Sección de Matemáticas
Universidad de La Laguna

Ayoze González Martín

Asignación bajo preferencias

Assignment under preferences

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Mes de Año

Dirigido por
Carlos González Alcón

Carlos González Alcón
Departamento de Matemáticas,
Estadística e Investigación
Operativa
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

Agradecer a mi tutor Carlos por su entrega e implicación en esta memoria, sobretodo por sus ideas y consejos siempre tan acertados.

A toda mi familia, en especial a mis padres y hermano, su ayuda en todos esos días de estudio ha sido muy importante para mí, así como su apoyo al embarcarme en esta aventura que desde un principio sabíamos que no sería fácil.

A mis amigos, tanto a los de siempre como a los que he podido hacer durante el grado, sin ellos hubiera tirado la toalla hace mucho tiempo y sus consejos me han valido para seguir adelante.

A Alba, con su forma de ser me enseñó que siempre que algo te guste y apasione no tienes que pensarlo dos veces e ir a por ello.

Ayoze González Martín
La Laguna, 10 de julio de 2023

Resumen · Abstract

Resumen

La teoría de emparejamiento pretende obtener asignaciones en diferentes ambientes que puedan propiciar buenos resultados para todas las partes involucradas. En este trabajo se estudiarán tres problemas de asignación como son el problema de matrimonios, el de habitaciones y el de aceptación en universidades. Se profundizará en el primer problema dando una generalización del mismo así como su implementación en el lenguaje de programación R. Se verán casos en la vida real del uso de estos algoritmos que han dado lugar a grandes avances para nuestra sociedad como son los trasplantes de órganos o las residencias de estudiantes en hospitales.

Palabras clave: *Emparejamiento – Estable – Pareja*

Abstract

Matching theory aims to obtain assignments in different environments that can provide good outcomes for all involved parties. In this work, three assignment problems will be studied: the marriage problem, the roommates problem, and the college admissions problem. The first problem will be further explored by presenting a generalization of it and its implementation in the R programming language. Real-life cases will be examined to showcase the use of these algorithms, which have led to significant advancements for our society, such as organ transplants and student residencies in hospitals.

Keywords: *Pairing – Stable – Couple*

Contenido

Introducción

La teoría de emparejamiento es una rama de la teoría de juegos que se centra en cómo se forman los grupos en diferentes contextos y cómo estos pueden ser mejorados para obtener resultados más beneficiosos. Un ejemplo de aplicación de esta teoría se encuentra en los mercados laborales, donde las empresas buscan empleados y los trabajadores buscan trabajo. En este contexto, el problema trata de cómo hacer que los empleadores y los trabajadores se emparejen de tal manera que ambas partes estén satisfechas con la relación que se establezca.

David Gale y Lloyd Shapley fueron dos destacados matemáticos y economistas estadounidenses que trabajaron en diversos campos de la matemática aplicada. Gale, doctorado en Matemáticas por la Universidad de Princeton en 1949, se especializó en teoría de juegos y programación lineal, mientras que Shapley, doctorado en Filosofía en 1953 en la misma universidad que su compañero, se enfocó en la teoría de emparejamiento y la teoría de juegos cooperativos. Publicaron un artículo en 1962 [1] donde proporcionaban un algoritmo capaz de dados dos conjuntos, en su caso de hombres y mujeres con el mismo número de individuos y que cada uno tenga una lista de preferencia donde se ordene de mejor a peor las personas del conjunto que daba como resultado un emparejamiento en el que ninguno de los miembros de dos parejas distintas se prefieran antes que a su pareja actual, o como los autores lo denominaron, un emparejamiento estable. Desde su publicación hasta día de hoy, grandes figuras del mundo matemático han intentado mejorar de alguna manera este algoritmo que a día de hoy se sigue utilizano. Esto se debe a que Gale y Shapley nos dejaron un método que es capaz de ofrecernos un emparejamiento donde aquellos que proponen serán los que obtengan el mejor resultado de acuerdo a sus preferencias, mientras que los propuestos recibirán el peor. Pero, en 1989, F. Masarani afirmó junto a Sadik S. Gokturk [6] que no era posible encontrar un algoritmo capaz de darnos un emparejamiento que satisfaga todas las preferencias de los individuos de ambos conjuntos.

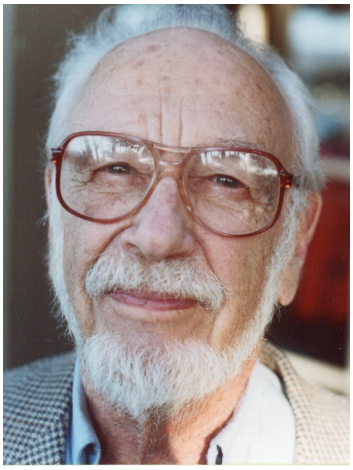


Figura 0.1. David Gale (1921-2008)

Figura 0.2. Lloyd S. Shapley (1923-2016)

Este trabajo constará de tres capítulos donde se hablará de los problemas de asignación que Gale y Shapley mencionaron en su artículo. Empezaremos con los distintos conceptos de la teoría de emparejamientos y el ya expuesto problema de asignación de matrimonios, pero daremos un paso más en este problema. Nos centraremos en la idea de que no sean necesarios dos conjuntos del mismo tamaño, así como que las listas de preferencia no tengan que ser completas, contemplando que nuestros individuos puedan quedarse solteros.

En nuestro segundo capítulo daremos un vistazo al problema de asignación de habitaciones, el cual consiste en emparejar miembros de un mismo conjunto en habitaciones de dos ocupantes. Para ello nos apoyaremos en el trabajo descrito por Robert W. Irving en 1985 [2] donde nuestro autor nos presenta un algoritmo en dos fases capaz de darnos un emparejamiento estable si este existiera.

Por último, en el tercer capítulo explicaremos en qué consiste el problema de asignación de universidades. Este caso trata de emparejar un conjunto de estudiantes con otro de universidades, donde ambos cuentan con sus listas de preferencia pero, a diferencia del primer problema, las universidades pueden contar con un cupo máximo de alumnos que pueden aceptar. Veremos que Gale y Shapley nos dieron un algoritmo para poder resolver este problema que proporciona como resultado un emparejamiento estable. Otras figuras también han conseguido dar con formas de resolver este problema desde distintos puntos de vista, entre los que destaca Alvin E. Roth, economista estadounidense doctorado por la Universidad de Stanford en 1974 combinando matemáticas y logística. En 1982 [4], Roth propuso lo que hoy conocemos como el algoritmo de Boston. Este algoritmo, junto al de Gale y Shapley, permitió entre otros proyectos que la Real Academia de Ciencias de Suecia le otorgara el popularmente conocido premio Nobel de Economía en 2012 a Roth y Shapley por su trabajo en el diseño de

mercado y la teoría de juegos cooperativos. Estos autores han contribuido a la teoría y práctica del diseño de mercado, así como han desarrollado algoritmos que mejoran la eficiencia de los mercados y su impacto en la vida real.

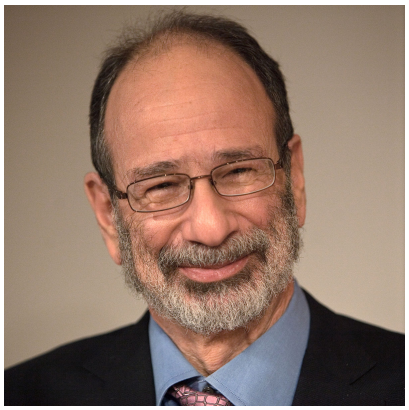


Figura 0.3. Alvin E. Roth (1951-)

1

Problema de asignación de matrimonios

La teoría de emparejamientos es una rama de la teoría de juegos y matemáticas aplicadas que se centra en estudiar cómo se pueden asignar recursos o personas de manera óptima y justa, en función de ciertos criterios. Uno de los problemas clásicos que aborda esta teoría es el problema de asignación de matrimonios, que trata de emparejar a un conjunto de hombres y mujeres de la manera más satisfactoria posible para ambas partes.

David Gale y Lloyd S. Shapley propusieron una solución ingeniosa para este problema en su artículo de 1962, conocida como el algoritmo de Gale-Shapley. Su solución se basa en la idea de que cada hombre y mujer tiene una lista de preferencias ordenada de los demás miembros del grupo opuesto. A partir de estas listas, el algoritmo propone un emparejamiento estable en el que no existe un hombre y una mujer que prefieran estar juntos en lugar de con su pareja asignada.

En este capítulo, exploraremos en detalle cómo funciona el algoritmo de Gale-Shapley y algunas de sus propiedades más importantes, así como también veremos una implementación en el lenguaje de programación R de una versión generalizada que proponemos.

1.1. Emparejamiento y estabilidad

Para dar comienzo a este primer capítulo, durante esta sección se definirá el concepto de emparejamiento entre dos conjuntos disjuntos donde cada individuo cuenta con una lista de preferencia sobre los elementos del otro conjunto. También hablaremos del término de estabilidad para las asignaciones.

1.1.1. Emparejamiento

Considérense dos conjuntos finitos disjuntos no vacíos, H y M. Llamaremos *proponentes* y *propuestos* a los individuos de H y de M, respectivamente.

Inicialmente consideraremos que ambos conjuntos tienen tamaño n.
Cada individuo tiene una lista de preferencias completa y estricta sobre todos los del conjunto contrario. Esto quiere decir que cada persona deberá ordenar, de manera que no puedan ocurrir repeticiones ni empates, a los elementos del conjunto opuesto, pudiendo incluso estar él mismo en la lista indicando su preferencia de estar sin pareja antes que con los elementos situados en peor posición. Las listas de preferencia estarán ordenadas de manera que la primera posición la ocupe el individuo preferido y en la última el menos deseado.

Vamos a definir algunos conceptos.

Definición 1.1. Un *emparejamiento* entre el conjunto de proponentes H y el de propuestos M es una función biyectiva

$$\alpha : H \cup M \longrightarrow H \cup M$$

que satisface lo siguiente:

- $\alpha(\alpha(x)) = x$, para cada $x \in H \cup M$
- Si $h \in H$ entonces $\alpha(h) \in M \cup \{h\}$ (α empareja a cada elemento de H con uno de M o consigo mismo)
- Si $m \in M$ entonces $\alpha(m) \in H \cup \{m\}$ (α empareja a cada elemento de M con uno de H o consigo mismo)

Por la primera condición vemos que $\alpha(i) = j \Leftrightarrow \alpha(j) = i$, con lo cual, cuando $i \neq j$, podemos decir que i *forma pareja con* j en α.
Si encontramos que $\alpha(i) = i$, diremos que este individuo se queda *soltero* en α.

Como se indicó, la lista de preferencia de cada individuo es completa y estricta, por lo que no se podrán dar casos en que un h se sienta indiferente entre dos miembros de M, deberá en todo momento indicar a quién prefiere y no podrá dejar a ningún elemento de M fuera de la lista.
Podemos denotar las preferencias de la siguiente manera:

$$\text{Sea } h \in H \text{ y sean } i, j \in M \cup \{h\},\ i \succ_h j \Leftrightarrow h \text{ prefiere a } i \text{ sobre } j.$$

Con esta notación también podemos expresar la preferencia sobre estar soltero: $h \succ_h j$ indica que h prefiere quedarse soltero antes que estar con j.
Además, si queremos expresar que h prefiere estar antes con i que con j o que j pudiera ser igual a i (ver página 6), lo denotaremos de la forma $i \succeq_h j$.
Hay que añadir que no solo los elementos de H tienen preferencias, los individuos de M también cuentan con las suyas siguiendo las mismas pautas.
Dichas preferencias vendrán dadas por una permutación de $M \cup \{h\}$ para cada individuo $h \in H$. Así mismo, las preferencias de cada elemento $m \in M$ serán una permutación de $H \cup \{m\}$.

Ejemplo 1.2. Supóngase las siguientes preferencias de los conjuntos $H = \{a, b, c\}$ y $M = \{A, B, C\}$, donde cada columna corresponde a un individuo, y sus preferencias aparecen enumeradas de la más preferida a la menos.

a	b	c	A	B	C
A	A	C	a	b	c
B	B	A	b	c	C
a	C	c	c	B	a
C	b	B	A	a	b

Tabla 1.1. Preferencias del ejemplo 1.2

Si nos fijamos en las preferencias de b (columna encabezada por b),

$$A \succ_b B \succ_b C \succ_b b,$$

tenemos que prefiere a cualquier individuo del otro conjunto antes que quedarse soltero. Pero, si vemos las preferencias de C,

$$c \succ_C C \succ_C a \succ_C b,$$

indican que prefiere quedarse soltero antes que estar con a o b, dicho de otro modo, solo aceptaría estar con c.

1.1.2. Estabilidad

En esta subsección veremos definiciones y características de los emparejamientos según cumplan diversas condiciones.

Definición 1.3. Un emparejamiento α puede ser *bloqueado* por un par de individuos (i, j) cuando $i \neq \alpha(j)$ y ambos individuos prefieren estar juntos a ser emparejados con los individuos determinados por α. Esto es, $j \succ_i \alpha(i)$ e $i \succ_j \alpha(j)$. El par de individuos (i, j) recibe el nombre de *par bloqueante*.

Ejemplo 1.4. Para los conjuntos y preferencias del ejemplo 1.2 consideremos la siguiente asignación $\{(a, B), (b, A), (c, C)\}$. Podemos ver claramente que esta asignación puede ser bloqueada por la pareja (a, A), ya que en ambos casos, los individuos se prefieren entre sí antes que a su actual pareja, $A \succ_a \alpha(a) = B$ y $a \succ_A \alpha(A) = b$.

Definición 1.5. Un emparejamiento diremos que es *estable* si no puede ser bloqueado por ningún par de individuos. Diremos que el emparejamiento es *inestable* en caso de poder ser bloqueado.

Como es de esperar, no siempre será posible encontrar un emparejamiento que satisfaga en máximo grado las preferencias de todos los individuos. En el ejemplo anterior, si nos fijamos en la tabla 1.1 vemos que a y b prefieren a A

en primer lugar, pero uno de los dos tendrá que renunciar debido a que no es posible emparejar a ambos con A. En este caso, como a está mejor posicionado que b en la lista de preferencias de A, entendemos que un emparejamiento tal que $\alpha(A) = b$ no sería estable pues A siempre podrá cambiar a b por a, y a abandonaría a su pareja, sea cual sea, por A.

Ejemplo 1.6. Vamos a tomar las siguientes preferencias de los conjuntos $H = \{Augusto, Bruto, Casio\}$ y $M = \{Diana, Estela, Flavia\}$.

Augusto	***Bruto***	***Casio***	***Diana***	***Estela***	***Flavia***
Diana	*Diana*	*Flavia*	*Bruto*	*Augusto*	*Augusto*
Flavia	*Estela*	*Estela*	*Casio*	*Casio*	*Bruto*
Estela	*Flavia*	*Diana*	*Augusto*	*Bruto*	*Casio*

Tabla 1.2. Tabla con las preferencias del ejemplo 1.6

Si tomamos la asignación

$$\{(Augusto, Flavia), (Bruto, Diana), (Casio, Estela)\}$$

obtendremos un emparejamiento estable, ya que no podemos encontrar una pareja bloqueante que se prefieran entre sí antes que a sus actuales parejas.
En cambio, si tomamos la asignación

$$\{(Augusto, Diana), (Bruto, Flavia), (Casio, Estela)\}$$

tendremos que la pareja $(Bruto, Diana)$ es una pareja bloqueante, prefieren estar juntos antes que quedarse con sus respectivas parejas, por lo que es un emparejamiento inestable.

Definición 1.7. Llamaremos *núcleo* al conjunto de todos los emparejamientos estables.

1.2. El algoritmo de Gale-Shapley

Para el *problema de asignación* Shapley y Gale proponen un algoritmo que encuentra un emparejamiento estable.

1.2.1. Problema de asignación estable

Inicialmente se tomarán dos conjuntos disjuntos H y M, a los cuales llamaremos hombres y mujeres respectivamente, ambos de tamaño n. Cada elemento debe tener una lista de preferencia sobre los individuos del conjunto opuesto, esta lista será estricta, completa y no se aceptará el estado de soltería.

Teorema 1.8. *El núcleo es siempre un conjunto no vacío.*

Demostración. Podemos construir un algoritmo que nos proporcione una emparejamiento estable.

Paso 1: Cada hombre propone en primera instancia a la mujer que ocupa el primer lugar en su lista de preferencias.
Paso 2: Las mujeres que han recibido más de una propuesta aceptan (provisionalmente) al más preferido y rechazan al resto.
Paso 3: Los hombres que han sido rechazados proponen ahora a la siguiente mujer en su lista de preferencia.

Se repiten los pasos 2 y 3 hasta que cada mujer mantiene una única propuesta. Este algoritmo acaba con un emparejamiento estable debido a que las propuestas se hacen en sentido descendente en las listas de preferencia de los hombres, por lo que, si un hombre prefiere a una mujer que está mejor posicionada que la que se le asigna, esta ya le habrá descartado y por tanto no la prefiere a su pareja asignada. □

Ejemplo 1.9. Suponemos la siguiente lista de preferencias ya considerada en el ejemplo 1.6 y tratamos de encontrar un emparejamiento estable utilizando el algoritmo Gale-Shapley.

Augusto	***Bruto***	***Casio***	***Diana***	***Estela***	***Flavia***
Diana	*Diana*	*Flavia*	*Bruto*	*Augusto*	*Augusto*
Flavia	*Estela*	*Estela*	*Casio*	*Casio*	*Bruto*
Estela	*Flavia*	*Diana*	*Augusto*	*Bruto*	*Casio*

Tabla 1.3.

- Paso 1: Los hombres proponen a las mujeres que estén en primera posición, es decir, *Augusto* y *Bruto* proponen a *Diana* y *Casio* propone a *Flavia*.
- Paso 2: *Diana* ha recibido más de una propuesta por lo que le toca decidir. Como en su lista de preferencias $Bruto \succ_{Diana} Augusto$, *Diana* decide descartar a *Augusto* y permanecer con la propuesta de *Bruto*. Mientras que *Flavia* permanece con la propuesta de *Casio*.
- Paso 3: Como *Augusto* ha sido el único que ha sido rechazado en el paso anterior, le toca proponer a la siguiente mujer en su lista, es decir, *Augusto* propone a *Flavia*.
- Paso 2*: *Flavia* ahora tiene dos propuestas, la de *Augusto* y la de *Casio*, por lo que le toca decidir. Como $Augusto \succ_{Flavia} Casio$, *Flavia* decide rechazar la propuesta de *Casio* y conserva la propuesta de *Augusto*.
- Paso 3*: Ahora es *Casio* el que debe proponer a la siguiente mujer en su lista, y propone a *Estela*.
- Como en este punto todas las mujeres mantienen una única propuesta, el algoritmo acaba con el siguiente emparejamiento estable.

$$\{(Augusto, Flavia), (Bruto, Diana), (Casio, Estela)\}$$

Teorema 1.10. *El algoritmo Gale-Shapley termina en un número finito de pasos.*

Demostración. Como tenemos n proponentes y n propuestos, cada proponente, a lo sumo propone a n propuestos distintos, con lo cual, el algoritmo termina a lo sumo en n^2 pasos. □

Definición 1.11. Un emparejamiento estable α es *óptimo* para el conjunto H si cada solicitante está por lo menos tan bien bajo de él como sujeto por cualquier otro emparejamiento estable, es decir, se deberán cumplir las dos condiciones siguientes:

- α es estable.
- Para cada $h \in H$ tenemos que $\alpha(h) \succeq_h \beta(h)$ para todo β emparejamiento estable.

Definición 1.12. Diremos que un emparejamiento estable α es *pésimo* para el conjunto H si se cumplen las dos condiciones siguientes:

- α es estable.
- Para cada $h \in H$ tenemos que $\alpha(h) \preceq_h \beta(h)$ para todo β emparejamiento estable.

Definición 1.13. Diremos que (h, m) es una *pareja válida* si existe un emparejamiento estable α tal que $\alpha(h) = m$.

Teorema 1.14. *En el proceso definido por Gale-Shapley no existen parejas válidas rechazadas.*

Demostración. Suponemos por reducción al absurdo que el primer rechazo de una pareja válida es el de m a h por w en una etapa i-ésima. Entonces $w \succ_m h$ y por la definición 1.13. existe un emparejamiento estable α tal que $\alpha(h) = m$. Sea k tal que $\alpha(w) = k$. Dado que la primera pareja válida rechazada es la de m a h por w, tenemos que $m \succ_w k$, ya que si no existiría un $j < i$ tal que se daría que en la etapa j-ésima k ha rechazado a w. Pero esto no podría suceder ya que sabíamos que el primer rechazo de pareja válida es de m a h en la etapa i-ésima. Esto contradice que α sea estable ya que tenemos que $w \succ_m \alpha(m) = h$ y $m \succ_w \alpha(w) = k$, por lo que (w, m) es un par bloqueante de α. □

Teorema 1.15. *El algoritmo Gale-Shapley proporciona un emparejamiento óptimo para los proponentes y pésimo para los propuestos.*

Demostración. Tomamos H y M los conjuntos de los proponentes y los propuestos respectivamente y sea α el emparejamiento estable de aplicar Gale-Shapley. Vamos a suponer por reducción al absurdo que existe un emparejamiento estable

β y existe $h \in H$ tal que $\beta(h) \succ_h \alpha(h)$, por tanto, en la construcción del emparejamiento α, h rechaza a $\beta(h)$ en algún momento debido a que h propone en orden decreciente según su lista de preferencia, por tanto nos encontramos con una contradicción con el Teorema 1.14 ya que $(h, \beta(h))$ es una pareja válida.
Ahora vamos a ver que para los propuestos es pésimo. Sea $m \in M$ tal que $\alpha(m) = h$ y sea γ cualquier emparejamiento estable tal que $\gamma(m) = s$ y $\gamma(h) = w$. Vamos a suponer por reducción al absurdo que $\alpha(m) \succ_m \gamma(m)$. Entonces $h \succ_m s$ y, como α es óptimo para los proponentes, $\alpha(h) \succ_h \gamma(h)$ con lo que $m \succ_h w$ y tendríamos que (h, m) sería el par bloqueante de γ ya que $h \succ_h \gamma(m) = s$ y $m \succ_h \gamma(h) = m$, llegando así a un absurdo ya que habíamos supuesto que γ es estable. □

Teorema 1.16. *El emparejamiento generado por Gale-Shapley es independiente del orden de las proposiciones.*

Demostración. Tomamos α y β dos emparejamientos generados por la ejecución de Gale-Shapley en distinto orden, y sea H el conjunto de los proponentes. Por el teorema 1.15 sabemos que α y β son óptimos para los proponentes y tenemos que para todo $h \in H$:

- $\alpha(h) \succeq_h \gamma(h)$ para todo γ emparejamiento estable, en particular $\alpha(h) \succeq_h \beta(h)$ para todo $h \in H$
- $\beta(h) \succeq_h \gamma(h)$ para todo γ emparejamiento estable, en particular $\beta(h) \succeq_h \alpha(h)$ para todo $h \in H$

Por lo que no queda más remedio que $\alpha(h) = \beta(h)$ para todo $h \in H$. □

1.3. Implementación en R

En esta sección ofrecemos una codificación del algoritmo para el problema de asignación de matrimonios en el lenguaje de programación R [7]. También veremos una variante al problema propuesto inicialmente en el que no se exigirá que el número de hombres y mujeres sea el mismo, y además se podrá considerar el estado de soltero o soltera.

1.3.1. Algoritmo de matrimonios en R

En primer lugar vamos a crear una función que nos ayude a comprobar que en nuestras listas de preferencias no ocurre ninguna repetición.
El código está disponible en [8].

```
unicos <- function(v){
    vs0 <- v[v!=0]
    length(vs0) == length(unique(vs0))
}
```

Una vez creada esta función auxiliar, procedemos a construir nuestro algoritmo en R donde se crea una función que tenga como entrada las matrices de preferencias de los individuos de cada conjunto. Cabe destacar que el conjunto que tomemos como A será el que proponga al conjunto B.

```
AsigMatrimonio1 <- function(prefA,prefB){ # Tiene como entrada
# prefA= matriz de preferencia de los que proponen y
# prefB= matriz de preferencia de los propuestos
numA <- dim(prefA)[1]
numB <- dim(prefB)[1]
# Primero comprobamos que las dimensiones de las matrices
# son correctas y que no se repiten preferencias
if (dim(prefA)[2]>numB || dim(prefB)[2]>numA){
  return(print("Hay que revisar las matrices"))
} else {
  if (all(apply(prefA,1,unicos))==FALSE ||
  all(apply(prefB,1,unicos)==FALSE)){
    return(print("Hay que revisar las listas de preferencia"))
  }
}
asigAp <- rep(1,numA)  # Ronda de propuesta de cada hombre
asigAs <- rep(0,numA)  # Asignación de cada hombre
asigBp <- rep(numA+1,numB)  # Preferencia de cada hombre
# asignado a la mujer
asigBs <- rep(0,numB)  # Asignación de cada mujer
noasignados <- 1:numA  # Vector de hombres no emparejados
while(!all(asigAs != 0)){   # Mientras algún hombre no esté
# asignado se repite
  for(candidato in noasignados){  # Para aquellos no asignados
    propB <- prefA[candidato,asigAp[candidato]]  # Almacenamos
    # la propuesta del candidato según la ronda de propuesta
    # por la que se encuentre
    bondadcandidato <- match(candidato, prefB[propB,])
    # Almacenamos la posición de pref del candidato en
    # la lista de la mujer
    if(bondadcandidato < asigBp[propB]){  # Si la asignación
    # anterior es peor que la nueva
      asigAs[asigBs[propB]] <- 0  # Desasignamos al candidato
      # en peor posición
      asigBp[propB] <- bondadcandidato  # Almacenamos la nueva
      # posición del hombre en la lista de la mujer
      asigBs[propB] <- candidato  # Asignamos el candidato a
      # la propuesta del mismo
```

```
      asigAs[candidato] <- propB  # Asignamos la propuesta
      # al candidato
      asigAp[asigBs[propB]] <- asigAp[asigBs[propB]] + 1
      # Aumentamos en uno la ronda de propuesta del candidato
      # rechazados
    } else {
      asigAp[candidato] <- asigAp[candidato] + 1  # Aumentamos
      # en uno la ronda de propuesta del candidato
    }
  }
  noasignados <- which(asigAs == 0)  # Almacenamos aquellos
  # hombres que no han sido asignados
}
names(asigAs) <- 1:numA
print("La asignación final viene dada por:"); print(asigAs)
}
```

Ya implementado el algoritmo de Gale-Shapley en R vamos a ver un ejemplo.

Ejemplo 1.17. Vamos a usar las listas de preferencia de la tabla 1.3 y tratamos de encontrar un emparejamiento estable utilizando el algortimo Gale-Shapley. Primero debemos tomar un formato que R pueda entender, para ello colocamos las preferencias de forma numérica asignando números a cada hombre y mujer en orden, es decir, *Augusto*, *Bruto* y *Casio* son 1, 2 y 3 respectivamente, y *Diana*, *Estela* y *Flavia* son 1, 2 y 3 respectivamente, por tanto, las listas de preferencia quedarían de la siguiente manera:

```
# Prefencias de los hombres
h_1 <- c(1,3,2)
h_2 <- c(1,2,3)
h_3 <- c(3,2,1)
# Preferencias de las mujeres
m_1 <- c(2,3,1)
m_2 <- c(1,3,2)
m_3 <- c(1,2,3)
# Matrices de preferencias de los hombres y las mujeres
prefA <- rbind(h_1,h_2,h_3); prefA
prefB <- rbind(m_1,m_2,m_3); prefB

A matrix: 3 × 3 of type dbl
h_1        1 3 2
h_2        1 2 3
h_3        3 2 1
A matrix: 3 × 3 of type dbl
```

```
m_1         2 3 1
m_2         1 3 2
m_3         1 2 3
```

Ahora que tenemos las matrices solo falta aplicar nuestra función, la cual nos dará un emparejamiento estable.

```
AsigMatrimonio1(prefA,prefB)
```

```
[1] "La asignación final viene dada por:"
[1] 1 2 3
[1] 3 1 2
```

Es decir, nuestro primer hombre *Augusto* queda emparejado con nuestra tercera mujer *Flavia*, nuestro segundo hombre *Bruto* queda emparejado con nuestra primera mujer *Diana* y nuestro tercer hombre *Casio* queda emparejado con nuestra segunda mujer *Estela*, dando lugar al siguiente emparejamiento estable:

$$\{(Augusto, Flavia), (Bruto, Diana), (Casio, Estela)\}$$

Podemos comprobar que es exactamente igual a la asignación estable resultante del ejemplo 1.9.

1.3.2. Generalización del algoritmo de matrimonios en R

Después de implementar el algoritmo formulado por Gale y Shapley nos podemos preguntar si habría alguna forma de reescribir el código de modo que no se necesitaran conjuntos del mismo tamaño, por lo que algunos elementos quedarán necesariamente desparejados (solteros). Además podríamos hacer que las listas de preferencias no fueran completas. Esto indicaría que ese sujeto prefiere quedarse soltero si no se le empareja con los que sí figuran en la lista.
El código está disponible en [8].

```
AsigMatrimonio2 <- function(prefA,prefB){ #Tiene como
# entrada prefA= matriz de preferencia de los que proponen
# y prefB= matriz de preferencia de los propuestos
numA <- dim(prefA)[1]       # Número de proponentes
numB <- dim(prefB)[1]       # Número de personas que son propuestas
# Primero comprobamos que las dimensiones de las matrices son
# correctas y que no se repiten preferencias
if (dim(prefA)[2]>numB || dim(prefB)[2]>numA){
  return(print("Hay que revisar las matrices"))
} else {
  if (all(apply(prefA,1,unicos))==FALSE ||
  all(apply(prefB,1,unicos)==FALSE)){
    return(print("Hay que revisar las listas de preferencia"))
```

```
  }
}
asigAp <- rep(1,numA)       # Ronda de propuesta de cada proponente
asigAs <- rep(0,numA)       # Asignación de cada proponente
asigBp <- rep(numA+1,numB) # Preferencia de cada proponente
# asignado a la persona propuesta
asigBs <- rep(0,numB)       # Asignación de cada persona propuesta
noasignados <- 1:numA       # Vector de proponentes no emparejados
while(!all(asigAs != 0)){  # Mientras alguien que propone no
# esté asignado o tenga aún a quien proponer se repite
  for(candidato in noasignados){ # Para aquellos no asignados
    if (asigAp[candidato] > numB ||
    prefA[candidato,asigAp[candidato]]==0){ # Si el candidato
    # ha superado ya sus propuestas se le asigna -1, es decir,
    # se queda soltero/a
      asigAs[candidato] <- -1
    } else {
       propB <- prefA[candidato,asigAp[candidato]] # Almacenamos
       # la propuesta del candidato según la ronda de propuesta
       # por la que se encuentre
       bondadcandidato <- match(candidato, prefB[propB,])
       # Almacenamos la posición de pref del candidato en la
       # lista de los propuestos
       if(is.na(bondadcandidato)){  # Si el candidato no está
       # en la lista de pref de los propuestos, se le asigna
       # una mala posición
         bondadcandidato <- numA+1
       }
       if(bondadcandidato < asigBp[propB]){ # Si la asignación
       # anterior es peor que la nueva
        asigAs[asigBs[propB]] <- 0  # Dejamos sin asignar al
        # candidato en peor posición
        asigAp[asigBs[propB]] <- asigAp[asigBs[propB]] + 1
        # Aumentamos en uno la ronda de propuesta del candidato
        # rechazado
        asigBp[propB] <- bondadcandidato  # Almacenamos la nueva
        # posición del que propone en la lista de los que son
        # propuestos
        asigBs[propB] <- candidato  # Asignamos el candidato a la
        # propuesta del mismo
        asigAs[candidato] <- propB  # Asignamos la propuesta al
        # candidato
```

```
       } else {
        asigAp[candidato] <- asigAp[candidato] + 1  # Aumentamos
        # en uno la ronda de propuesta del candidato
       }
     }
   }
   noasignados <- which(asigAs == 0)  # Almacenamos aquellos
   # que proponen que no han sido asignados
}
names(asigAs) <- 1:numA
print('La asignación final viene dada por:'); print(asigAs)
}
```

Ejemplo 1.18. Vamos a tomar un conjunto de 5 hombres H y un conjunto de 4 mujeres M con las siguientes listas de preferencias y sus matrices asociadas:

```
# Listas de preferencias de los hombres
h_1 <- c(2,4,5,1,3,0)
h_2 <- c(4,6,1,3,2,5)
h_3 <- c(1,2,4,3,0,0)
h_4 <- c(2,4,0,0,0,0)
h_5 <- c(2,1,4,3,0,0)
# Listas de preferencias de las mujeres
m_1 <- c(4,5,3,2,1)
m_2 <- c(5,1,2,4,3)
m_3 <- c(3,4,1,5,2)
m_4 <- c(1,5,2,3,4)
m_5 <- c(2,3,1,5,4)
m_6 <- c(2,5,1,4,3)
# Matrices de preferencias de los hombres y las mujeres
prefA <- rbind(h_1,h_2,h_3,h_4,h_5); prefA
prefB <- rbind(m_1,m_2,m_3,m_4,m_5,m_6); prefB
```

```
A matrix: 5 × 6 of type dbl
h_1     2 4 5 1 3 0
h_2     4 6 1 3 2 5
h_3     1 2 4 3 0 0
h_4     2 4 0 0 0 0
h_5     2 1 4 3 0 0
```

```
A matrix: 6 × 5 of type dbl
m_1     4 5 3 2 1
```

```
m_2         5 1 2 4 3
m_3         3 4 1 5 2
m_4         1 5 2 3 4
m_5         2 3 1 5 4
m_6         2 5 1 4 3

# Aplicamos ahora la función
AsigMatrimonio2(prefA,prefB)
[1] "La asignación final viene dada por:"
[1]  1  2  3  4  5
[1]  4  6  1 -1  2
```

Es decir, el emparejamiento propuesto sería $\{(h_1, m_4), (h_2, m_6), (h_3, m_1), (h_5, m_2)\}$ quedando soltero el cuarto hombre y la tercera y quinta mujer. Con este ejemplo podemos ver que con nuestra generalización del problema, aunque tengamos un número de mujeres mayor al número de hombres, estos podrían quedar solteros si así lo quisieran.

2

Problema de asignación de habitaciones

En el capítulo 1 pudimos ver que para el problema de asignación de matrimonios siempre nos será posible encontrar una solución estable cuando hablamos de emparejar dos conjuntos disjuntos. En este capítulo veremos si es posible encontrar una solución estable cuando emparejamos elementos de un único conjunto. Ya Gale y Shapley comentaron en su artículo el ejemplo de alojar en habitaciones de dos a un grupo de estudiantes.
Como en el problema anterior, cada persona deberá ordenar en una lista de preferencias al resto, donde el primer lugar lo ocupará aquel que sea considerado el favorito y el último lugar lo ocupará el menos deseado.
El objetivo será encontrar un compañero a cada persona de modo que la asignación final sea estable. La necesidad de resolver un problema de este estilo puede surgir en múltiples contextos: alojamiento de universitarios en habitaciones dobles, creación de grupos de trabajo, organización de enfrentamientos deportivos, etc.
La principal diferencia respecto al problema de asignación de matrimonios es que no siempre existirá una asignación que nos proporcione un emparejamiento estable para un conjunto de personas y sus preferencias. En 1985, Robert W. Irving [2] propuso un algoritmo en dos fases que nos proporcionará un emparejamiento estable si este existiera.

2.1. Fase I del algoritmo de Irving

Antes de empezar con las distintas fases debemos tener claro que los términos de pareja bloqueante y emparejamiento estable son los mismos que para el problema de asignación de matrimonios, por lo que las definiciones 1.3 y 1.4 del primer capítulo son válidas, solo hay que tener en cuenta que ahora hablamos de un solo conjunto.
Como se ha considerado anteriormente, en este tipo de problemas no siempre nos encontraremos con una solución estable, por lo que a la hora de abordarlos podemos encontrarnos con dos casos:

Definición 2.1. Diremos que un problema es *resoluble* si podemos encontrar un emparejamiento estable, en otro caso, diremos que el problema es *irresoluble*.

Veamos un ejemplo donde no podemos encontrar una solución estable y por lo tanto el problema es irresoluble.

Ejemplo 2.2. Sea $H = \{A, B, C, D\}$ un conjunto de estudiantes que quieren organizarse en habitaciones de dos personas cada una. En la siguiente tabla podemos ver las listas de preferencias de cada individuo. Al encontrarnos ante un

$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{C}$	$\boldsymbol{D}$
B	C	A	A
C	A	B	B
D	D	D	C

Tabla 2.1. Preferencias del ejemplo 2.2

conjunto con un cardinal $|H| = 4$ solo podemos encontrar tres emparejamientos posibles $G_1 = \{(A, B), (C, D)\}$, $G_2 = \{(A, C), (B, D)\}$ y $G_3 = \{(B, C), (A, D)\}$, veamos que ninguno de estos emparejamientos es estable:

- En G_1, encontramos que la pareja (B, C) es una pareja bloqueante. Por lo que G_1 no es un emparejamiento estable pues B prefiere a C antes que el asignado (A), y C prefiere a B antes que el asigando (D).
- De igual manera en G_2 encontramos que la pareja (A, B) es bloqueante, por lo que G_2 no es un emparejamiento estable.
- Y en G_3, la pareja (A, C) es bloqueante, por lo que G_3 no es un emparejamiento estable.

Por lo tanto, podemos afirmar que no hay emparejamiento estable posible para el conjunto H con sus listas de preferencias actuales y por tanto este problema es irresoluble.

Sin embargo, no todos los problemas son irresolubles, hay situaciones en las que sí nos será posible encontrar un emparejamiento estable. Irving ideó un algoritmo en dos fases capaz de encontrar un emparejamiento estable siempre y cuando exista.
La Fase I se divide en dos etapas, vamos a ver cuáles son los pasos de la primera etapa que denotaremos "Etapa I.1":

Paso 1: Originalmente todos los elementos comienzan sin ninguna propuesta. A continuación, cada uno propone a la persona que tenga mejor situada en su lista de preferencia, siempre y cuando no haya sido rechazado con anterioridad o no le hayan aceptado alguna propuesta.

Paso 2: Cada persona que haya recibido una propuesta en el paso anterior:
- Si aún no había aceptado ninguna propuesta, acepta la de la persona que tenga en mejor posición en su lista de preferencias, rechazando a todos los demás.
- Si había aceptado una propuesta, acepta la de la persona que más prefiera de entre todos los que le han propuesto, incluyendo la propuesta que había aceptado provisionalmente, rechazando todas las demás propuestas.

Paso 3: Repetimos los pasos 1 y 2 hasta que se dé alguna las dos condiciones siguientes, en cuyo caso se termina la Etapa I.1:
- Todas las personas han aceptado una propuesta, por lo tanto, pasaríamos a la Etapa de reducción I.
- Alguna persona ha sido rechazada por todos los demás, entonces no existe emparejamiento estable para el problema.

Vamos a ver ahora unos cuantos resultados que hacen referencia a la Etapa I.1 que nos ayudarán a entender la parte final de esta fase, la cual consiste en reducir las listas de preferencias de nuestros elementos eliminando a aquellos que no serán relevantes para la Fase II.

Lema 2.3. *Si y rechaza a x durante la Etapa I.1 del algortimo entonces x e y no serán pareja en ningún emparejamiento estable.*

Demostración. Vamos a suponer que el primero de todos los rechazos es el de y a x y suponemos por reducción al absurdo que existe un emparejamiento estable α tal que x e y son compañeros. Si y rechaza a x es porque, o bien acaba de recibir o ya tenía una propuesta mejor de otra persona, la cual denotaremos z. Pero si $z \succ_y x$ entonces por la estabilidad de α, como el par (x, y) son compañeros, la pareja de z no es y, por tanto z debe preferir a su compañero en α, el cual denotaremos t, antes que a y. Finalmente, antes de que z pueda hacer una propuesta a y, este tiene que haber sido rechazado por t, lo cual se contradice con la hipótesis de partida. □

Con este Lema podemos dar lugar a un gran número de corolarios, como por ejemplo los dos siguientes.

Corolario 2.4. *Si en cualquier momento de la secuencia de proposiciones x le hace una propuesta a y, entonces en cualquier emparejamiento estable:*

1. *x no puede tener mejor compañero que y.*
2. *y no puede tener peor compañero que x.*

Demostración. Si x propone a y, esto quiere decir que es su primera opción o ya ha sido rechazado por todos los que están mejor situados que y, por tanto, por el Lema 2.3, ninguna persona que rechaza a otra pueden ser compañeros en un emparejamiento estable. Con lo cual hemos probado 1.
Si y tiene a z como compañero en un emparejamiento estable α y $z \succ_y x$, por

tanto, por 1, x prefiere a z sobre su propio compañero, lo que contradice la estabilidad de α. Por tanto hemos probado 2. □

Corolario 2.5. *Si algún elemento ha sido rechazado por todos los demás al terminar la Etapa I.1 del algoritmo, entonces no existe emparejamiento estable.*

Demostración. Por el Lema 2.3, la persona rechazada no puede tener ningún compañero en un emparejamiento estable, por lo que nunca existirá un emparejamiento estable. □

Una vez ya hemos visto estos dos últimos resultados, podemos dar paso al final de esta primera fase, la Etapa de Reducción I, la cual, como hablamos antes, se encarga de eliminar aquellos elementos que no son importantes para la Fase II.

Corolario 2.6. ***(Etapa de Reducción I)*** *Si la Etapa I.1 del algoritmo termina con que cada persona tiene una propuesta, entonces la lista de preferencias de un x, cuya propuesta es de y, puede reducirse eliminando a las siguientes personas.*

1. *Todo z tal que $y \succ_x z$.*
2. *Todo z tal que su pareja provisional, t, sea anterior a x en su lista de preferencias (incluyendo todos aquellos que han rechazado a y).*
   En las listas de preferencias reducidas resultantes:
3. *x está el primero de y, e y está el último en la lista de x.*
4. *En general, z aparece en la lista de t si y sólo si t aparece en la de z.*

Demostración. 1 y 2 siguen directamente del Corolario 2.4, (2); 3 sigue de 1 y de la parte entre paréntesis de 2, es decir, y está el último en la lista de x ya que, por el apartado 1, se eliminan todos aquellos que estén detrás de y; x está el primero en la lista de y por el apartado 2, eliminamos aquellos que rechazaron a y. 4 es trivial, debido a que un si un elemento no está en la lista de preferencias de otro, esa pareja no se va a dar en el algoritmo. □

Lema 2.7. *Si al finalizar la Etapa de Reducción I las listas de preferencias solo cuentan con una persona cada una, el algoritmo termina con un emparejamiento estable.*

Demostración. Que las listas especifiquen un emparejamiento es consecuencia del Corolario 2.6 (4). Supongamos que x prefiere a y a la única persona que está en su lista. Entonces x tuvo que ser rechazado por y. Pero la propuesta final de y es de la única persona en la lista reducida de y, por lo que y prefiere a esta persona a x. Por lo tanto, no puede haber inestabilidad en el emparejamiento resultante por las listas de preferencias reducidas. □

Vamos a ver ahora un ejemplo de la Fase I del algoritmo de Irving.

Ejemplo 2.8. Vamos a tomar un conjunto $H = \{A, B, C, D, E, F\}$ de personas que quieren buscar un compañero de cuarto. Se les pide a todos que expresen en una lista sus preferencias respecto al resto de candidatos, dando así lugar a la siguiente tabla de preferencias:

A	***B***	***C***	***D***	***E***	***F***
E	A	E	B	D	A
C	C	F	A	F	D
D	E	D	F	A	E
F	F	B	C	C	B
B	D	A	E	B	C

Tabla 2.2. Lista de preferencias del ejemplo 2.8

Una vez ya tenemos las preferencias de cada persona podemos comenzar con la Etapa I.1 del algoritmo de Irving. Iniciamos con la ronda donde cada individuo hará su propuesta a la persona que ocupe la primera posición en la lista. Si una persona recibe más de una propuesta, eligirá cuál conserva y cuáles descarta, esto se repetirá hasta que todos hayan recibido una propuesta o alguien ha sido descartado por todos.

A: E
B: $\not{A}\ C$
C: $\not{E}\ F$
D: B
E: D
F: A

Podemos observar que nadie ha sido descartado por todos los demás, por tanto podemos pasar a la Etapa de Reducción I.
Empezamos eliminando aquellas personas que han sido descartadas en la primera etapa y seguido eliminamos a sus simétricos, ya que son parejas que nunca se van a poder dar, es decir, eliminamos a A de la lista de B y eliminamos a E de la lista de C y viceversa.
Siguiendo el Corolario 2.6, debemos eliminar a A de la lista de C y a B de la lista de E, también a sus simétricos.

A	***B***	***C***	***D***	***E***	***F***
E	$\not{A}$	$\not{E}$	B	D	A
$\not{C}$	C	F	A	F	D
D	$\not{E}$	D	F	A	E
F	F	B	C	$\not{C}$	B
$\not{B}$	D	$\not{A}$	E	$\not{B}$	C

Finalmente, hemos terminado la Fase I y obtenemos la siguiente tabla de preferencias reducidas:

A	***B***	***C***	***D***	***E***	***F***
E	C	F	B	D	A
D	F	D	A	F	D
F	D	B	F	A	E
			C		B
			E		C

Dicha tabla realmente no presenta un emparejamiento, pero hemos conseguido reducir en cierta medida algunas listas de preferencias.

2.2. Fase II del algoritmo de Irving

Como hemos podido ver en el ejemplo anterior, la Fase I del algoritmo puede no terminar en un emparejamiento por lo que es necesario una segunda fase donde podremos conseguir, si existe, un emparejamiento estable.
Esta Fase II consiste en un proceso iterativo que nos permitirá seguir reduciendo estas listas, para ello necesitamos conocer unos resultados antes.

Definición 2.9. Diremos que un conjunto de listas de preferencias está *reducido* si cumple las siguientes condiciones:

- Han pasado por la Etapa de Reducción I.
- Han pasado por cero o más reducciones de la Fase II.

Esta segunda fase consiste en generar secuencias cíclicas de personas $a_1a_2...a_r$ tales que:

- Para $i = 1, ..., r-1$, la segunda persona de la actual lista reducida de a_i será la primera persona de la de a_{i+1}; denotaremos a esa persona como b_{i+1}.
- La segunda persona de la actual lista de preferencias reducida de a_r es la primera de la de a_1; denotaremos a esa persona como b_1.

Para esta Fase II vamos a necesitar usar unos ciclos denominados "todo o nada". Estos ciclos nos permitirán acortar aún más las listas reducidas que obtengamos. Bastaría tomar un individuo p_i cuya lista de preferencias reducida tenga dos o más personas, y generar las siguientes secuencias p y q:

- q_i = Segunda persona de la actual lista reducida de p_i.
- p_{i+1} = Última persona de la actual lista reducida de q_i.

Continuamos generando las secuencias hasta que encontremos un ciclo en la sucesión p (se repita un p_s), y entonces definiremos:

$$a_i = p_{s+i-1} \qquad (i = 1, 2, ...)$$

La Fase II del algoritmo de Irving toma como base las listas de preferencias reducidas de la Fase I y se va desarrollando iterativamente. Cada iteración cuenta con los siguientes pasos:

Paso 1: (Etapa de Reducción II): Generamos un ciclo "todo o nada" $a_1...a_r$ y forzamos a cada b_i $(1 \leq i \leq r)$ a rechazar la propuesta que tenía de a_i, obligando a cada a_i a hacerle una propuesta a b_{i+r} (módulo r), siendo este la segunda persona de su actual lista reducida. Con esto conseguimos eliminar a a_i de la lista reducida de b_i y al simétrico, reduciendo de esta manera las listas.

Paso 2: Aplicamos ahora la Etapa de Reducción I a las nuevas listas reducidas de la Etapa de Reducción II. Debemos tener en cuenta las propuestas aceptadas en el paso anterior.

Paso 3: Repetimos los pasos 1 y 2 hasta que nos encontremos con una de las dos siguientes posibilidades:

- Alguna lista se ha quedado vacía, por lo que no tenemos solución estable.
- Todas las listas reducidas solo contienen una sola persona, por lo que tenemos el emparejamiento estable generado por el algoritmo.

Ejemplo 2.10. Continuando con el Ejemplo 2.8, retomamos ahora con la Fase II del algoritmo con la siguiente tabla de listas reducidas:

A	***B***	***C***	***D***	***E***	***F***
E	C	F	B	D	A
D	F	D	A	F	D
F	D	B	F	A	E
			C		B
			E		C

A:	E
B:	C
C:	F
D:	B
E:	D
F:	A

Seguimos entonces aplicando ciclos "todo o nada" para poder reducir las listas de preferencias.

Paso 1: Tomamos $p_1 = A$, ya que A cuenta con más de una persona en su lista reducida. Vamos a crear el ciclo "todo o nada" a partir de la Definición 2.10.

$$\begin{array}{ll} q_1 = D & p_2 = E \\ q_2 = F & p_3 = C \\ q_3 = D & p_4 = E \end{array}$$

Hemos encontrado que $p_2 = p_4$, por lo que tomamos $s = 2$, entonces podemos definir:

$$\begin{array}{ll} a_1 = p_2 = E & b_1 = D \\ a_2 = p_3 = C & b_2 = F \\ a_3 = p_2 = E & b_3 = D \end{array}$$

Tenemos un ciclo "todo o nada" ECE, por lo que ahora vamos a forzar a que D rechace a E y que F rechace la propuesta de C. Por tanto E le hace una propuesta a F y C le hace una a D. Las nuevas listas reducidas por la Fase de Reducción II y la nueva lista de propuestas quedarían de la siguiente manera:

A	***B***	***C***	***D***	***E***	***F***
E	C	~~F~~	B	~~D~~	A
D	F	D	A	F	D
F	D	B	F	A	E
			C		B
			~~E~~		~~C~~

A:	E
B:	C
C:	~~F~~ D
D:	B
E:	~~D~~ F
F:	A

Paso 2: Aplicamos ahora la Etapa de Reducción I a la lista reducida obtenida en el paso anterior, es decir, por el Corolario 2.6 vemos que hay que eliminar a B de la lista de F, y por simetría, eliminamos a F de la lista de B.

A	***B***	***C***	***D***	***E***	***F***
E	C	D	B	F	A
D	~~F~~	B	A	A	D
F	D		F		E
			C		~~B~~

Paso 3: Podemos observar que no hay ninguna lista vacía, por lo que debemos repetir los pasos 1 y 2.

Paso 1*: Tomamos ahora de nuevo $p_1 = A$, ya que A aún cuenta con dos personas en su lista reducida.

$$\begin{array}{ll} q_1 = E & p_2 = C \\ q_2 = B & p_3 = D \\ q_3 = A & p_4 = F \\ q_4 = D & p_5 = C \end{array}$$

Hemos encontrado que $p_2 = p_5$, por lo que tomamos $s = 2$, entonces podemos definir:

$$\begin{array}{ll} a_1 = p_2 = C & b_1 = D \\ a_2 = p_3 = D & b_2 = B \\ a_3 = p_4 = F & b_3 = A \\ a_4 = p_5 = C & b_4 = D \end{array}$$

Tenemos un ciclo "todo o nada" CDFC, por tanto, forzamos que D rechace a C, que B rechace a D y que A rechace a F, así como a sus simétricos. Obtenemos de esta manera:

A	***B***	***C***	***D***	***E***	***F***
E	C	~~D~~	~~B~~	F	~~A~~
D	~~D~~	B	A	A	D
~~F~~			F		E
			~~C~~		

A:	E
B:	C
C:	~~D~~ B
D:	~~B~~ A
E:	F
F:	~~A~~ D

Paso 2*: Aplicamos ahora la Etapa de Reducción I, pero no tendríamos ningún cambio.

A	***B***	***C***	***D***	***E***	***F***
E	C	B	A	F	D
D			F	A	E

Paso 3*: Aún quedan listas reducidas con más de una persona por lo que debemos volver a repetir los pasos 1 y 2 con la lista reducida del paso anterior.

Paso 1**: Como aún la lista de A cuenta con más de una persona, elegimos $p_1 = A$.

$$\begin{array}{ll} q_1 = D & p_2 = F \\ q_2 = E & p_3 = A \end{array}$$

Hemos encontrado que $p_1 = p_3$, por lo que tenemos que $s = 1$, entonces podemos definir:

$$\begin{aligned} a_1 &= p_1 = A & b_1 &= E \\ a_2 &= p_2 = F & b_2 &= D \\ a_3 &= p_3 = A & b_3 &= E \end{aligned}$$

Por tanto tenemos un ciclo "todo o nada" AFA, forzamos a E a rechazar a A y a D que rechace a F, así como a sus simétricos.

A	***B***	***C***	***D***	***E***	***F***
~~*E*~~	*C*	*B*	*A*	*F*	~~*D*~~
D			~~*F*~~	~~*A*~~	*E*

A:	~~*E*~~ *D*
B:	*C*
C:	*B*
D:	*A*
E:	*F*
F:	~~*D*~~ *E*

Paso 2**: Aplicamos ahora la Etapa de Reducción I, pero no tendríamos ningún cambio.

Paso 3**: Finalmente, en cada lista reducida hemos encontramos una persona por lo que terminamos el algoritmo encontrando un emparejamiento estable, organizando así a las 6 personas en habitaciones dobles de la siguiente manera:

$$\{(A, D), (B, C), (E, F)\}$$

Ahora vamos a ver un Lema que trata algunas propiedades en relación a los elementos de los ciclos "todo o nada".

Lema 2.11. *Sea $a_1 ... a_r$ un ciclo "todo o nada" relativo a un conjunto de listas de preferencias reducidas, y sea b_i la primera persona de la lista reducida de a_i $(1 \leq i \leq r)$. Entonces:*

1. *En cualquier emparejamiento estable contenido en esas listas reducidas, tanto a_i como b_i o están emparejados para todo valor de i, o no lo están para ningún valor de i.*
2. *Si hay un emparejamiento estable en el que a_i y b_i son compañeros, entonces existe otro en el que no lo son.*

Demostración. Vamos a demostrar (1), consideramos los subíndices módulo r, y supongamos que para algún i fijado, el par (a_i, b_i) es una pareja de compañeros de habitación en algún emparejamiento estable contenido en las listas reducidas. Sabemos que a_i es la última persona en la lista reducida de b_i, pero también sabemos que b_i es la segunda persona en la lista de a_{i-1}. Por tanto, $a_{i-1} \succ_{b_i} a_i$. Entonces a_{i-1} prefiere a su compañero antes que a b_i, pues en caso contrario no

se daría la estabilidad, y la única persona que se encuentra delante en su lista reducida es b_{i-1}. Si repetimos sucesivamente este razonamiento llegamos a que para todos los valores de i, a_i y b_i son compañeros en el emparejamiento estable. Seguimos ahora con (2). Sean $A = \{a_1, ..., a_r\}$ y $B = \{b_1, ..., b_r\}$. Si $A \cap B \neq \emptyset$, decimos que $a_j = b_k$, luego es imposible que todos los a_i tengan su primera opción restante, ya que b_k tiene su última cuando a_k tiene su primera. Por lo tanto, por (1), $A \cap B \neq \emptyset$ implica que ninguno de los a_i y b_i pueden ser compañeros, por lo que podemos asumir que $A \cap B = \emptyset$.
Suponemos ahora que α es un emparejamiento estable, contenido en las listas reducidas, en las cuales para todo $1 \leq i \leq r$, a_i y b_i son compañeros. Denotamos por β el emparejamiento en el que cada a_i está emparejado con b_{i+1}, y cualquier persona que no esté en $A \cup B$ está emparejado con la misma persona que en α. Podemos decir que β es estable.
Como es evidente, cada persona de B obtiene una mejor pareja en β, desde su punto de vista, una mejor que la que tenía en α. A las personas de A son a los únicos a los que les va peor en β que en α, por lo que cualquier inestabilidad en β, que no esté presente en α, debe implicar a algún a_i.
Si a_i prefiere a cierto x antes que a b_{j+1} (su compañero en β), entonces solo debemos considerar tres casos:

1. a_i y x eran compañeros en α (i.e., $x = b_i$); pero en este caso, x prefiere sin lugar a dudas a su nuevo compañero a_{i-1} a a_i.
2. a_i también prefiere a x antes que a b_i, en cuyo caso x no aparece en la lista reducida de preferencias de a_i, donde él precedería a la primera opción restante conocida de a_i, es decir, a b_i. Entonces x ha rechazado voluntariamente o ha sido forzado a rechazar a a_i. En el primer caso, x debe haber recibido una mejor propuesta que la de a_i, mientras que, en el segundo caso, el rechazo forzado debe haber llevado a que x reciba una mejor propuesta. En cualquier caso debe prefereir incluso a su última opción restante, y por lo tanto, ciertamente a su pareja en β antes que a a_i.
3. a_i prefiere a b_i a x, en cuyo caso, x se encuentra entre b_i y b_{i+1} en la lista de preferencias original de a_i, pero no se encuentra en la lista reducida actual de a_i. Esta ausencia debe deberse a que x ha recibido una mejor propuesta de alguien mejor que a_i, al igual que en el caso anterior, x debe preferir a su pareja en β antes que a a_i. □

Gracias al Lema 2.12 obtenemos directamente los dos siguientes corolarios.

Corolario 2.12. *Si el problema original admite un emparejamiento estable, entonces hay un emparejamiento estable contenido en cualquier conjunto de listas reducidas.*

Corolario 2.13. *Si una o más listas de un conjunto de listas de preferencias reducidas están vacías, entonces el problema original no admite ningún emparejamiento estable.*

El Lema siguiente es una extensión del Lema 2.12, justificando la circunstancia bajo la cual el algoritmo llega a una solución estable.

Lema 2.14. *Si en un conjunto de listas de preferencias reducidas, toda lista contiene a una sola persona, entonces estas listas determinan un emparejamiento estable.*

Demostración. Que las listas determinen un emparejamiento es consecuencia del Corolario 2.6. Suponemos que x prefiere a y antes que a la única persona de su lista reducida, por tanto, sabemos que y prefiere a la única persona de su lista reducida sobre x, es decir, tal emparejamiento no puede ser inestable, luego se trata de un emparejamiento estable. □

3

Problema de asignación de universidades

El problema de asignación de universidades es un problema clásico en la teoría de emparejamientos, el cual se basa en cómo asignar a un grupo de estudiantes a universidades de manera óptima y justa, teniendo en cuenta las preferencias tanto de los estudiantes como de las universidades.
En este caso y al igual que en el capítulo 1, nos encontramos ante dos conjuntos disjuntos S y U que llamaremos alumnos y universidades, respectivamente. A diferencia de nuestros capítulos anteriores, en este caso cada universidad cuenta con un cupo q_i de alumnos que puede aceptar, por tanto, este problema no será del tipo "uno a uno", sino de "muchos a uno".
En este capítulo, exploraremos en detalle cómo funciona el algoritmo de Gale-Shapley en el contexto del problema de asignación de universidades y cómo se ha aplicado a diferentes escenarios. También examinaremos otras soluciones propuestas por la teoría de emparejamientos y cómo se comparan con el algoritmo de Gale-Shapley.

3.1. Algoritmo de Gale-Shapley para Universidades

Este problema de asignación es una variante del problema de matrimonios, por lo que podemos tomar el concepto de estabilidad como el que ya conocemos. Además, no será necesario que el número de elementos de S coincida con el de U. Cada sujeto de S debe ordenar en una lista de preferencias a las universidades de modo que no se tengan repeticiones y no hará falta que en la lista aparezcan todas, solo aquellas a las que está despuesto a ir. Del mismo modo, las universidades también tendrán sus listas de preferencia con los alumnos a los que sí podrían aceptar, obviando a aquellos que no. Estas listas también deberán ser ordenadas y estrictas.
Una vez conocidas las preferencias de todos los elementos de nuestros dos conjuntos, podemos comenzar el algoritmo:

Paso 1: Los alumnos proponen a aquellas universidades que tengan en primer lugar en sus listas de preferencia.

Paso 2: Las universidades dejarán en lista de espera a los alumnos siempre y cuando su cupo q_i no esté lleno, cuando esto ocurra, las universidades deberán revisar la lista de espera y rechazar a aquellos alumnos que consideren peores.

Paso 3: Los alumnos rechazados en el paso anterior deben proponer ahora a la siguiente universidad que tengan en su lista de preferencia.

Paso 4: Repetiremos los pasos 2 y 3 hasta que se dé una de las dos condiciones siguientes:

- Todos los alumnos están en alguna de las listas de espera de las universidades, con lo cual todos los alumnos en dichas listas son aceptados y el emparejamiento resultante es estable.
- Los alumnos pendientes han sido rechazados por todas las universidades. En este caso también tenemos un emparejamiento estable.

Vamos ahora a ver un ejemplo de cómo funciona del algoritmo de Gale y Shapley.

Ejemplo 3.1. Para nuestro ejemplo se pretende emparejar a cuatro estudiantes $S = \{A, B, C, D\}$ con dos universidades $U = \{X, Y\}$ donde el cupo de ambas universidades es de dos alumnos, sus listas de preferencias son las siguientes:

A	***B***	***C***	***D***
Y	X	Y	Y
X	Y	X	X

X	***Y***
B	A
D	B
A	C
C	D

Ya podemos dar comienzo al algoritmo,

Paso 1: Los alumnos A, C y D proponen a la universidad Y, mientras que B propone a X.

Paso 2: Y ha recibido más propuestas de las que puede asumir con su cupo y debido a que $A, C \succ_Y D$, decide rechazar a D y aceptar a A y C en su lista de espera. Por otro lado, X decide aceptar en su lista de espera a B.

Paso 3: Como D ha sido rechazado en el paso anterior, ahora propone a su siguiente universidad, es decir, a X.

Paso 4: X deja en lista de espera a D junto a B.

Finalmente, todos los alumnos están en la lista de espera de una universidad, por lo que estas aceptan a todos los estudiantes en ellas dando lugar a un emparejamiento estable.

Universidad X	B	D
Universidad Y	A	C

Con este algoritmo siempre podemos generar un emparejamiento estable, vamos a demostrarlo en el siguiente teorema.

Teorema 3.2. *El algoritmo de Gale y Shapley para el problema de universidades siempre genera un emparejamiento estable.*

Demostración. Procederemos por reducción al absurdo suponiendo que existe una pareja bloqueante (s, u) en un emparejamiento α resultante de aplicar el algoritmo, es decir, el alumno s no ha sido aceptado por la universidad u cumpliéndose alguna de estos casos:

Caso a: No se ha cubierto el cupo de u y además, s no ha sido aceptado por ninguna otra universidad. Este caso no puede suceder ya que si u no ha llenado el cupo, en algún momento del algoritmo s le ha propuesto a u, ya que esta debe figurar en la lista de preferencias del alumno.

Caso b: No se ha cubierto el cupo de u y además, s prefiere a u antes que a la universidad a la que ha sido asignado. Este caso tampoco puede suceder: si el cupo de u está incompleto y s la prefiere antes que a la que ha sido asignado, este le habría propuesto antes, por lo que hubiera sido aceptado.

Caso c: u prefiriere antes a s que a algún otro alumnno que haya sido aceptado en α y s no ha sido aceptado por ninguna universidad. Si durante el algoritmo u ha decidido aceptar a otro alumno antes que a s es porque lo prefiere, lo que contradice la hipótesis de este caso.

Caso d: u prefiere antes a s que a algún otro alumno que haya sido aceptado en α y s prefiere a u antes que a la universidad a la que ha sido asignado. Si s prefiere a u le tendría que haber pedido ser admitido antes que a la universidad en la que ha terminado finalmente, y si este ha sido rechazado por u, eso contradice la hipótesis de este caso donde se dice que u prefiere a s antes que a algún otro alumno.

Por lo que podemos afirmar que no puede llegar a existir ninguna pareja bloqueante, y por lo tanto, el emparejamiento obtenido por el algoritmo es estable. □

3.2. Variante del algoritmo Gale-Shapley

A lo largo del tiempo se han desarrollado distintos algoritmos para resolver este problema, uno de los más conocidos es el algoritmo de Boston, ideado por Alvin E. Roth y otros economistas en la década de 1980. Roth fue uno de los

principales investigadores en el campo de la teoría de emparejamiento y desarrolló varias mejoras y variaciones del algoritmo a lo largo de los años. Su trabajo en el algoritmo de Boston y en la teoría de emparejamiento en general le valió el Premio Nobel de Economía en 2012, junto con Lloyd Shapley ya que Gale había fallecido en 2008. Si bien ambos algoritmos buscan emparejar a los estudiantes con las universidades de acuerdo con sus preferencias, difieren en su enfoque y en los resultados que producen. Mientras que el algoritmo de Gale y Shapley se centra en garantizar la estabilidad del emparejamiento, el algoritmo de Boston se enfoca en maximizar la satisfacción de los estudiantes y las universidades.
Vamos a ver cuáles son los pasos de este algoritmo:

Paso 1: Los estudiantes presentan una solicitud a un conjunto de universidades, clasificando las universidades en orden de preferencia.

Paso 2: Cada universidad tiene un número limitado de plazas disponibles. Las universidades evalúan las solicitudes de los estudiantes y seleccionan a los estudiantes que cumplen sus criterios de admisión.

Paso 3: Las universidades envían ofertas condicionales a los estudiantes seleccionados. Las ofertas condicionales se basan en el orden de preferencia de los estudiantes, y los estudiantes reciben ofertas de las universidades que se encuentran en su lista de preferencias.

Paso 4: Los estudiantes pueden aceptar o rechazar las ofertas condicionales de las universidades. Si un estudiante acepta una oferta de una universidad, automáticamente rechaza las ofertas de todas las universidades que se encuentran por debajo en su lista de preferencias.

Paso 5: Si un estudiante acepta una oferta de una universidad, la universidad reserva una de sus plazas disponibles para ese estudiante. Si todos los estudiantes han aceptado ofertas de universidades, el proceso de asignación finaliza. Si no, se procede al paso 6.

Paso 6: Si quedan plazas disponibles en las universidades después de la primera ronda de ofertas, las universidades hacen nuevas ofertas condicionales a los estudiantes que no han recibido ninguna oferta. Estas ofertas se basan en la lista de preferencias de los estudiantes y las plazas disponibles en las universidades.

Paso 7: Los estudiantes pueden aceptar o rechazar las nuevas ofertas condicionales. Si un estudiante acepta una oferta de una universidad en esta ronda, automáticamente rechaza las ofertas de todas las universidades que se encuentran por debajo en su lista de preferencias.

Paso 8: El proceso se repite hasta que se han asignado todas las plazas disponibles en las universidades.

Es importante destacar que el algoritmo de Boston es un algoritmo basado en la oferta y la demanda, que se centra en las preferencias de los estudiantes y las plazas disponibles en las universidades. A diferencia del algoritmo de Gale-

Shapley, el algoritmo de Boston no garantiza la estabilidad del emparejamiento. Para demostrar que no siempre se genera un emparejamiento estable veamos el siguiente ejemplo.

Ejemplo 3.3. Sean $S = \{A, B, C, D, E\}$ los alumnos y $U = \{X, Y, Z\}$ las universidades con un cupo de $q_X = q_Y = 2$ y $q_Z = 1$. Vamos a utilizar el algoritmo de Boston para generar un emparejamiento haciendo uso de las siguientes listas de preferencia:

A	***B***	***C***	***D***	***E***
X	Z	Y	Y	Y
Y	Y	Z	Z	X
Z	X	X	X	Z

X	***Y***	***Z***
A	B	C
E	C	E
D	E	D
B	A	B
C	D	A

El algoritmo de Boston procedería de la siguiente manera.

- Los alumnos empiezan con sus propuestas a las universidades que tengan en primera posición, es decir, A manda su solicitud a la universidad X, B se la envía a Z y C, D y E a Y.
- Las universidades revisan las propuestas recibidas y Y es la única que debe elegir ya que le han enviado más de las que puede aceptar. Como D es el que más abajo está en la lista de preferencias de la universidad, es descartado y Y acepta a C y D. Por otro lado, X acepta a A y Z a B.
- Como el único rechazado es D, envía su propuesta a la siguiente universidad de su lista, en este caso, a Z.
- La universidad Z ya tiene su cupo lleno por lo que no puede aceptar la propuesta de D.
- Debido a que han vuelto a rechazar a D, este tiene que enviar ahora la propuesta a su última opción, es decir, a X.
- En este caso X puede aceptar la propuesta de D ya que su cupo no está lleno y no ha recibido más solicitudes.

Una vez el algoritmo ha terminado, hemos obtenido la asignación:

Universidad X	A	D
Universidad Y	C	E
Universidad Z	B	

Es fácil comprobar que este emparejamiento es inestable pues (D, Y) es una pareja bloqueante, esto se debe a que el alumno D quedó con su peor opción

debido a que había postulado primero a Y. Si hubiera mentido en sus preferencias indicando que preferiría a Z, hubiese quedado en esa universidad.

Con el ejemplo anterior hemos podido ver que este algoritmo no siempre genera un emparejamiento estable. Es lógico preguntarse entonces por qué utilizar este método en vez de el propuesto por Gale y Shapley. Cada uno tiene sus fortalezas y sus debilidades.
En cuanto al algoritmo de Gale y Shapley, entre sus fortalezas se encuentra que garantiza que el emparejamiento resultante es estable; también que cada estudiante tiene la oportunidad de ser asignado a su mejor opción posible, de acuerdo con sus preferencias; y por último, el algoritmo puede manejar diferentes números de estudiantes y universidades, así como tamaños de grupos de estudiantes para cada universidad. La debilidad es que el algoritmo no garantiza la satisfacción máxima de los estudiantes y las universidades, es decir, no asegura que ambos grupos acaben emparejados con sus preferencias más deseadas.
En referencia a las fortalezas del algoritmo de Boston, el algoritmo maximiza la satisfacción de los estudiantes y las universidades, lo que puede conducir a un mejor ajuste de las preferencias y permite que las universidades limiten el número de estudiantes a los que pueden admitir, lo que puede ayudar a prevenir el exceso de demanda. Por otro lado, las debilidades pueden ser que el algoritmo no garantiza que el emparejamiento resultante sea estable y en algunos casos, puede generar asignaciones ineficientes, es decir, parejas de estudiantes y universidades que podrían haber obtenido una mejor opción.
Conocidas las fortalezas y debilidades, la elección entre ellos dependerá del contexto y las necesidades específicas de los dos grupos involucrados.

3.3. Aplicaciones de los algoritmos

Los algoritmos de emparejamiento han sido útiles para resolver problemas de asignación en diferentes campos, desde la asignación de estudiantes a escuelas hasta la de donantes de riñón y residentes en hospitales. El algoritmo de Gale-Shapley y el de Boston son algunos ejemplos de estos algoritmos, los cuales se basan en la idea de permitir que los agentes expresen sus preferencias y asignar los recursos de manera eficiente y justa para todas las partes involucradas. De esta manera, se puede lograr un resultado satisfactorio para todas las partes involucradas.
En esta sección, exploraremos algunas de las aplicaciones más comunes de los algoritmos de emparejamiento, así como algunas de las limitaciones y desafíos asociados con su implementación en diferentes contextos. Veremos cómo estos algoritmos han sido utilizados para mejorar la eficiencia y la equidad en diversas situaciones, y discutiremos algunas de las implicaciones más amplias de su uso

en la toma de decisiones.

3.3.1. Trasplante de órganos

Primero empezaremos hablando de la utilización de estos algoritmos a la hora de los intercambios de órganos, en este caso riñones de pacientes en hospitales. Tanto el algoritmo de Gale-Shapley como el algoritmo de Boston han sido utilizados en el intercambio de riñones, y cada uno tiene sus ventajas y desventajas.

El algoritmo de Gale-Shapley es especialmente útil para el emparejamiento de donantes de riñón vivos y receptores cuando se trata de parejas de donantes y receptores en desacuerdo o no compatibles. En este contexto, el algoritmo de Gale-Shapley puede garantizar que todos los pacientes sean atendidos. Además, permite que los donantes y receptores expresen sus preferencias y se emparejen con la mejor opción disponible para ellos. El algoritmo comienza con cada donante ofreciéndose como posible donante para un receptor en la lista de espera. Los receptores que tienen un donante compatible asignado a ellos son retirados de la lista de espera y la búsqueda continúa con los receptores restantes y los donantes no asignados. El proceso continúa hasta que se ha agotado la lista de espera o se ha encontrado una cadena de intercambio completa que incluye a todos los receptores en la lista de espera.

Por otro lado, el algoritmo de Boston es útil para la asignación de riñones a través de cadenas de intercambio, en las cuales un donante incompatible con su receptor original dona su riñón a un receptor compatible y a su vez recibe un riñón de otro donante. El algoritmo de Boston es capaz de manejar con éxito cadenas de intercambio de mayor complejidad que el algoritmo de Gale-Shapley, y puede tener en cuenta múltiples factores, como la ubicación geográfica y la compatibilidad sanguínea, para maximizar la cantidad de pacientes atendidos.

En resumen, ambos algoritmos tienen aplicaciones útiles en el intercambio de riñones, dependiendo del contexto específico. El algoritmo de Gale-Shapley es mejor para el emparejamiento de donantes y receptores vivos en desacuerdo, mientras que el algoritmo de Boston es más adecuado para la asignación de riñones a través de cadenas de intercambio complejas.

Un ejemplo real del uso de estos algoritmos, en concreto el de Boston, fue en marzo de 2004 donde por la donación de un donante altruista se pudo salvar la vida de seis personas. Se realizaron cadenas de intercambios en cinco hospitales de cuatro comunidades autónomas distintas. Comenzando en el Hospital 12 de Octubre, Madrid, se fueron realizando intercambios puente con otros pacientes y donantes de otros centros hospitalarios hasta, finalmente, en abril del mismo año se encontró un donante compatible para emparejar con el primer paciente de esta cadena. (Fuente: https://culturacientifica.com/2014/10/01/matematica-discreta-y-trasplantes-cruzados-de-rinones/)

3.3.2. Residentes en hospitales

El problema de las plazas de médicos residentes en hospitales es un problema clásico de asignación que se originó en los años 50 y 60 en los Estados Unidos. En este problema, los hospitales tienen una serie de plazas de residencia y los estudiantes de medicina, conocidos como residentes, tienen que elegir un hospital en el que trabajar durante su formación. A su vez, los hospitales también tienen que seleccionar a los residentes que consideren más adecuados para su programa de formación.
Este problema ha sido estudiado extensamente en la literatura de teoría de juegos y algoritmos de emparejamiento. Tanto el algoritmo de Gale y Shapley como el de Boston se han utilizado en la resolución del problema demostrando ser muy útiles en la práctica.
En este problema, se ha encontrado que el algoritmo de Gale y Shapley es mejor opción que el algoritmo de Boston, ya que garantiza un emparejamiento estable y óptimo en términos de preferencias declaradas por los residentes y los hospitales.
El algoritmo de Gale y Shapley permite que los residentes y hospitales declaren sus preferencias y, a partir de esta información, crea parejas estables entre ellos, lo que garantiza que no haya incentivos para que ninguno de ellos cambie de pareja. Por otro lado, el algoritmo de Boston, aunque también permite que los residentes y los hospitales declaren sus preferencias, no garantiza un emparejamiento estable en todos los casos y puede llevar a situaciones donde los residentes y los hospitales tengan incentivos para cambiar de pareja.
En el contexto de los residentes de hospitales, el algoritmo de Gale y Shapley ha demostrado ser más efectivo en la práctica y se ha utilizado en muchos países.
El MIR (Médico Interno Residente) es un examen que se realiza en España para la selección de médicos que realizarán su formación especializada en hospitales públicos. Aunque el proceso de asignación de plazas del MIR tiene ciertas similitudes con el problema de residentes de hospitales, en la actualidad no se utiliza el algoritmo de Gale-Shapley ni el algoritmo de Boston para resolverlo.
El proceso de asignación de plazas del MIR es llevado a cabo por el Ministerio de Sanidad de España y consiste en un sistema informatizado que tiene en cuenta diferentes criterios, como la nota obtenida en el examen, la posición del solicitante en la lista de preferencias de cada hospital, y otras variables. A pesar de que el proceso de asignación del MIR no sigue exactamente los mismos principios que los algoritmos de Gale-Shapley o de Boston, su objetivo final es similar: asignar a cada médico en formación la plaza de residencia que mejor se adapte a sus preferencias y habilidades. (Fuente: https://academiamir.com/el-mir/)

3.4. Implementación en R

En esta sección planteamos una forma de codificar el algoritmo del problema de asignación de universidades en el lenguaje de programación R [7]. Rescataremos la función "unicos" vista en la implementación del algoritmo del problema de matrimonios para usarla en nuestra nueva función. Dicha función tendrá como entrada las listas de preferencias tanto de los alumnos como de las universidades, junto con el cupo de cada universidad. Tomaremos el conjunto A como los alumnos y el B como las universidades, mientras que el cupo será el vector q.
El código está disponible en [8].

```
universidades <- function(prefA,prefB,q){ #Tiene como
# entrada prefA= matriz de preferencia de los alumnos,
# prefB= matriz de preferencia de las universidades y
# q= cupo de las universidades
numA <- dim(prefA)[1]
numB <- dim(prefB)[1]
if (dim(prefA)[2]>numB || dim(prefB)[2]>numA){
  return(print("Hay que revisar las matrices"))
} else {
  if (all(apply(prefA,1,unicos))==FALSE ||
  all(apply(prefB,1,unicos)==FALSE)){
    return(print("Hay que revisar las listas de preferencia"))
  }
}
asigAp <- rep(1,numA)       # Ronda de propuesta de cada alumno
asigAs <- rep(0,numA)       # Asignación de cada alumno
asigBs <- matrix(0,nrow = numB, ncol = numA)       # Asignación
# de cada universidad
noasignados <- 1:numA       # Vector de alumnos no emparejados
  while(!all(asigAs != 0)){
    for (candidato in noasignados){
      if(asigAp[candidato] > numB ||
      prefA[candidato,asigAp[candidato]]==0){ # Si el candidato
      # ha superado ya sus propuestas se le asigna -1, es decir,
      # se queda soltero/a
        asigAs[candidato] <- -1
      } else {
        propB <- prefA[candidato,asigAp[candidato]] # Almacenamos
        # la propuesta del candidato según la ronda de propuesta
        # por la que se encuentre
        a <- asigBs[propB,]
```

```
        bondadcandidato <- match(candidato, prefB[propB,])
        # Almacenamos la posición de pref del candidato en la
        # lista de los propuestos
        if(is.na(bondadcandidato)){  # Si el candidato no está
        # en la lista de pref de los propuestos, se le asigna
        # una mala posición
          bondadcandidato <- numA+1
        }
      }
      if(length(a[a!=0]) < q[propB] & asigAs[candidato]!=-1){
        asigBs[propB,candidato] <- bondadcandidato
        asigAs[candidato] <- propB
      } else {
        if(bondadcandidato < max(asigBs[propB,]) &
        asigAs[candidato]!=-1){ # Si la asignación anterior es
        # peor que la nueva
          peor <- which.max(asigBs[propB,])
          asigAs[peor] <- 0  # Dejamos sin asignar al candidato
          # en peor posición
          asigAp[peor] <- asigAp[peor] + 1  # Aumentamos en uno
          # la ronda de propuesta del candidato rechazado
          asigBs[propB,candidato] <- bondadcandidato  # Asignamos
          # el candidato a la propuesta del mismo
          asigAs[candidato] <- propB  # Asignamos la propuesta
          # al candidato
        } else {
          asigAp[candidato] <- asigAp[candidato] + 1  # Aumentamos
          # en uno la ronda de propuesta del candidato
        }
      }
    }
    noasignados <- which(asigAs == 0)  # Almacenamos aquellos
    # que proponen que no han sido asignados
  }
names(asigAs) <- 1:numA
print('La asignación final viene dada por:'); print(asigAs)
}
```

Ejemplo 3.4. Para nuestro ejemplo vamos a tomar un conjunto de 5 alumnos y uno de 2 universidades con un cupo de dos alumnos cada una. Vemos primero las preferencias de ambos grupos:

```
# Listas de preferencias de los alumnos
h_1 <- c(2,1)
h_2 <- c(1,2)
h_3 <- c(2,1)
h_4 <- c(2,1)
h_5 <- c(1,2)
# Listas de preferencias de las universidades
m_1 <- c(2,4,5,1,3)
m_2 <- c(1,2,3,4,5)
# Cupo de las universidades
q <- c(2,2)
# Matrices de preferencias de los alumnos y las universidades
prefA <- rbind(h_1,h_2,h_3,h_4,h_5); prefA
prefB <- rbind(m_1,m_2); prefB

A matrix: 5 × 2 of type dbl
a_1        2 1
a_2        1 2
a_3        2 1
a_4        2 1
a_5        1 2
A matrix: 2 × 5 of type dbl
u_1        2 4 5 1 3
u_2        1 2 3 4 5

# Utilizamos ahora la función
universidades(prefA,prefB,q)
[1] "La asignación final viene dada por:"
[1]  1  2  3  4  5
[1]  2  1  2  1 -1
```

Por tanto, el emparejamiento resultante sería $\{(a_1, u_2), (a_2, u_1), (a_3, u_2), (a_4, u_1)\}$ quedando finalmente el alumno a_5 sin universidad. En cambio, si hacemos una variación en las preferencias de nuestro alumno a_4, podemos ver cómo se altera nuestro emparejamiento.

```
# Listas de preferencias de los alumnos
h_1 <- c(2,1)
h_2 <- c(1,2)
h_3 <- c(2,1)
h_4 <- c(2,0)
h_5 <- c(1,2)
# Listas de preferencias de las universidades
```

```
m_1 <- c(2,4,5,1,3)
m_2 <- c(1,2,3,4,5)
# Cupo de las universidades
q <- c(2,2)
# Matrices de preferencias de los alumnos y las universidades
prefA <- rbind(h_1,h_2,h_3,h_4,h_5); prefA
prefB <- rbind(m_1,m_2); prefB

A matrix: 5 × 2 of type dbl
a_1        2 1
a_2        1 2
a_3        2 1
a_4        2 0
a_5        1 2
A matrix: 2 × 5 of type dbl
u_1        2 4 5 1 3
u_2        1 2 3 4 5

# Utilizamos ahora la función
universidades(prefA,prefB,q)
[1] "La asignación final viene dada por:"
[1]  1  2  3  4  5
[1]  2  1  2 -1  1
```

Podemos observar que ahora nuestro emparejamiento es $\{(a_1, u_2), (a_2, u_1), (a_3, u_2), (a_5, u_1)\}$, dejando a a_4 sin universidad ya que prefería quedarse fuera a estar en u_1.

Como hemos comentado anteriormente, el algoritmo también puede ayudar en otros ámbitos de la sociedad, como puede ser el ejemplo a continuación.

Ejemplo 3.5. Tenemos 4 asignaturas que quieren realizar un proyecto con dos alumnos cada una, para ello, realizan sus listas de preferencia en relación a la nota de los 8 alumnos que realizarán los proyectos.

```
# Listas de preferencias de los alumnos
a_1 <- c(2,4,1,3)
a_2 <- c(4,3,2,1)
a_3 <- c(1,2,3,4)
a_4 <- c(4,2,3,1)
a_5 <- c(3,1,2,4)
a_6 <- c(2,4,3,1)
a_7 <- c(2,3,1,4)
a_8 <- c(3,4,1,2)
# Listas de preferencias de las materias
```

```
m_1 <- c(3,5,7,8,1,2,4,6)
m_2 <- c(7,2,3,1,5,8,6,4)
m_3 <- c(3,7,8,1,4,2,6,5)
m_4 <- c(1,7,3,5,4,2,8,6)
# Cupo de las materias
q <- c(2,2,2,2)
# Matrices de preferencia de los alumnos y las materias
prefA <- rbind(a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8); prefA
prefB <- rbind(m_1,m_2,m_3,m_4); prefB

A matrix: 8 × 4 of type dbl
a_1        2 4 1 3
a_2        4 3 2 1
a_3        1 2 3 4
a_4        4 2 3 1
a_5        3 1 2 4
a_6        2 4 3 1
a_7        2 3 1 4
a_8        3 4 1 2
A matrix: 4 × 8 of type dbl
m_1        3 5 7 8 1 2 4 6
m_2        7 2 3 1 5 8 6 4
m_3        3 7 8 1 4 2 6 5
m_4        1 7 3 5 4 2 8 6

# Utilizamos ahora la función
universidades(prefA,prefB,q)
[1] "La asignación final viene dada por:"
[1] 1 2 3 4 5 6 7 8
[1] 2 4 1 4 1 3 2 3
```

Es decir, nuestro emparejamiento quedaría de la siguiente manera

Asignatura 1	a_3	a_5
Asignatura 2	a_1	a_7
Asignatura 3	a_6	a_8
Asignatura 4	a_2	a_4

Conclusiones

En la presente memoria se han abordado algunos de los desafíos relacionados con la asignación que aún se están investigando con el fin de lograr resultados más óptimos. El estudio de estos problemas desempeña un papel crucial en nuestra sociedad, ya que permite aumentar significativamente la satisfacción de los individuos involucrados.

Los avances tecnológicos han demostrado que al implementar estos algoritmos en lenguajes de programación, podemos obtener importantes beneficios. En el ámbito de los trasplantes de órganos, por ejemplo, se ha comprobado que la correcta aplicación del algoritmo de trasplantes cruzados puede salvar un número considerable de vidas. Esto demuestra que estos problemas no solo brindan beneficios emocionales o económicos a los participantes, sino que también pueden mejorar su calidad de vida.

El objetivo principal de este trabajo es proporcionar una visión general de los problemas de emparejamiento, así como reconocer los logros sobresalientes de los investigadores en este campo, los cuales han contribuido significativamente a nuestra comunidad. Se ha abarcado una parte de esta área de estudio, la cual continúa trabajando día a día para aumentar su potencial. Además, se ha logrado generalizar el algoritmo en el contexto del problema de emparejamiento entre dos conjuntos, eliminando en la práctica algunas restricciones como la necesidad de listas de preferencias o elementos específicos en los conjuntos, lo que posibilita un mayor aprovechamiento de dicho algoritmo. Hemos revisado un algoritmo que nos proporciona emparejamientos estables para el problema de emparejamiento entre los elementos de un único conjunto, siempre y cuando exista una solución factible. Además, presentamos el algoritmo desarrollado por nuestros matemáticos y su implementación en R, que también ofrece importantes contribuciones.

A lo largo de este estudio, hemos podido explorar las características y el funcionamiento de dichos algoritmos, evidenciando su eficacia en la resolución de problemas de asignación. Su implementación en el lenguaje de programación

R ha permitido aprovechar las capacidades y las funcionalidades de esta herramienta, brindando una solución eficiente y precisa a los distintos problemas.

Bibliografía

[1] GALE, D. y SHAPLEY, L.S. College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 1962, vol. 69 (1), pp. 9–15.

[2] IRVING, R.W. An Efficient Algorithm for the "Stable Roommates" Problem. *Journal of Algorithms*, 1985, vol. 6, pp. 577–595.

[3] HERRERO, C. Alvin E. Roth y Lloyd Shapley, Premios Nobel de Economía 2012. *La Gaceta de la RSME*, 2013, vol. 16 (3), pp. 465–478.

[4] ROTH, A.E. The economics of matching: Stability and incentives. *Mathematics of operations research*, 1982, vol. 7 (4), pp. 617–628.

[5] DUBINS, L.E. y FREEDMAN, D.A. Machiavelli and the Gale-Shapley Algorithm. *The American Mathematical Monthly*, 1981, vol. 88 (7), pp. 485–494.

[6] MASARANI, F. y GOKTURK, S.S. On the existence of fair matching algorithms. *Theory and Decision*, 1989, vol. 26 (3), pp. 305–322.

[7] R Core Team. *R: A language and environment for statistical computing*. 2021. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.

[8] Enlace al Google Colaboratory donde se encuentran todos los algoritmos mostrados: https://colab.research.google.com/drive/1_ysnlK9Vl2raMa-yr3mNaxXGBJWnhvax?usp=sharing

Assignment under preferences

Ayoze González Martín
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101124377@ull.edu.es

Abstract

MATCHING THEORY *aims to obtain assignments in different environments that can provide good outcomes for all involved parties. In this work, three assignment problems will be studied: the marriage problem, the roommates problem, and the college admissions problem. The first problem will be further explored by presenting a generalization of it and its implementation in the R programming language. Real-life cases will be examined to showcase the use of these algorithms, which have led to significant advancements for our society, such as organ transplants and student residencies in hospitals.*

1. Marriage assignment problem

David Gale and Lloyd Shapley developed an algorithm in 1962 capable of matching two sets, where each member will have their preference list over the other set, these preference lists must be complete and strict, in addition, both sets must have the same size. Our resulting matching will be "stable," meaning there will be no two members from different couples who prefer each other over their respective partners.
Next we define the algorithm:

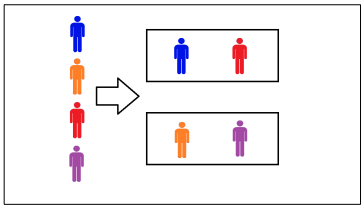
```
Initially all proponents are unpaired
WHILE (there is an unmatched proposer)
    - proposes those who have not proposed
    according to their order of preference
    IF (the proposed is free)
        - accept the proposal
    ELSE
        IF (the proposed person prefers his
        current partner)
        - reject the proposal
        ELSE
        - accept the proposal
        END IF
    END IF
END WHILE
```

In our work, we have relaxed the constraints of the initial algorithm

- It will not be necessary for the preference lists to be complete, meaning we accept the state of "singleness".
- it is also not required for both sets to be of the same size.

2. Roommates assignment problem

The roommate assignment problem was also mentioned by Gale and Shapley in their article, but it wasn't until 1985 that Robert W. Irving proposed a two-phase algorithm capable of solving this problem, providing a stable matching if one exists.



Phase I: This phase consists of provisionally assigning a roommate and discarding those who do not allow for a stable final matching.

Phase II: In this phase, we aim to reduce the preference lists following a loop called the "all-or-nothing cycle". If, at the end of this phase, there are still preference lists with more than one person, we return to Phase I and repeat the process with the reduced preference lists.

We can highlight that if at any point in the algorithm, any preference list becomes empty, we can consider that the problem will not have a stable solution.

3. College admissions assignment problem

Gale and Shapley also created what we now know as the college admissions problem, a variation of the marriage problem where the assignment is one-to-many, as each college can accept a limited number of students, known as a quota.
Next we define the algorithm:

```
Initially all students are unpaired
WHILE (there is an unmatched student)
    - proposes those who have not proposed
    according to their order of preference
    IF (the college has free quota)
        - accept the proposal
    ELSE
        IF (the college prefers its
        current students)
        - reject the proposal
        ELSE
        - accept the proposal
        END IF
    END IF
END WHILE
```

This algorithm has had a significant impact on society and has been utilized in numerous instances, such as assigning residents to hospitals or even aiding in organizing kidney transplants.
In the 1980s, Alvin E. Roth proposed the Boston algorithm, a variant that, unlike the original algorithm, does not remove a previously accepted student.

References

[1] GALE, D. y SHAPLEY, L.S. College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 1962, vol. 69 (1), pp. 9–15.

[2] ROBERT W. IRVING. An Efficient Algorithm for the "Stable Roommates" Problem. *Journal of Algorithms*, 1985, vol. 6, pp. 577–595.

[3] ALVIN E. ROTH. The economics of matching: Stability and incentives. *Mathematics of operations research*, 1982, vol. 7 (4), pp. 617–628.