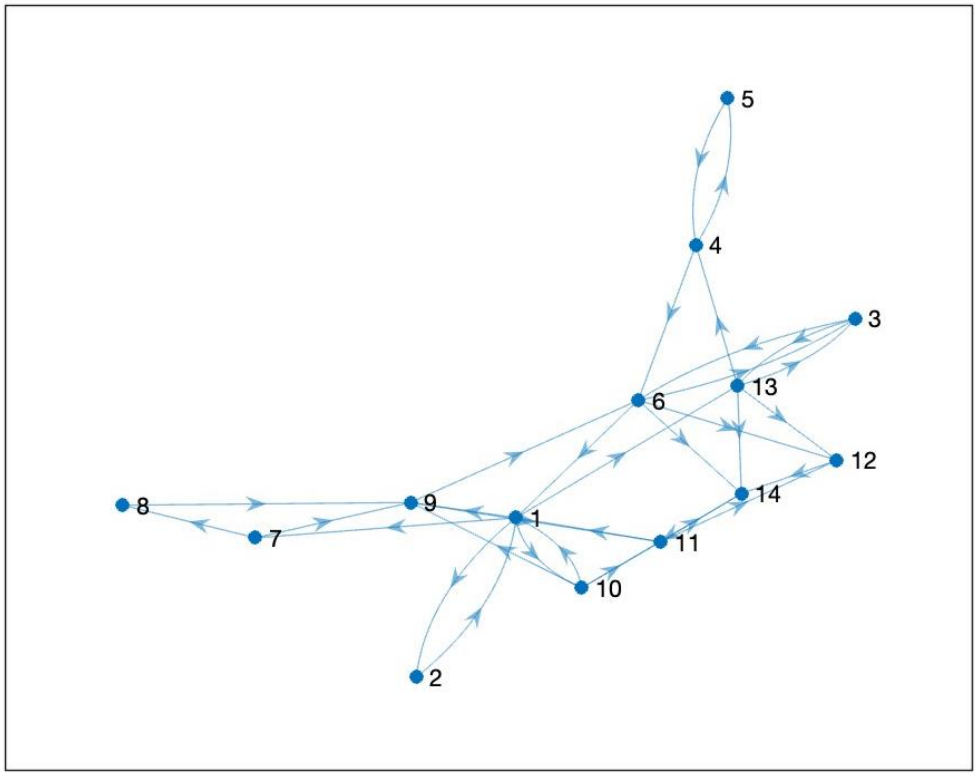



Figura 27 Representación gráfica de MATLAB del grafo G correspondiente a la planificación de tráfico que resultaría en la mejor conexión del barrio de Los Remedios.

# 5 APLICACIÓN 2ª: ÁRBOLES MAXIMALES DE MÍNIMO COSTO

En este capítulo vamos a ver cómo podemos calcular el **árbol maximal de mínimo costo (AMMC)** utilizando MATLAB. Recordemos que un árbol es un grafo no orientado, conexo y acíclico. Partiremos de un grafo no orientado $G = (V, A)$ y unos pesos $p$ de manera que cada arista $e$ tiene asociado un peso $p_e$, y el objetivo será buscar un subgrafo de $G$ que sea un árbol y que minimice el valor total de los pesos.

## 5.1 Resolución con MATLAB

### 5.1.1 El comando *minspantree*

El comando con el que resolveremos el problema de encontrar un AMMC es *minspantree* (del inglés, *Minimum spanning tree*). De esta manera, si hemos definido un grafo no orientado *G*, simplemente hemos de indicar lo siguiente:

$>> \ T = minspantree(G)$

Es conveniente almacenar el resultado, en este caso le asignamos el nombre $T$, puesto que la salida de esta instrucción es un nuevo grafo, de hecho, un subgrafo de $G$, que solamente contiene los arcos que minimizan el valor total del árbol. Una vez que ya tenemos el árbol $T$, sobre él podemos obtener toda la información que deseemos, al igual que hemos hecho en la aplicación 1. Por ejemplo, si queremos conocer qué aristas se han incluido en este subgrafo, utilizamos la instrucción:

$>> \ T.Edges$

Si el grafo original lo hemos definido indicando las aristas con los vectores *s* y *t*, y el vector *peso* para indicar los pesos, podemos hacer una representación gráfica del grafo y su AMMC:

$>> \ G = graph(s, t, peso)$

$>> \ h = plot(G)$

$>> \ labeledge(h, 1{:}\, numberedges(G), peso)$

$>> \ T = minspantree(G)$

$>> \ highlight(h, T, `EdgeColor', `red')$

Con la tercera instrucción añadimos los pesos a las aristas, mientras que con la quinta nos marcará las aristas del árbol en color rojo.

En ocasiones, podemos estar interesados en crear un AMMC partiendo de un vértice dado. En tal caso, dicho vértice se indicará como raíz. Si por ejemplo el vértice elegido es el tercero, la instrucción a utilizar será:

$>> \ T = minspantree(G, `Root', 3)$

Por defecto, salvo que se indique el vértice inicial, se tomará como raíz al primer vértice.

Hemos de notar que, si el grafo introducido no es conexo, el comando *minspantree* solamente calcula el AMMC asociado a la componente conexa a la que pertenece su raíz (que como hemos comentado es el primer vértice, salvo que se indique otra cosa). Cuando el grafo original no es conexo, una posibilidad es calcular el bosque de menor valor. En tal caso, MATLAB calculará el AMMC para cada una de las componentes conexas. Para

calcular el bosque en vez del árbol, podemos utilizar la siguiente instrucción:

$>> T = minspantree(G,'Type','forest')$

**Ejemplo 5.1.** *Consideremos el grafo no orientado que aparece en la Figura 28.*

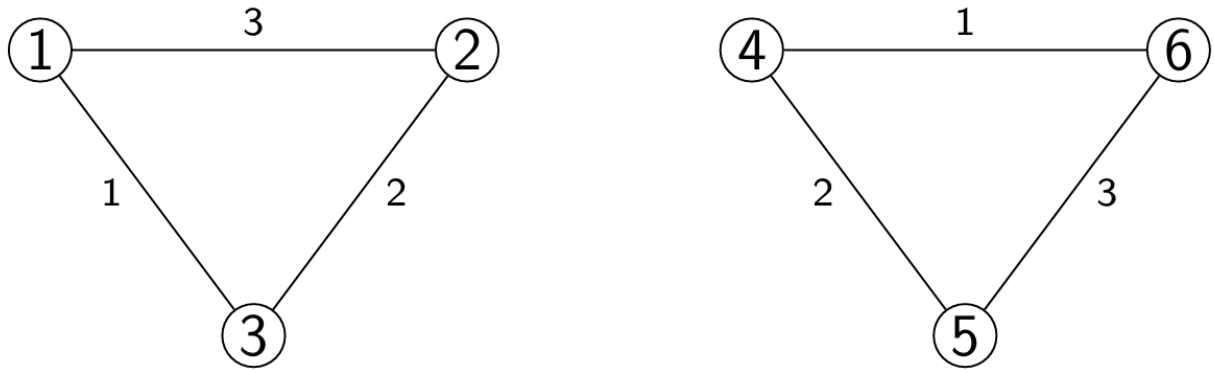



Figura 28 Grafo del Ejemplo 5.1.

*Definamos este grafo en MATLAB:*

$>> s = [1\ 1\ 2\ 4\ 4\ 5];$

$>> t = [2\ 3\ 3\ 5\ 6\ 6];$

$>> peso = [3\ 1\ 2\ 1\ 2\ 3];$

$>> G = graph(s,t,peso)$

$G =$

$graph\ with\ properties:$

$Edges: [6x2\ table]$

$Nodes: [6x0\ table]$

*Obviamente, este grafo no es conexo, como podemos comprobar:*

$>> conncomp(G)$

$ans =$

1 1 1 2 2 2

*Si utilizamos el comando minspantree sin especificar la raíz, tomará como tal al vértice 1, obteniendo el siguiente resultado:*

$>> T1 = minspantree(G)$

$T1 =$

$graph\ with\ properties:$

$Edges: [2x2\ table]$

$Nodes: [6x0\ table]$

$>> T1.Edges$

$ans =$

$EndNodes\ \ Weight$

_____ _____

1 3 1

2 3 2

*Como vemos, el árbol creado solamente hace referencia a los vértices de la primera componente conexa, formada por los vértices 1, 2 y 3, dejando a los vértices 4, 5 y 6 como vértices aislados. Por contra, si ahora buscamos el AMMC indicando como raíz al vértice 4, obtenemos lo siguiente:*

$>> T2 = minspantree(G, `¡'Root', 4)$

$T2 =$

$graph\ with\ properties:$

$Edges: [2x2\ table]$

$Nodes: [6x0\ table]$

$>> T2.Edges$

$ans =$

| EndNodes | | Weight |
|---|---|---|
| 4 | 5 | 1 |
| 4 | 6 | 2 |

*En este caso, el árbol solamente hace referencia a los vértices de la segunda componente conexa (vértices 4, 5 y 6). En este segundo caso, los vértices 1, 2 y 3 han quedado aislados. Si por contra, en vez de solicitar un árbol, solicitamos el bosque, el resultado es el siguiente:*

$>> F = minspantree(G, `Type', 'forest')$

$F =$

$graph\ with\ properties:$

$Edges: [4x2\ table]$

$Nodes: [6x0\ table]$

$>> F.Edges$

$ans =$

| EndNodes | | Weight |
|---|---|---|
| 1 | 3 | 1 |
| 2 | 3 | 2 |
| 4 | 5 | 1 |
| 4 | 6 | 2 |

*En este último caso, nos ha calculado el AMMC en cada una de las componentes, y juntos forman el bosque de unión de valor mínimo. En la Figura 29 podemos ver la representación de los grafos $T_1$, $T_2$ y $F$ obtenidos en este ejemplo.*

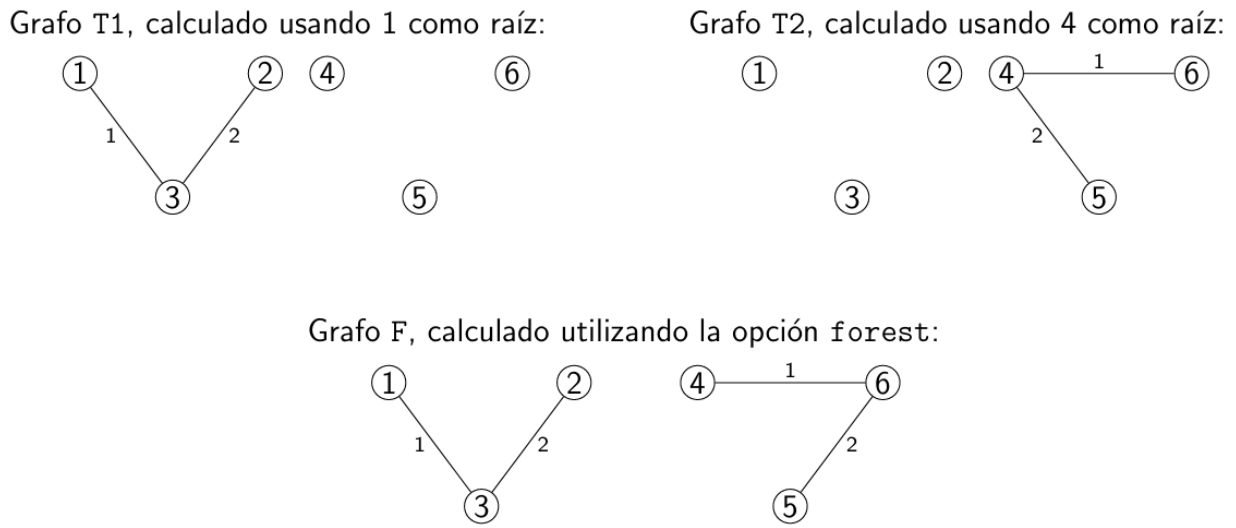



Figura 29 Grafos T1 and T2 obtenidos utilizando el comando *minspantree* con los nodos 1 y 4 como raíces, respectivamente (figura de arriba a la izquierda y derecha, respectivamente). En la figura inferior se muestra el bosque de unión de valor mínimo.

### 5.1.2 Opciones del comando *minspantree*

En el apartado anterior ya hemos visto algunas de las opciones que tiene el comando *minspantree*, como son las siguientes:

a) **Root**. Nos permite especificar la raíz del árbol.

b) **Forest**. Para grafos inconexos, calcula el AMMC en cada componente conexa y forma con ellos un bosque.

Otras opciones interesantes son las siguientes:

a) **Tipo de algoritmo.** Por defecto, el comando *minspantree* utiliza el algoritmo de Prim, que comienza en la raíz y va añadiendo aristas mientras recorre el grafo. Si queremos especificar el tipo de algoritmo, debemos de utilizar la opción *dense*, para el algoritmo de Prim, y *sparse*, para el algoritmo de Kruskal, que ordena las aristas según su peso y a continuación añade los vértices. Por ejemplo, para especificar el algoritmo de Kruskal, debemos utilizar la expresión:

$>> \ T = minspantree(G, `Method', `sparse')$

b) **Lista de predecesores.** Además de obtener el AMMC como salida, también podemos solicitar un segundo vector que contiene los predecesores de cada nodo, comenzando a partir de la raíz. En tal caso, la raíz no tiene predecesor, y por tanto le asigna el valor 0.

$>> \ [T, pred] = minspantree(G)$

**Ejemplo 5.2.** *Continuando con el grafo del Ejemplo 5.1 representado en la Figura 29, podemos volver a solicitar el bosque de unión de valor mínimo añadiendo también la lista de predecesores:*

$>> \ [F, pred] = minspantree(G, `Type', `forest')$

$F \ =$

$graph \ with \ properties$:

$Edges$: $[4x2 \ table]$

$Nodes$: $[6x0 \ table]$

$pred \ =$

0 3 1 0 4 4

*El significado del vector pred es el siguiente: los vértices 1 y 4 tienen asignado el valor 0, puesto que ambos han sido tomados como raíces (cada uno de su componente conexa); el predecesor del vértice 2 es el 3, y el predecesor del vértice 3 es el 1; el vértice 4 es el predecesor de los vértices 5 y 6.*

## 5.2 Ejemplo ilustrativo

Consideremos un ejemplo de un conjunto de pueblos del Aljarafe con los pesos que son función de las distancias y la calidad de la carretera.

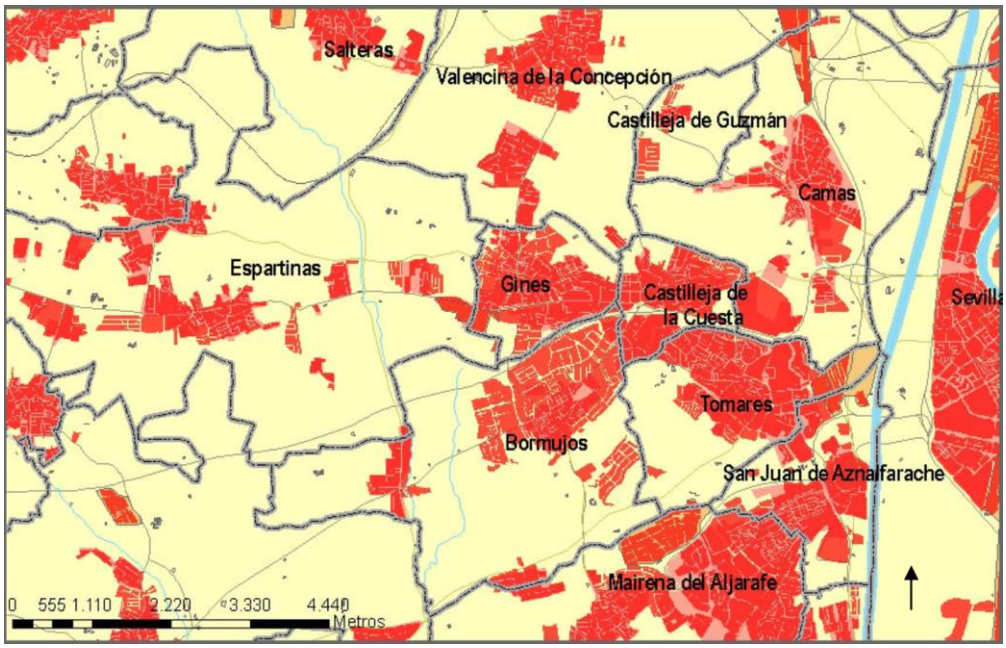



Figura 30 Sistema de carreteras el aljarafe.

El objetivo es, por tanto, minimizar la longitud de la línea, lo que reducirá costes tanto de instalación como de mantenimiento. En nuestros términos, lo que hemos de hacer es encontrar un AMMC.

Los datos son:

(1) Mairena → (2) San Juan: 0,48

(2) San Juan → (3) Tomares: 0,86

(5) Castilleja de la Cuesta → (6) Camas: 0,45

(1) Mairena → (3) Tomares: 0,53

(3) Tomares → (4) Bormujos: 0,5

(3) Tomares → (6) Camas: 0,78

(1) Mairena → (4) Bormujos: 0,56

(3) Tomares → (5) Castilleja de la Cuesta: 0,45

(6) Camas → (7) Castilleja de Guzmán: 0,51

(8) Valencina → (6) Camas: 0,73

(8) Valencina → (7) Castilleja de Guzmán: 0,35

(8) Valencina → (9) Gines: 0,55

(9) Gines → (5) Castilleja de la Cuesta: 0,61

(9) Gines → (4) Bormujos: 0,5

(9) Gines → (6) Camas: 1,09

(9) Gines → (10) Espartinas: 0,79

(11) Salteras → (8) Valencina: 0,35

(10) Espartinas → (11) Salteras: 0,35

Comencemos definiendo el grafo:

$>> s = [1\ 2\ 5\ 1\ 3\ 3\ 1\ 3\ 6\ 8\ 8\ 8\ 9\ 9\ 9\ 9\ 11\ 10];$

$>> t = [2\ 3\ 6\ 3\ 4\ 6\ 4\ 5\ 7\ 6\ 7\ 9\ 5\ 4\ 6\ 10\ 8\ 11];$

$\gg$ peso $= [0.48\ 0.86\ 0.45\ 0.53\ 0.5\ 0.78\ 0.56\ 0.45\ 0.51\ 0.73\ 0.35\ 0.55\ 0.61\ 0.5\ 1.09\ 0.79\ 0.35\ 0.35];$

$>>$ G = graph(s, t, peso, {'O', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'T'})

G =

graph with properties:

Edges: [18 × 2 table]

Nodes: [11 × 1 table]

$>>$ h = plot(G)

$>>$ labeledge(h, 1: numedges(G), peso)

De esta manera hemos definido el grafo y hemos obtenido la representación gráfica de la Figura 31.

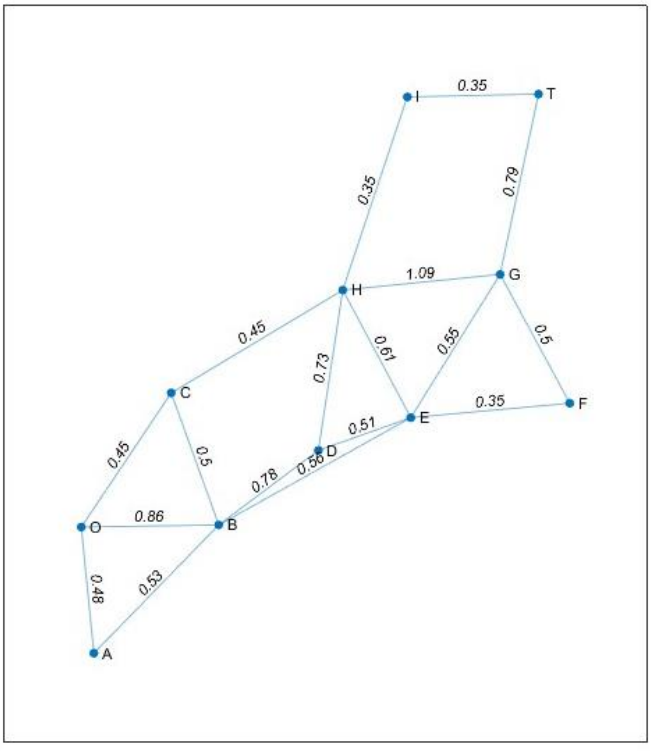



Figura 31 Representación gráfica de MatLab sistema de carreteras entre Pueblos del Aljarafe.

Se busca a continuación el AMMC. Una vez definido el grafo, el uso del comando *minspantree* permite obtener el árbol maximal de mínimo costo, que corresponde con el árbol de menor peso dentro del grafo definido.

$>> T = minspantree(G)$

$T =$

$graph\ with\ properties$:

$Edges$: [$10 \times 2\ table$]

$Nodes$: [$11 \times 1\ table$]

A continuación, mediante el comando *T.Edges*, se solicita la información referente a las aristas que ha seleccionado para formar parte del árbol:

$>>\ T.Edges$

$ans\ =$

$10 \times 2\ table$

| $EndNodes$ | | $Weight$ |
|---|---|---|
| $\{'O'\}$ | $\{'A'\}$ | 0.48 |
| $\{'O'\}$ | $\{'B'\}$ | 0.53 |
| $\{'B'\}$ | $\{'C'\}$ | 0.5 |
| $\{'B'\}$ | $\{'D'\}$ | 0.45 |
| $\{'C'\}$ | $\{'H'\}$ | 0.5 |
| $\{'D'\}$ | $\{'E'\}$ | 0.45 |
| $\{'E'\}$ | $\{'F'\}$ | 0.51 |
| $\{'F'\}$ | $\{'G'\}$ | 0.35 |
| $\{'G'\}$ | $\{'T'\}$ | 0.35 |
| $\{'I'\}$ | $\{'T'\}$ | 0.35 |

También puede solicitarse la matriz de adyacencia mediante el comando *adjacency(T)* o la matriz completa, mediante *full(adjacency(T))*. Si se quiere representar gráficamente el AMMC, pueden utilizar la siguiente instrucción, mediante la cual se destaca el mismo en rojo en la Figura 32:

$>>\ highlight(h,T,'EdgeColor','red')$

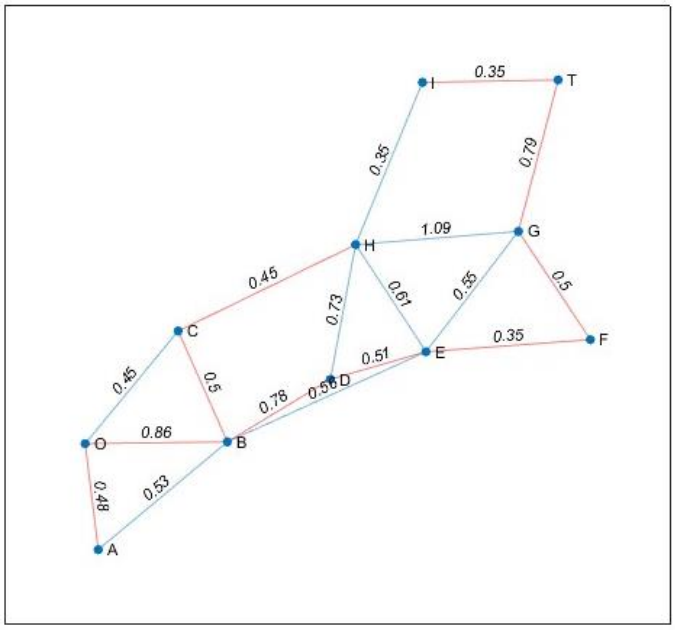



Figura 32 Árbol de unión de menor valor para el sistema de carreteras del Aljarafe.

# 6 APLICACIÓN 3ª: CAMINOS Y CADENAS DE MENOR VALOR

En la primera aplicación hemos dado los primeros pasos en MATLAB en lo que a la Teoría de Grafos se refiere. En particular, hemos visto cómo definir grafos, añadir y/o modificar arcos y/o vértices, cómo hacer representaciones gráficas y cómo analizar la conexión. En esta aplicación vamos a ver cómo utilizar MATLAB para resolver el problema de **la ruta más corta o camino de menor valor (CMV)**.

Recordemos la formulación del problema de la ruta más corta: comenzamos con una red $R=(V,A,p)$ conexa, es decir, un grafo conexo con vértices $V$ y arcos $A$, además de un vector de pesos $p$ que asigna a cada arco $e$ un peso $p_e$, de manera que todo vértice es alcanzable desde cualquier otro vértice. El objetivo es, dado un par de vértices $i$ y $j$, encontrar el camino/cadena que va desde $i$ a $j$ de manera que tenga asignado un menor valor: si el camino está formado por los arcos $e_1, \dots, e_k$, se trata de minimizar el valor del camino $p_{e_1} + \cdots + p_{e_k}$. Como hemos visto en el primer capítulo de teoría, hay varios algoritmos que nos permiten resolver este problema:

- Si todos los pesos son no negativos, el mejor algoritmo es el de Dijkstra.
- Si hay algún peso negativo, el problema tendrá solución si no hay ciclos de valor negativo. En tal caso, hemos de utilizar el algoritmo de Floyd.

Hemos de notar que, si el grafo es no orientado, el CMV existe solamente si los pesos son no-negativos. Por lo tanto, en tal caso se utilizaría el algoritmo de Dijkstra.

A continuación pasamos a explicar cómo resolver el problema de la ruta más corta utilizando MATLAB.

## 6.1 Resolución con MATLAB

### 6.1.1 El comando *shortestpath*

Partimos de un grafo $G$ que podemos definir como hemos visto en la primera aplicación. Este grafo tiene asociado un vector de pesos que asigna un valor a cada arco/arista. La orden con la que MATLAB nos permite resolver el problema del CMV es *shortestpath*. En esta orden hemos de especificar tres valores: el nombre que le hemos dado al grafo y dos vectores, $s$ y $t$, de manera que obtendremos el CMV desde el vértice $s(i)$ hasta el vértice $t(i)$.

$>> shortestpath(G, s, t)$

Obtendremos por pantalla los vértices que forman el camino de menor valor buscado.

**Ejemplo 6.1.** *Consideremos el grafo representado gráficamente en la Figura 33. En este grafo, que tiene seis vértices y diez arcos, nos interesa encontrar un camino de menor valor desde el primer vértice hasta el vértice 6.*

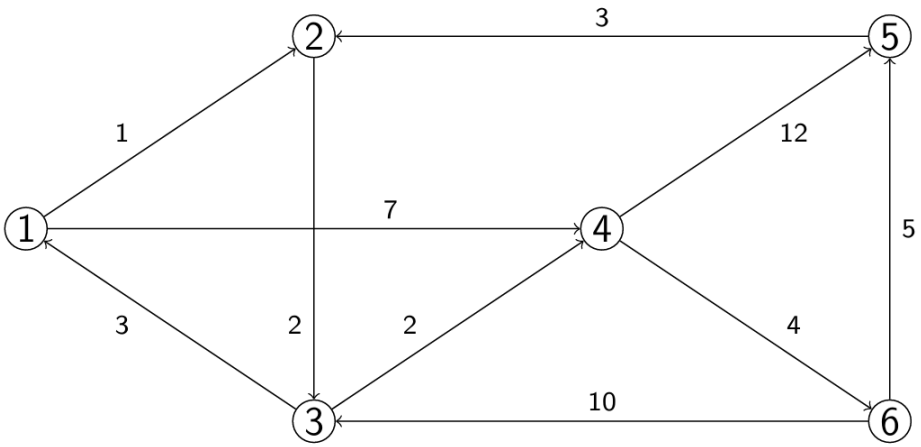



Figura 33 Grafo orientado con pesos.

*Para definir el grafo, se ordenan las siguientes instrucciones:*

$>> \ s = [1\ 1\ 2\ 3\ 3\ 4\ 4\ 5\ 6\ 6];$

$>> \ t = [2\ 4\ 3\ 1\ 4\ 5\ 6\ 2\ 3\ 5];$

$>> \ peso = [1\ 7\ 2\ 3\ 2\ 12\ 4\ 3\ 10\ 5];$

$>> \ G = digraph(s, t, peso)$

$G \ =$

$digraph\ with\ properties$:

$Edges$: $[10x2\ table]$

$Nodes$: $[6x0\ table]$

*La representación gráfica del grafo definido, incluyendo los pesos, se obtiene mediante la expresión vista en la aplicación:*

$>> \ h = plot(G, `Layout', `layered')$

$>> \ labeledge(h, 1{:}\, numedges(G), peso)$

*De esta manera se obtiene la representación gráfica de la Figura 34.*

*A continuación, se resuelve el problema del CMV:*

$>> \ CMV = shortestpath(G, 1{,}6)$

$CMV \ =$

1 2 3 4 6

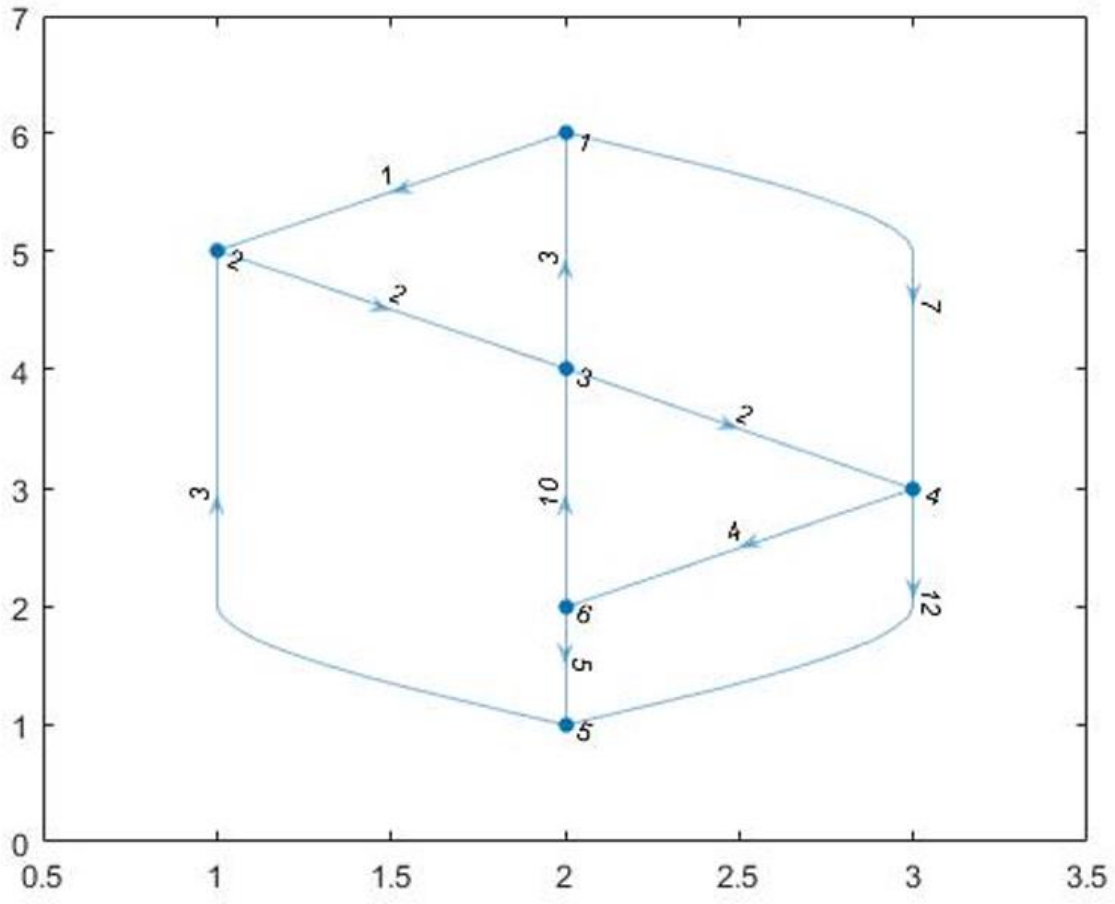



Figura 34 Representación gráfica de MATLAB del grafo definido en la figura anterior.

*El CMV por tanto es 1 → 2 → 3 → 4 → 6, con un valor de 1 + 2 + 2 + 4 = 9. En la Figura 37 tenemos la representación gráfica del CMV entre los vértices solicitados realizada por MATLAB utilizando las siguientes instrucciones:*

$>> highlight(h, CMV, 'EdgeColor', 'r')$

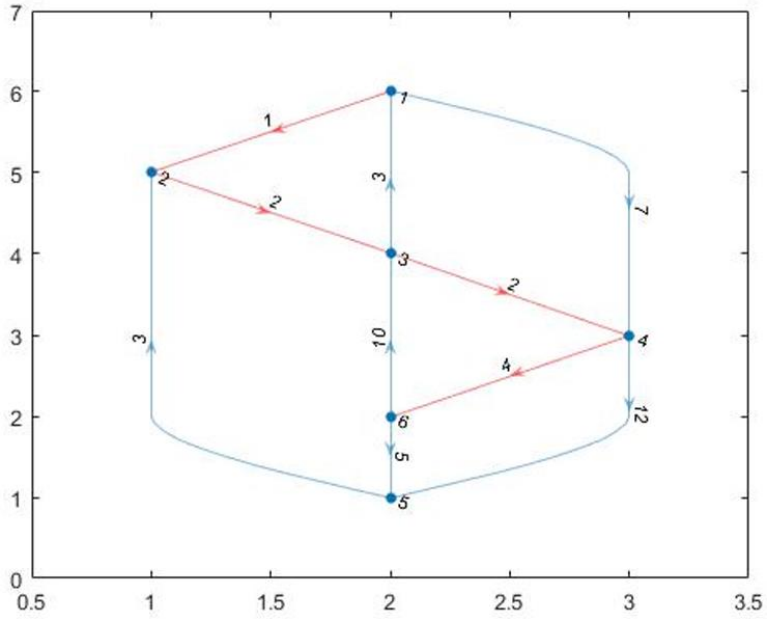



Figura 35 Representación gráfica de MATLAB del grafo definido en la figura anterior con el camino de menor

valor (dibujado en rojo).

*Utilizando la representación inicial de la Figura 33, el camino de menor valor se representa en la figura que sigue:*

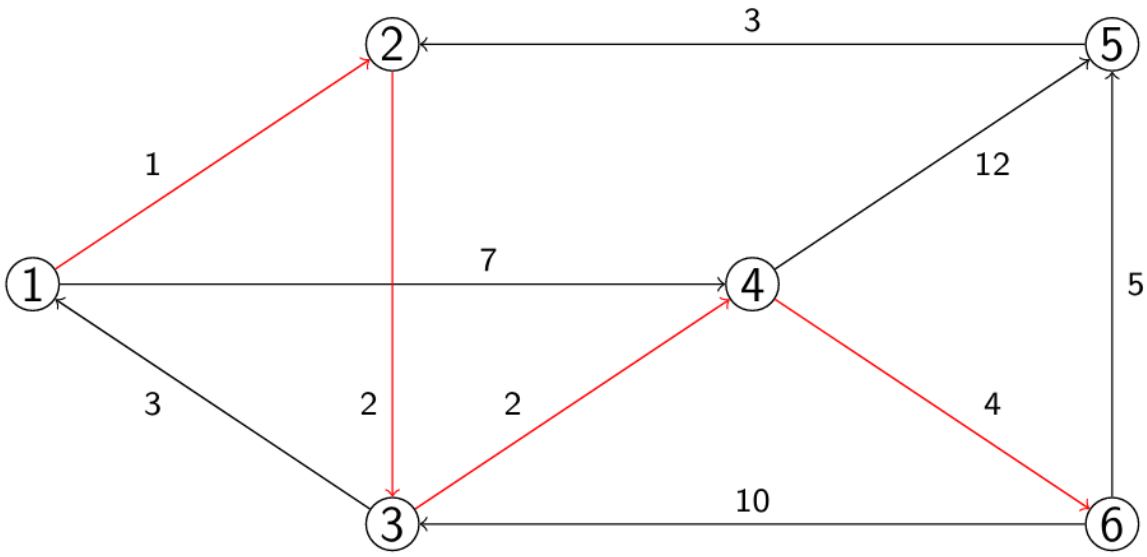



Figura 36 Grafo orientado con pesos

Como se ha comentado, si el grafo *G* es no orientado, el CMV existirá si los pesos tienen valores no negativos. En caso de utilizar el comando *shortestpath* con un grafo no orientado con pesos negativos, se obtendrá el siguiente error:

*Error using graph/shortestpath*

*Graph edge weights must be nonnegative*

En dicho mensaje, ya se advierte que no es posible utilizar el comando *shortestpath* al haber aristas de valor negativo. De la misma manera, si el grafo es orientado y hay pesos negativos, el CMV existirá si no hay ciclos de valor negativo. En caso de que sí haya ciclos de valor negativo, MATLAB también devolverá un mensaje de error advirtiendo del ciclo de valor negativo.

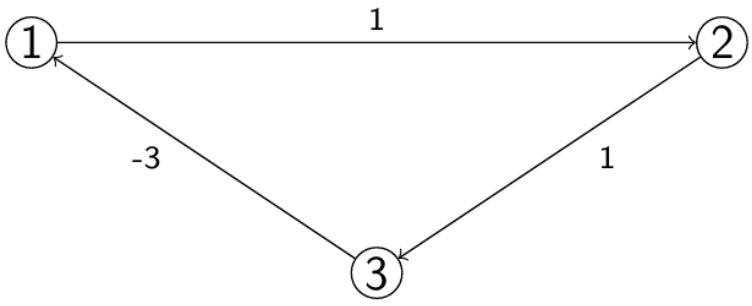



Figura 37 Grafo orientado con un ciclo de valor negativo.

**Ejemplo 6.2.** *Consideremos el grafo definido en la Figura 37. Dicho grafo puede ser definido de la siguiente manera:*

$>> A = [0\ 1\ 0; 0\ 0\ 1; -3\ 0\ 0];$

$>> G = digraph(A)$

$G =$

$digraph\ with\ properties$:

$Edges$: [$3x2\ table$]

$Nodes$: [$3x0\ table$]

*Ahora, si se quisiera calcular un CMV desde el vértice 1 al vértice 3, se ha de emplear el comando a continuación y se obtendría la siguiente salida:*

$>>\ shortestpath(G, 1,3)$

$Error\ using\ digraph/shortestpath$

$Shortestpath\ distance\ undefined\ because\ the\ graph\ contains\ a\ negative\ cycle\ of$

$length\ -1$ ($nodes$ 1 2 3 1).

*Inmediatamente, MATLAB identifica un ciclo de valor negativo: 1 → 2 → 3 → 1, con un valor de 1 + 1 - 3 = -1. Por lo tanto, no existe dicho CMV.*

Si el grafo introducido no es conexo, MATLAB calcula el CMV siempre que los dos vértices introducidos estén en la misma componente conexa.

### 6.1.2 Opciones del comando *shortestpath*

El comando *shortestpath* tiene algunas opciones muy interesantes que se detallan a continuación:

a) **Longitud del CMV.** Además de obtener como salida del comando *shortestpath* el CMV, también puede solicitarse la longitud de dicho camino. Para ello simplemente han de utilizarse las siguientes instrucciones:

   $>>\ [CMV, d] = shortestpath(G, s, t)$

   Bajo el nombre de CMV guardará un vector con los vértices que forman el CMV. Además, con el nombre *d* almacenará la longitud de dicho camino, que no es más que la longitud de CMV menos uno.

b) **Comando *shortestpath* por defecto.** El comando *shortestpath* tiene algunas opciones adicionales. Por defecto, si solamente lo utilizamos como hemos explicado en el apartado 6.1.1, ocurrirá lo siguiente:

   a. Si los pesos son positivos, utilizará el algoritmo de Dijkstra.

   b. Si hay pesos negativos, utilizará el algoritmo de Bellman-Ford. Evidentemente, este caso solamente se puede utilizar para grafos no orientados.

c) **Algoritmo de Dijkstra.** Si queremos utilizar el algoritmo de Dijkstra, tendremos que utilizar el comando *shortestpath* con las siguientes opciones:

   $>>\ shortestpath(G, s, t, 'Method', 'positive')$

   De esta manera forzamos a MATLAB a que utilice dicho algoritmo. Ahora bien, hemos de tener en cuenta que dicho algoritmo solamente funciona cuando los pesos son no negativos. Si forzamos a MATLAB a utilizar el algoritmo de Dijkstra en un grafo con pesos negativos, nos dará el siguiente error:

$Error\ using\ digraph/shortestpath$

$Method\ value\ can\ be\ 'positive'\ only\ if\ all\ edge\ weights\ are\ nonnegative.$

d) **Algoritmo de Bellman-Ford.** Al igual que en el apartado anterior, también podemos forzar a MATLAB a que utilice el algoritmo de Bellmand-Ford. En tal caso, la instrucción a utilizar sería:

$>>\ shortestpath(G,s,t,'Method','mixed')$

Esta opción la podremos utilizar siempre que el grafo sea orientado. Sin embargo, en el caso en que los pesos sean no negativos, ya sabemos que el algoritmo de Dijkstra sería más eficiente.

e) **Grafos acíclicos.** Para grafos acíclicos, hay un algoritmo más eficiente que los de Dijkstra y Bellman-Ford. Para utilizar este algoritmo, utilizaremos el comando:

$>>\ shortestpath(G,s,t,'Method','acyclic')$

Recordemos que, como vimos en la primera práctica, el comando *isdag* nos permite saber si un grafo es acíclico. Por lo tanto, lo correcto sería primero ver si el grafo es acíclico con la instrucción *isdag* y, en caso de que así sea, utilizar este método que será más eficiente que la opción que viene por defecto."

### 6.1.3 Problema de la ruta más corta

Si en un grafo lo que buscamos es simplemente el camino más corto entre dos vértices, en el sentido de tener que utilizar el menor número posible de arcos, tenemos dos opciones:

a) La primera opción es asignar pesos iguales a 1 a todos los vértices. De esta manera, el valor del camino será justamente el número de arcos utilizados.

b) Otra opción sería utilizar el comando shortestpath indicando como método la opción unweighted:

$>>\ shortestpath(G,s,t,`Method',`unweighted')$

## 6.2 Ejemplo ilustrativo

Consideraremos el siguiente ejemplo descrito basado en la bahía de Cádiz que está formada por varios núcleos de población importantes:

| NÚCLEO URBANO | IDENTIFICADOR |
|---|---|
| **CADIZ** | 1 |
| **SAN FERNANDO** | 2 |
| **CHICLANA** | 3 |
| **CONIL** | 4 |
| **CHIPIONA** | 5 |

| | |
|---|---|
| **SANLUCAR** | 6 |
| **PUERTO REAL** | 7 |
| **PUERTO DE SANTA MARIA** | 8 |
| **BARRIO JARANA** | 9 |

A continuación, se analizan los caminos de menor valor desde Cádiz a cada uno de los otros ocho núcleos principales. Antes de nada, ha de definirse el grafo en MATLAB:

$>> \ s = [1\ 1\ 1\ 1\ 2\ 2\ 3\ 3\ 4\ 5\ 5\ 6\ 7\ 7\ 7\ 8\ 8\ 8\ 9\ 9];$

$>> \ t = [2\ 7\ 8\ 9\ 1\ 3\ 2\ 4\ 3\ 6\ 8\ 5\ 1\ 8\ 9\ 1\ 5\ 7\ 1\ 7];$

$>> \ peso = [20\ 20\ 30\ 30\ 20\ 15\ 15\ 25\ 25\ 50\ 19\ 50\ 79\ 20\ 10\ 20\ 35\ 20\ 30\ 10];$

$\gg \ G = graph(s, t, peso)$

$graph\ with\ properties:$

$Edges: [18 \times 2\ table]$

$Nodes: [9 \times 0\ table]$

$>> \ h = plot(G)$

En la Figura 38 se representa la bahía de Cádiz de manera esquemática. Los vértices del 1 al 9 representan los núcleos de población antes comentados. Además, los pesos asociados a las aristas son los tiempos que un vehículo tarda en recorrer esa arista.

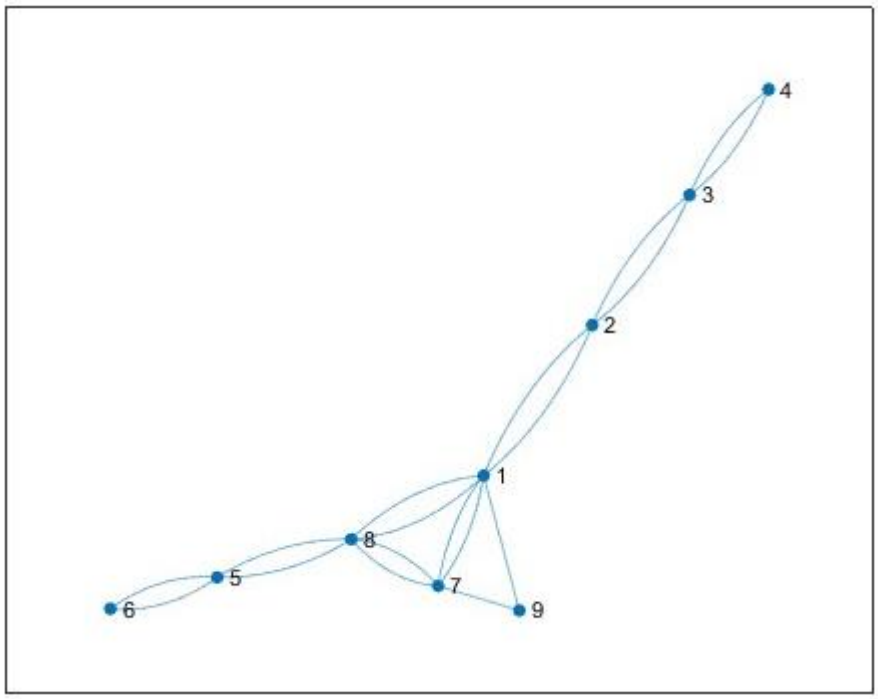



Figura 38 Representación gráfica de MATLAB del grafo asociado a la Bahía de Cádiz

A continuación, vamos a calcular los CMV desde Cádiz (vértice 1) hasta cada uno de los otros ocho núcleos urbanos principales:

$>> \ P2 = shortestpath(G, 1,2)$

$P2 =$

1 2

$>> highlight(h, P2, 'EdgeColor', 'r', 'LineWidth', 1.5)$

De manera análoga puede hacerse con la conexión con el resto de poblaciones, obteniéndose la tabla siguiente en la que se representan los resultados de los cálculos de los CMV desde Cádiz (vértice 1).

| Desde | Hasta | CMV | Longitud del camino | Distancia (en min) |
|---|---|---|---|---|
| **Cádiz** | **SAN FERNANDO** | 1 ↔ 2 | 1 | 20 |
| **Cádiz** | **CHICLANA** | 1 ↔ 2↔ 3 | 2 | 35 |
| **Cádiz** | **CONIL** | 1 ↔ 2 ↔ 3 ↔ 4 | 3 | 60 |
| **Cádiz** | **CHIPIONA** | 1 ↔ 8 ↔ 5 | 2 | 70 |
| **Cádiz** | **SANLUCAR** | 1 ↔ 8 ↔ 5 ↔ 6 | 3 | 79 |
| **Cádiz** | **PUERTO REAL** | 1 ↔ 7 | 1 | 20 |
| **Cádiz** | **PUERTO DE SANTAMARÍA** | 1 ↔ 7 ↔ 8 | 2 | 40 |
| **Cádiz** | **BARRIO JARANA / MARQUESADO** | 1 ↔ 7↔ 9 | 2 | 35 |

La representación gráfica del CMV desde Cádiz (vértice 1) hasta San Fernando (vértice 2) y de Puerto Real (vertice 7) hasta Conil (vértice 4) pueden observarse en las siguientes figuras.

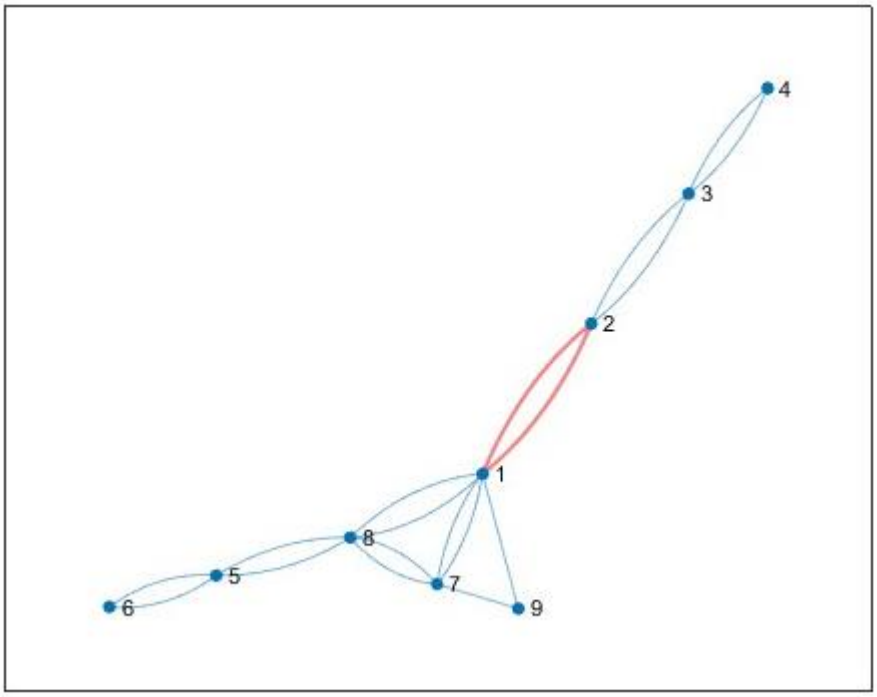



Figura 39 Representación gráfica de MATLAB del grafo asociado a la Bahía de Cádiz y el camino más corto entre Cádiz y San Fernando.

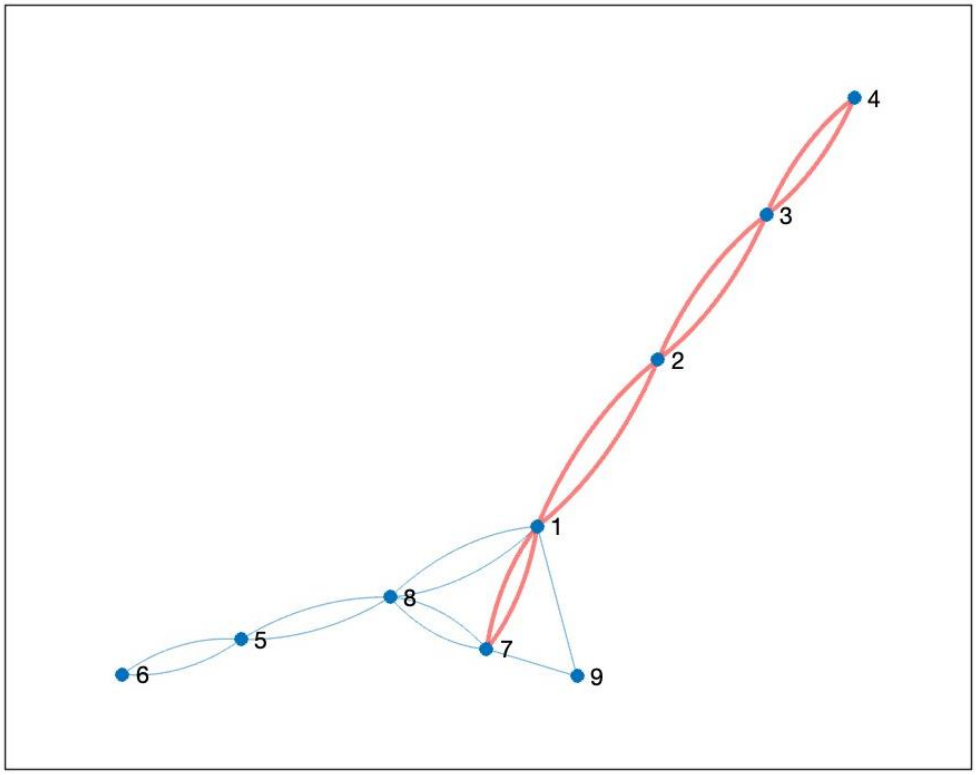


Figura 40 Representación gráfica de MATLAB del grafo asociado a la Bahía de Cádiz y el camino más corto entre Puerto Real y Conil.

# 7 APLICACIÓN 4ª: FLUJO DE VALOR MÁXIMO

En esta cuarta aplicación vamos a ver cómo resolver el problema de calcular un flujo de valor máximo. Para ello, tendremos un grafo G y un vector de pesos p que a cada arco e le asignará un peso pe. En este tema el peso de cada arco representará la capacidad de dicho arco. Tenemos por tanto que encontrar la manera óptima de asignar el flujo, de manera que este valor sea máximo y que en cada vértice intermedio entre y salga la misma cantidad de flujo.

## 7.1 Resolución con MATLAB

### 7.1.1 El comando *maxflow*

Partimos MATLAB dispone del comando *maxflow* que nos permite resolver fácilmente el problema del flujo máximo. Su uso es muy sencillo y solamente hemos de indicarle los vértices inicial y final (los vértices fuente y salida, utilizando la misma notación que en las clases de teoría). Si *s* y *t* denotan, respectivamente, la fuente y la salida de la red, la expresión a utilizar será:

$>> \ maxflow(G, s, t)$

De esta manera obtendremos el valor del flujo máximo permitido por esa red. Sin embargo, lo más interesante de dicho comando no es que calcule el valor del flujo máximo, sino que nos proporciona el grafo asociado al flujo máximo, esto es, el grafo en el que a cada arco se le asigna como peso el valor del flujo, siempre que el flujo sea no nulo. Para ello, utilizamos la siguiente instrucción:

$>> \ [flujo, Grafo] = maxflow(G, s, t)$

Bajo el nombre de *flujo* se almacenará el valor del flujo máximo, y *Grafo* el grafo asociado a este flujo. Con este nuevo grafo podemos utilizar todos los comandos vistos en la primera práctica, como por ejemplo los relativos a la representación gráfica.

Además de obtener el flujo máximo y su grafo asociado, también es posible obtener por pantalla el corte de valor mínimo. Para ello debemos añadir dos argumentos de salida, donde se almacenarán los vértices de la parte inicial y final del corte, respectivamente:

$>> \ [flujo, Grafo, corteInicial, corteFinal] = maxflow(G, s, t)$

**Ejemplo 7.1.** *Consideremos el grafo representado en la Figura 41. El primer paso será definir este grafo en MATLAB:*

$>> \ s = [1\ 1\ 2\ 3\ 3];$

$>> \ t = [2\ 3\ 4\ 2\ 4];$

$>> \ cap = [3\ 4\ 2\ 2\ 5];$

$>> \ G = digraph(s, t, cap)$

$G \ =$

$digraph\ with\ properties:$

$Edges: [5x2\ table]$

$Nodes$: $[4x0\ table]$

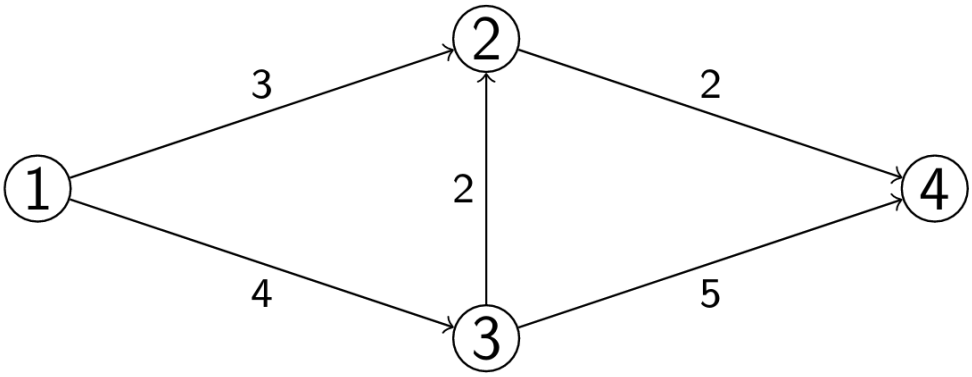



Figura 41 Grafo del Ejemplo 7.1.

*A continuación, para calcular el flujo máximo entre los vértices 1 y 4, así como su grafo y corte mínimos asociados, utilizamos la siguiente instrucción:*

$>>\ [flujo, Grafo, corteInicial, corteFinal] = maxflow(G, 1,4)$

$flujo\ =$

6

$Grafo\ =$

$digraph\ with\ properties$:

$Edges$: $[4x2\ table]$

$Nodes$: $[4x0\ table]$

$corteInicial\ =$

1

2

$corteFinal\ =$

3

4

*Vemos por tanto que el valor del flujo máximo es de 6 unidades, y ese flujo máximo se obtiene con el corte formado por los vértices 1 y 2 y los vértices 3 y 4. Vamos a pedir por pantalla los arcos del grafo asociado al flujo máximo para ver de qué manera se reparte el flujo:*

$>>\ Grafo.Edges$

$ans\ =$

| $EndNodes$ | | $Weight$ |
|---|---|---|
| 1 | 2 | 2 |
| 1 | 3 | 4 |
| 2 | 4 | 2 |
| 3 | 4 | 4 |

*Eso significa que para obtener el flujo máximo hemos de asignar el valor 2 a los arcos 1 → 2 y 2 → 4 y el valor 4 a los arcos 1 → 3 y 3 → 4. En la Figura 44 hemos representado en rojo los valores del flujo máximo y hemos puesto en línea discontinua el corte de valor mínimo:*

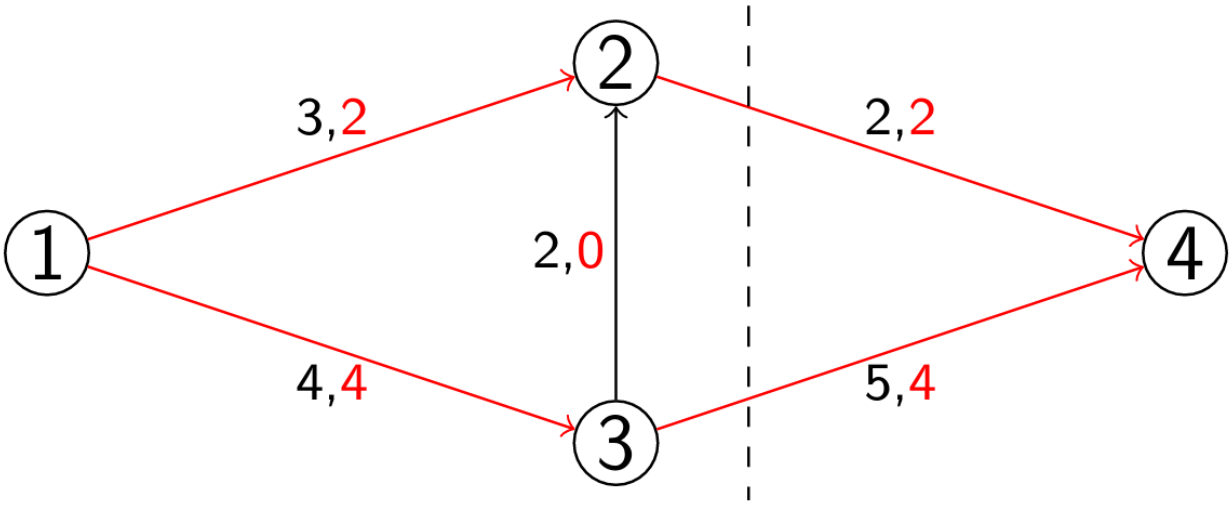



Figura 42 Grafo del Ejemplo 7.1 con su flujo máximo (en rojo) y corte mínimo (en línea discontinua).

### 7.1.2 Opciones del comando *maxflow*

En el apartado anterior ya hemos visto las opciones relativas a los argumentos de salida del comando *maxflow*. Además de estas opciones, también nos permite indicar el tipo de algoritmo que deseamos utilizar:

a) El algoritmo utilizado por defecto es el de Boykov-Kolmogorov. Para su uso habría que indicar la opción *searchtrees*.

b) Una alternativa, solamente válida para grafo orientados, es el uso del algoritmo de Ford-Fulkerson. Para ello deberíamos de indicar la opción *augmentpath*.

c) Para utilizar una segunda alternativa, también válida únicamente para grafos orientados, deberíamos utilizar la opción *pushrelabel*.

### 7.1.3 Representación gráfica del flujo de valor máximo

Al igual que hemos visto en las aplicaciones anteriores, una vez resuelto el problema del flujo máximo podemos utilizar las opciones de las representaciones gráficas vistas en la primera aplicación para indicar cuál es el flujo máximo y el corte mínimo. Comenzamos haciendo la representación gráfica del grafo original:

$>> \ h = plot(G)$

A continuación, remarcamos en color (rojo, en este caso) los arcos con un flujo no nulo:

$>> \ highlight(h, Grafo, 'EdgeColor', 'red')$

Además, podemos añadir a esos arcos remarcados el valor de su flujo asociado:

$>> \ cap = Grafo.Edges.EndNodes;$

$>> \ labeledge(h, cap(:,1), cap(:,2), Grafo.Edges.Weight);$

Con la primera instrucción almacenamos los vértices inicial y final de cada arco, y con la segunda instrucción asignaremos a cada arco el valor de su flujo. Por último, también podemos distinguir el corte de valor mínimo:

$>> \ highlight(h, corteInicial, 'NodeColor', `blue')$

$>> \ highlight(h, corteFinal, 'NodeColor', `green')$

Con la primera instrucción dibujaremos en azul los vértices de la parte inicial del corte de valor mínimo, y con la segunda instrucción dibujamos en verde los vértices de la parte final del corte.

**Ejemplo 7.2.** *Continuemos con el Ejemplo 7.1. En la Figura 44 habíamos representado el flujo máximo y el corte mínimo. Utilizando las instrucciones citadas anteriormente, MATLAB realizaría la representación gráfica de la Figura 45.*

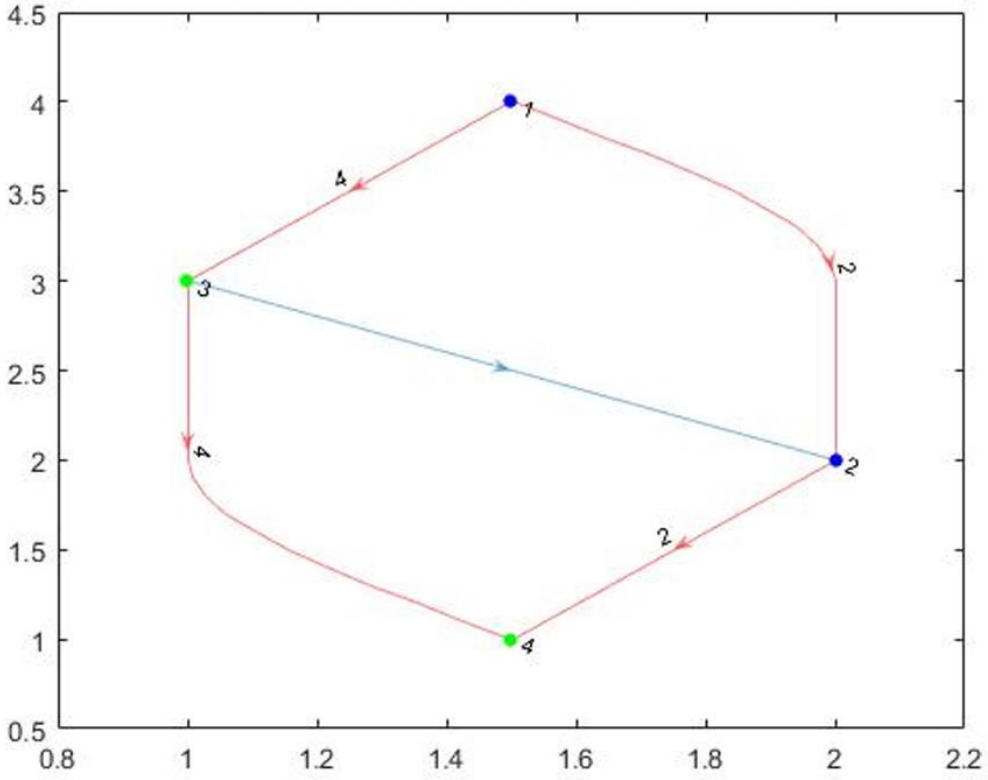



Figura 43 Representación gráfica de MATLAB del flujo máximo del grafo del Ejemplo 7.1.

## 7.2 Ejemplo ilustrativo

Una compañía de gas Gas Suco quiere enviar la mayor cantidad posible de petróleo por hora a través de oleoductos desde el vértice llamado *F* hasta el vértice llamado *S* (*F=fuente, S=salida.*). A lo largo de este trayecto, el petróleo podría pasar por las estaciones 1, 2 y/o 3. En la Figura 44 hemos representado gráficamente la estructura de estas estaciones, así como de los vértices inicial *F* y final *S*. Cada uno de los arcos tiene asociado un número que indica la capacidad de gas (en millones de litros por hora) que se pueden bombear a lo largo del arco. El objetivo es por tanto maximizar la cantidad de petróleo enviado desde *F* hasta *S* e indicar cómo debería hacerse dicho envío.

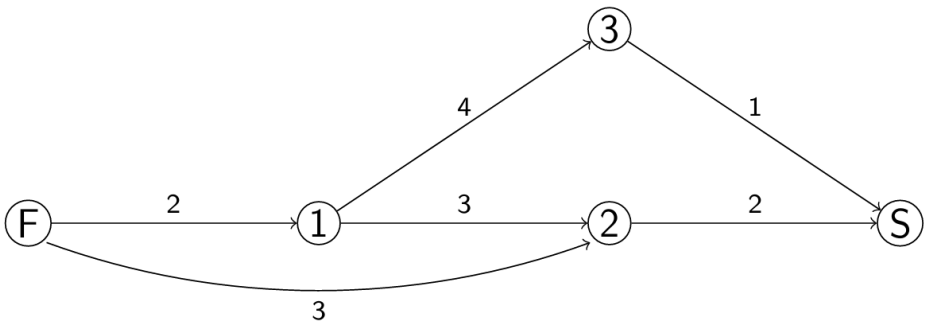



Figura 44 Distribución de las estaciones de la compañía Gas Suco con la capacidad de cada oleoducto.

Como hemos hecho en otras ocasiones, el primer paso será definir el grafo asociado a las estaciones en MATLAB:

$>> \ s = [1\ 1\ 2\ 2\ 3\ 4];$

$>> \ t = [2\ 3\ 3\ 4\ 5\ 5];$

$>> \ capacidad = [2\ 3\ 3\ 4\ 2\ 1];$

$>> \ G = digraph(s, t, capacidad, \{'F', 'Estación\ 1', 'Estación\ 2', 'Estación\ 3', 'S'\})$

$G \ =$

$digraph\ with\ properties$:

$Edges$: $[6x2\ table]$

$Nodes$: $[5x1\ table]$

$h = plot(G)$

Con esta última instrucción hemos solicitado la representación gráfica realizada por MATLAB, que podemos ver en la figura.

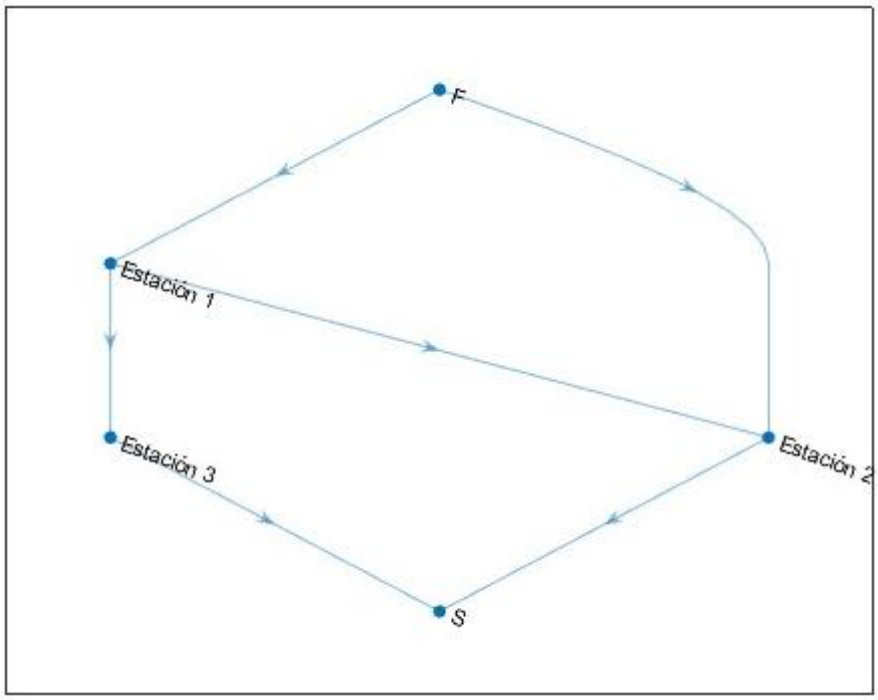



Figura 45 Representación gráfica de MATLAB del sistema de tuberías de la compañía Gas Suco.

En términos de grafos y redes, el problema se traduce en encontrar el flujo máximo en esta red:

$>> [flujo, Grafo] \ = \ maxflow(G, 'F', 'S')$

$flujo \ =$

3

$Grafo \ =$

$digraph\ with\ properties$:

$Edges$: $[5 \times 2\ table]$

$Nodes$: $[5 \times 1\ table]$

Con esta orden obtenemos el valor del flujo máximo, que es de 3 unidades, es decir, 3 millones de litros a la hora. Sin embargo, esta información es limitada, ya que no nos dice cómo debemos distribuir el flujo. A continuación, vamos a utilizar la misma instrucción *maxflow* pero solicitando tanto el grafo asociado al flujo máximo como el corte mínimo:

$>> \ [flujo, Grafo, corteInicio, corteFinal] = maxflow(G, 'F', 'S')$

$flujo\ =$

3

$Grafo\ =$

$digraph\ with\ properties$:

$Edges$: [$5 \times 2\ table$]

$Nodes$: [$5 \times 1\ table$]

$corteInicio\ =$

$4 \times 1\ cell\ array$

$\{'F'\quad\}$

$\{'Estación\ 1'\}$

$\{'Estación\ 2'\}$

$\{'Estación\ 3'\}$

$corteFinal\ =$

$1 \times 1\ cell\ array$

$\{'S'\}$

$>>\ Grafo.Edges$

$ans\ =$

$5 \times 2\ table$

| $EndNodes$ | | $Weight$ |
|---|---|---|
| $'F'$ | $'Estación\ 1'$ | 1 |
| $'F'$ | $'Estación\ 2'$ | 2 |
| $'Estación\ 1'$ | $'Estación\ 3'$ | 1 |
| $'Estación\ 2'$ | $'S'$ | 2 |
| $'Estación\ 3'$ | $'S'$ | 1 |

Con la última instrucción utilizada $Grafo.Edges$, sabemos cómo debemos distribuir el flujo para obtener el valor del flujo máximo, que ya sabemos que es de 3 millones de litros a la hora:

- Desde la estación de origen se enviará un millón de litros a la hora a la estación 1 y dos millones a la estación 2.
- Desde la estación 1 se enviará un millón de litros a la hora a la estación 3.
- Desde la estación 2 se enviarán dos millones de litros a la hora a la estación final.
- Desde la estación 3 se enviará un millón de litros a la hora a la estación final.

Si queremos la representación gráfica de MATLAB del grafo asociado al flujo máximo, podemos utilizar las siguientes instrucciones:

$>>\ h.EdgeLabel = \{\};$ $\#\ Con\ esta\ instrucción\ eliminamos\ los\ valores\ de\ la$

$\#\ capacidad\ de\ cada\ arco.$

$>>\ highlight(h, Grafo, 'EdgeColor', 'r')$ $\#\ Con\ esta\ instrucción\ remarcamos\ los$

$\#\ arcos\ con\ valores\ de\ flujo\ no\ nulo.$

$>> cap = Grafo.Edges.EndNodes;$ $\#$ *Con esta instrucción guardamos en el*
$\#$ *vector cap los vértices inicial y*
$\#$ *final de cada arco.*

$>> labeledge(h, cap(:,1), cap(:,2), Grafo.Edges.Weight)$

$>> highlight(h, corteInicio,' NodeColor',' red')$

$>> highlight(h, corteFinal,'NodeColor','blue')$

Las dos últimas instrucciones dibujarán los vértices de la parte inicial del corte mínimo en rojo y los vértices de la parte final del corte en azul. La representación final de MATLAB puede observarse a continuación.

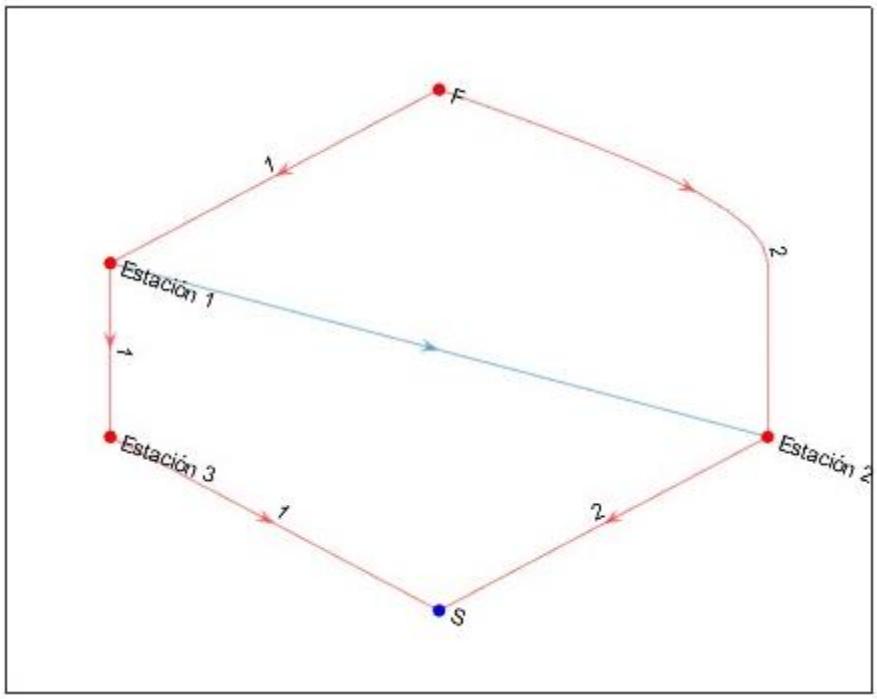



Figura 46 Representación gráfica de MATLAB del grafo asociado al flujo máximo en el sistema de tuberías de la compañía Gas Suco.

Asimismo, en la Figura 47 se representa el flujo máximo de acuerdo con la representación original, dibujando en rojo los arcos utilizados y añadiendo en rojo el valor del flujo. Además, la línea discontinua representa el corte de valor mínimo.

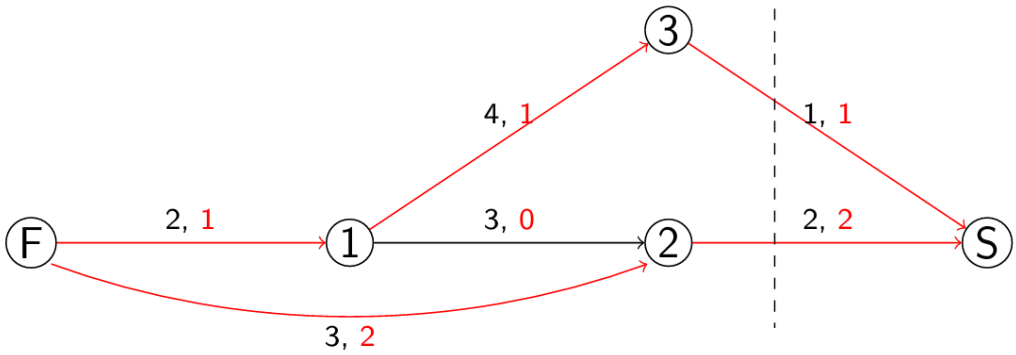



Figura 47 Distribución del flujo máximo en las estaciones petrolíferas de la compañía Sunco Oil con la capacidad de cada oleoducto.

# REFERENCIAS


[1] D. Easley y J. Kleinberg, «Networks, Crowds, and Markets: Reasoning about a Highly Connected World,» *Cambridge University Press*, 2010.

[2] J. Moody, «Race, School Integration, and Friendship Segregation in America,» *American Journal of Sociology*, 2001.

[3] R. K. Ahuja, A. V. Goldberg, J. B. Orlin y R. E. Tarjan, Finding minimum costflows by double scaling. Mathematical Programming, 1992, p. 243–266.

[4] D. A. Castanon, Efficient algorithms for finding the k best paths through a trellis, IEEE Trans. Aerospace and Electronic Systems, 1990, p. 405–410.

[5] M. H. a. H. K. M. Charikar, Improved approximation algorithms for label cover problems. In Proc. 17th Annu. European Sympos. Algorithms, volume 5757 of Lecture Notes Comput. Sci., 2009, p. 23–34.

[6] G. K. a. W. S. G. Even, On network design problems: fixed cost flowsand the covering steiner problem. ACM Trans. Algorithms, 2005, p. 74–101.

[7] M. G. a. D. S. Johnson, Computers and intractability : A guide to the theory of NP-completeness, W. H. Freeman, 1979.

[8] A. V. G. a. S. Rao, Beyond the flow decomposition barrier, J. ACM, 1998, p. 783–797.

[9] Y. K. a. C. Sommer, On shortest disjoint paths in planar graphs. Discrete Optimization, 2010, p. 234–245.

[10] H. N. S. S. H.-C. W. a. R. R. S. O. Krumke, Flow improvement and network flows with fixed costs. In Proc. Internat. Conf. Oper. Res., 1998, p. 158–167.

[11] O. Borůvka, About a certain minimal problem, 1926.

[12] M. T. Omran, J.-R. Sack y H. Zarrabi-Zadeh, «Finding paths with minimum shared edges,» *Journal of combinatorial optimization*, pp. 709-722, 2013.

[13] D. B. a. K. Madduri, Design and implementation of the HPCS graph. In Proc. 12th Internat. Conf. HighPerform. Comput., volume 3769 of Lecture Notes Comput. Sci., 2005, p. 465–476.

[14] S. P. Borgatti, «Centrality and network flow,» *Social Networks*, pp. 55-71, 2005.

[15] C. Bird, On cost allocation for a spanning tree: A game theoretic approach. Networks 6(4), 1976, pp. 335-350.