UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

Facultad de Ciencias Sociales y Jurídicas de Elche
Grado en Estadística Empresarial

Trabajo Fin de Grado
Curso 2018/2019

El problema de ordenamiento lineal: Aplicación a un caso real

Autor:
Valea Massó Ibarra

Tutora:
Mercedes Landete Ruiz

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

La optimización de recursos es uno de los objetivos clave en cualquier empresa, y, en particular, en aquellas involucradas en el proceso de la construcción de obras, donde se trabaja con macropedidos que, además, suelen servirse muchas veces a medio-largo plazo y, quizás, a larga distancia, por lo que una planificación y/o previsión inadecuada puede provocar pérdidas importantes a la empresa. Para evitar esto surgen los problemas de optimización combinatoria, que tratan de optimizar todos aquellos procesos de construcción, las redes de tráfico o telecomunicaciones, o incluso en la secuenciación de ADN, entre otros.

Dentro de los problemas clásicos de optimización combinatoria se encuentra el problema de ordenamiento lineal. Este problema consiste en agregar diferentes órdenes o rankings para establecer un orden conjunto. Estos problemas han sido muy estudiados en la literatura y continúan siéndolo en la actualidad.

Esta memoria se adentra en los problemas de ordenamiento lineal, se explica en qué consisten y se aborda una aplicación concreta real que se resolverá con el programa informático RStudio. Se puede dividir el documento en cuatro capítulos. El primer capítulo lo constituyen algunos de los problemas de optimización combinatoria básicos. El segundo capítulo contempla en profundidad en qué consisten los problemas de ordenamiento lineal, *Linear Ordering Problem*, consideraciones relevantes, métodos alternativos al convencional en el que nos centramos y en la formulación general con algunos ejemplos que clarifican la parte teórica. El tercer capítulo consta de una breve introducción al programa que se utilizará para resolver estos problemas, RStudio, y en la librería principal *lpSolve*; posteriormente, se centra en el código de programación que se debe usar y en la resolución de los ejemplos ya planteados. Por último, en el cuarto bloque se plantea y resuelve un ejemplo real de las fiestas de los caballos del vino de Caravaca de la Cruz, en Murcia.

INTRODUCCIÓN

La sociedad está sumida en un proceso de constante evolución y todas aquellas operaciones comerciales, ventas, subscripciones, y un gran etcétera, se convierten en datos. Estos datos pueden representarse, investigarse o archivarse, sin embargo, de muchos otros nace la necesidad de que se ordenen. Esta ordenación nos resulta de gran utilidad, ya que así podemos ver en qué punto nos encontramos y hacia dónde queremos ir.

Según el diccionario de la RAE, ordenar significa colocar algo o a alguien de acuerdo con un plan o de modo conveniente, en su primera acepción, y encaminar o dirigir algo o a alguien a un fin determinado, en su segunda. Dicho esto, si tenemos que ordenar unos datos concretos, teniendo en cuenta aquello que queremos que se cumpla o las determinaciones que queremos que se den, los problemas de ordenamiento lineal ayudan a decidir el mejor orden, teniendo en cuenta estos requisitos. Conviene señalar que, a la hora de ordenar los datos de mejor a peor, de principio a fin o cualquier rango que se desee, dependiendo del contexto, ser mejor o ser el primero puede tener distinto significado. Se ordenan los individuos o equipos en muchos deportes como equipos nacionales que se clasifican en la Clasificación Mundial de la FIFA, se ordenan los Juegos Olímpicos, cada país miembro (NOC) se clasifica sobre la base de oro, plata y medalla de bronce, se ordena la relación al crédito permanente o cualquier tipo de información (9).

En la literatura, a la ordenación se le denomina ranking completo o parcial, dependiendo de si ordenamos todo el conjunto de elementos o solo algunos de ellos (14). Un ranking es una relación entre un conjunto de elementos tales que, para uno o varios criterios, el primero de ellos presenta un valor superior al segundo, este a su vez mayor que el tercero y así sucesivamente. El orden se refleja asignando a cada elemento un ordinal, generalmente números enteros positivos. De este modo se pueden reducir medidas detalladas a una secuencia de números ordinales, proporcionando una clasificación más simple y fácil de entender y que sustituye a información más compleja que puede incluir múltiples criterios. Al resultado de la puesta en común de los distintos rankings se le llama agregación de rankings (7).

Para poder cumplimentar esta memoria he de mencionar algunas asignaturas del grado de Estadística Empresarial de la Universidad Miguel Hernández que me han dado los conocimientos necesarios. En primer lugar, la asignatura de estadística, donde

interioricé conceptos básicos de estadística y una primera toma de contacto con el programa RStudio; en segundo lugar, las asignaturas de modelos de optimización y optimización de recursos, donde aprendí los distintos modelos de optimización y cada una de sus utilidades; en tercer lugar, la asignatura de Minería de datos, donde profundicé en qué era una base de datos, todos los tipos de datos que las integraban y cómo debía tratarlos, además de aprender a extraer toda la información posible que puedan tener esos datos y saber interpretarla; por último, la asignatura de Simulación de Procesos y Sistemas, donde estudié las Cadenas de Markov, la simulación y la teoría de colas.

Esta memoria consta de cuatro capítulos que se centran en los problemas de ordenamiento lineal[1] perteneciente a los problemas de optimización combinatoria. Además de hacer un recorrido por la literatura, se resuelven breves ejemplos y una aplicación concreta real en RStudio.

En el capítulo 1, se introducen conceptos básicos de optimización combinatoria necesarios para situar los LOP. Los problemas de optimización combinatoria son aquellos cuyas variables de decisión son enteras y donde el espacio de soluciones está formado por ordenaciones o subconjuntos de números naturales. Seguidamente, se adentra en algunos de los problemas clásicos de optimización (8), (22). Se refiere el problema de la mochila (6), problema que tiene gran uso en la actualidad y muy fácil de comprender, que consiste en la decisión de una persona que tiene una mochila con una cierta capacidad y tiene que elegir qué elementos pondrá en ella, es decir, en buscar la mejor solución entre un conjunto finito de posibles soluciones a un problema. Posteriormente, encontramos el LOP, en el que se profundizará a lo largo de la memoria; el problema de asignación (11), que tiene numerosas aplicaciones como la asignación de personal a máquinas, horarios a maestros, etc. Aparecen, también, los problemas de recubrimiento y empaquetado y, por último, el problema del viajante de comercio (1), probablemente, el más conocido de todos, donde se debe encontrar una ruta óptima para satisfacer las demandas de un conjunto de clientes.

En el capítulo 2, nos adentramos en el LOP específico que trataremos, (4), (7), (10), (13), (19), junto con algunas de sus aplicaciones principales, las tablas input-

[1] En adelante, los problemas de ordenamiento lineal serán mencionados por sus siglas en inglés LOP.

output (5). Además, se instruyen algunas consideraciones y algunos métodos alternativos con los que también se puede establecer un orden. Posteriormente, se formula de manera general (7) el problema, explicando cada una de las partes que forman la función objetivo y el conjunto de restricciones. Por último, visualizamos algunos ejemplos.

En el capítulo 3, se introduce el programa y el lenguaje, R y RStudio, programa que se usará para la resolución de los problemas planteados y del ejemplo real que se verá en el último capítulo. Además, se explica la librería empleada *lpSolve* (12), (15), (16), (17), (18), cómo cargar datos en RStudio (20), cada una de las partes que deben formar el código a implementar en el programa para dicha resolución y se resuelven aquellos problemas vistos a lo largo de la memoria con RStudio. Por último, se hace una comparación entre los métodos alternativos planteados y el LOP.

Finalmente, en el capítulo 4, se plantea un ejemplo real: la fiesta de los caballos del vino de Caravaca de la Cruz, en Murcia (21). Se introduce en qué consisten dicha fiesta y se explica de qué parte de la fiesta es de la que se extraen datos para establecer un orden.

CAPÍTULO 1

En este capítulo haremos un recorrido por los conceptos de optimización combinatoria y por aquellos problemas básicos que componen dichos problemas. En la Sección 1.1 se introducen conceptos básicos de optimización combinatoria. La Sección 1.2 se centra en el problema de la mochila o "Knapsack problem". La Sección 1.3 está dedicada al LOP que se resuelve durante toda la memoria. En la Sección 1.4 se aborda la formulación y un ejemplo de los problemas de asignación. La Sección 1.5 focaliza los problemas de recubrimiento o "set covering". La Sección 1.6 refiere los problemas de empaquetado o "set packing" y, en la Sección 1.7 se plantean los problemas del viajante de comercio, problema clásico de optimización combinatoria.

1.1. Introducción a la optimización combinatoria

Los problemas de optimización en los que las variables de decisión son enteras, es decir, donde el espacio de soluciones está formado por ordenaciones o subconjuntos de números naturales, reciben el nombre de problemas de optimización combinatoria. En este caso, se trata de hallar el mejor valor de entre un número finito o numerable de

soluciones viables. Sin embargo, la enumeración de este conjunto resulta prácticamente imposible, aún para problemas de tamaño moderado.

Según Yepes (2002), las raíces históricas de la optimización combinatoria subyacen en ciertos problemas económicos: la planificación y gestión de operaciones y el uso eficiente de los recursos. Pronto comenzaron a modelizarse, de esta manera, aplicaciones más técnicas, y hoy vemos problemas de optimización discreta en diversas áreas: informática, gestión logística (rutas, almacenaje), ciencias, ingenierías y administración de organizaciones.

La trascendencia de estos modelos, además del elevado número de aplicaciones, estriba en el hecho de que "contiene los dos elementos que hacen atractivo un problema a los matemáticos: planteamiento sencillo y dificultad de resolución" (Garfinkel, 1985).

En la optimización combinatoria confluyen la matemática discreta, la teoría de algoritmos y la programación lineal y lineal-entera. La optimización combinatoria y la programación lineal-entera están muy ligadas dado que la mayoría de los problemas de optimización combinatoria se pueden resolver como un problema de programación lineal-entera.

A continuación, se muestran algunos de los problemas clásicos en optimización combinatoria.

1.2. Problema de la mochila o *"Knapsack problem"*

Se trata de maximizar el valor total de la elección de un conjunto de *n* proyectos sin sobrepasar el presupuesto *b* disponible, siendo v_j y c_j el valor y coste de cada proyecto *j* respectivamente.

Las variables del problema son $X_{ij}\begin{cases}1\ si\ elije\ el\ proyecto\ j\\0\ en\ caso\ contrario\end{cases}$

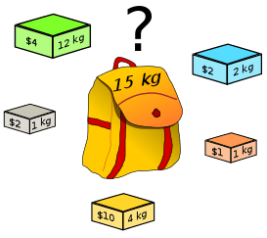
La formulación del problema es

$$\max \sum_{j=1}^{n} v_j x_j$$

s.a:

$$\sum_{j=1}^{n} c_j x_j \leq b$$

$$x_j \in \{0,1\}$$



Fuente: (Francisco, 2015)

Este tipo de problema explica la decisión de un excursionista, que debe preparar su mochila, la cual tiene una capacidad limitada, y, por tanto, no le permite llevar todos los artículos que quisiera llevar a la excursión. Cada artículo que el excursionista puede incluir en la mochila le reporta una determinada utilidad. El problema consiste, por consiguiente, en seleccionar un subconjunto de objetos de tal forma que se maximice la utilidad que el excursionista obtiene, pero sin sobrepasar la capacidad de acarrear objetos.

1.3. Problema de ordenamiento lineal

El *Linear ordering problem* (LOP), es uno de los problemas clásicos de optimización combinatoria. Se comenzó a estudiar por Chenery y Watanabe [1958] con las primeras ideas sobre cómo obtener soluciones para estos problemas, pero no fue hasta poco después cuando Garey y Johnson [1979] demostraron que era un problema de NP-duro.

Estos problemas se tratan en los siguientes capítulos.

1.4. Problema de asignación

Se trata de asignar la realización de *n* tareas a *n* personas (máquinas, etc.). Las variables toman valores enteros sin exigir esta condición en la formulación del problema. Consiste en minimizar el coste total de realizar las tareas sabiendo que cada tarea *i* debe ser hecha por una sola persona y cada persona *j* debe realizar una única tarea, siendo C_{ij} el coste de realizar la tarea *i* por la persona *j*.

Son $X_{ij}\begin{cases}1\ si\ se\ \ asigna\ la\ tarea\ i\ a\ la\ persona\ j\\0\ en\ caso\ contrario\end{cases}$, $\forall i,j$

$$min \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}\,X_{ij}$$

s.a:

$$\sum_{j=1}^{n} X_{ij} = 1 \qquad i = 1,\dots,n$$

$$\sum_{i=1}^{n} X_{ij} = 1 \qquad j = 1,\dots,n$$

$$X_{ih} \geq 0$$

Donde $\{C_{ijhk}\} = a_{ij} f_{ij} d_{hk}$ es igual al coste de asignar los departamentos *i* y *k* a los sitios *h* y *k*, respectivamente,

f_{ij} es el flujo entre departamentos *i* y *j*.

d_{hk} es la distancia entre sitios *h* y *k*.

a_{ij} es el coste de mover una unidad de distancia entre los departamentos *i* y *j*.

$$X_{ij} = \begin{cases} 1\ Si\ el\ departamento\ i\ es\ asignado\ al\ sitio\ k \\ 0\ en\ caso\ contario \end{cases}$$

El problema de la asignación consiste en encontrar un emparejamiento de peso óptimo en un grafo bipartito ponderado. El problema de asignación es un caso particular del problema de transporte, en el que la oferta en cada origen y la demanda en cada destino son ambas de valor 1.

1.5. Problema de recubrimiento o "set covering"

Existen *m* características y *n* combinaciones (subconjuntos) de dichas características. La elección de una combinación implica realizar todas las características de esta. Se trata de minimizar el coste total de las combinaciones elegidas de manera que se cubra o posea cada característica *i* al menos una vez. Los datos son c_j el coste de elegir la combinación *j* y la matriz de pertenencia de cada característica *i* a cada combinación j,

$$a_{ij} = \begin{cases} 1\ Si\ i\ pertenece\ a\ j \\ 0\ \ en\ caso\ contario \end{cases}$$

Denominamos las variables $x_j = \begin{cases} 1\ Si\ se\ elige\ la\ combinación\ j \\ 0\ \ en\ caso\ contario \end{cases}$

$$\min \sum_{j=1}^{n} c_j x_j$$

s.a:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad i = 1, \dots, m$$

$$x_j \in \{0,1\}$$

Un ejemplo de los problemas de recubrimiento es el de una compañía aérea, que necesita asignar sus tripulaciones para cubrir todos sus vuelos. En particular, quiere

resolver el problema de asignar tres tripulaciones con base en San Francisco a unos vuelos en concreto y bajo unas condiciones determinadas.

1.6. Problema de empaquetado o "set packing"

Se tienen que realizar *m* proyectos divididos en *n* paquetes. La elección de un paquete implica realizar todos los proyectos de este. Se trata de maximizar el beneficio total de manera que cada proyecto *i* del conjunto de todos los paquetes que lo incluyen no pueda ser elegido más de una vez. c_j es el beneficio de elegir el paquete j, la matriz de pertenencia de cada proyecto *i* a cada paquete *j* es

$$a_{ij} = \begin{cases} 1\ Si\ i\ pertenece\ a\ j \\ 0\ \ si\ no\ pertenece \end{cases}$$

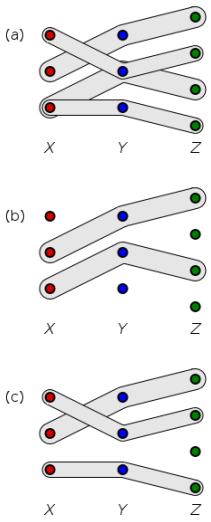
Denominamos las variables $x_j = \begin{cases} 1\ Si\ se\ elige\ el\ paquete\ j \\ 0\ \ en\ caso\ contario \end{cases}$

$$\min \sum_{j=1}^{n} c_j x_j$$

s.a:

$$\sum_{j=1}^{n} a_{ij} x_j \leq 1 \quad i = 1, \dots, m$$

$$x_j \in \{0,1\}$$



Fuente: (Coincidencia tridimensional, 2019) (4)

Un problema de empaquetado consiste en lo siguiente: supongamos que en la cocina tenemos una colección de distintos ingredientes para cocinar y tenemos un recetario con distintas recetas; cada receta necesita un subconjunto de ingredientes. Queremos preparar el conjunto más grande de recetas que aparecen en el recetario y que necesitan de los ingredientes de los que disponemos. Estamos, en resumen, buscando un empaquetamiento de conjuntos en una colección de recetas cuyo conjunto de ingredientes son distintos.

1.7. Problema del viajante de comercio

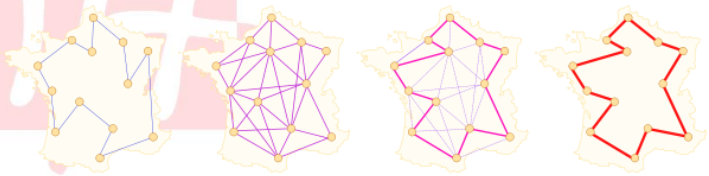
El problema consiste en hacer un recorrido que pase por *n* ciudades sin repetir ninguna y volviendo a la ciudad de partida de manera que la distancia (o tiempo o coste) total sea mínima. Es un problema de asignación, pero con la condición de que la asignación sea un ciclo.

Este, es uno de los problemas más importantes en la historia de la programación matemática, por todas las investigaciones a las que ha dado lugar y por todas las aplicaciones que tiene, tanto directamente o apareciendo como sub-problema dentro de otros más complejos.

Sea c_{ij} la distancia entre las ciudades *i* y *j*.

$$x_{ij} = \begin{cases} 1 \; Si \; se \; va \; de \; la \; ciudad \; i \; a \; la \; ciudad \; j \\ 0 \;\; en \; caso \; contario \end{cases}$$

T_j: Instante de llegada a la ciudad *j*



$$\min \sum_{i,j} c_{ij} x_{ij}$$

$$\sum_{i} x_{ij} = 1 \;\; \forall \, j$$

$$\sum_{j} x_{ij} = 1 \;\; \forall \, i$$

$T_j \geq \; T_i + \; c_{ij} - m(1 - x_{ij}) \;\; \forall \, i,j$ *Fuente:* (Algoritmos de homigas y el problema del viajante., 2018)

$T_j \geq \; c_{1j} - m(1 - x_{1j}) \;\; \forall \, j \neq 1$

$x_{ij} \in \{0,1\}, T_j \geq 0$

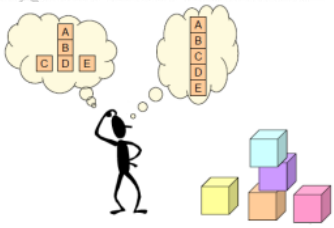
El problema del viajante de comercio consiste en resolver cómo un empresario puede visitar todas sus sedes, situadas en distintos países, con el menor coste. Se debe construir un itinerario que pase por todas las ciudades una sola vez, y que termine en el mismo lugar inicial, pero con la particularidad que sea el más barato.

CAPÍTULO 2

En el capítulo 2 se profundiza en el LOP. En primer lugar, en la Sección 2.1 se observa de manera teórica en qué consisten estos problemas y cuáles son sus principales usos, ilustrándose todos ellos con un sencillo ejemplo. En la Sección 2.2, se establecen consideraciones que deben tener las matrices de datos que se quieren ordenar. En la Sección 2.3, veremos algunos métodos alternativos al que se desarrolla en esta memoria pero que también son válidos a la hora de ordenar cualquier rango de datos; cada uno de estos métodos se explica brevemente y se ilustra con un ejemplo. En la Sección 2.4, se explica de manera matemática con la formulación general del LOP, explicando cada una de las partes que la forman. Por último, en la Sección 2.5, se plantean matemáticamente algunos ejemplos, para una mayor comprensión.

2.1. El problema de ordenamiento lineal

El LOP es un problema de optimización combinatoria clásico que ha recibido la atención de la comunidad de investigación desde que fue estudiado por primera vez por Chenery y Watanabe (1958). Garey y Johnson (1979) demostraron que el LOP es un problema de NP-duro, lo que evidencia la dificultad de resolver las instancias de LOP hasta la optimalidad. Sin embargo, debido a sus numerosas aplicaciones como en la economía (Leontief, 2008) podemos encontrar una amplia variedad de artículos que han tratado el LOP mediante estrategias exactas, heurísticas y metaheurísticas.



Fuente: (Algoritmos de Ordenamiento Lineal, 2017) (2)

Entre los métodos exactos, los más significativos incluyen Branch and Bound (Kaas, 1981; Charon y Hudry, 2006), Branch and Cut (Grötschel et al., 1984) y los algoritmos CuttingPlane (Mitchell and Borchers, 1996, 2000). Estos métodos, como lo destacaron Schiavinotto y Stützle (2004), se comportan competitivamente para instancias desde puntos de referencia específicos con hasta unos pocos cientos de columnas y filas, sin embargo, su tiempo de cálculo aumenta considerablemente con el

tamaño de las instancias y, por lo tanto, no es posible para resolver grandes instancias en un lapso de tiempo razonable. Más allá de las propuestas exactas, los trabajos pioneros propusieron heurísticas constructivas (Chenery y Watanabe, 1958; Aujac, 1960; Becker, 1967). Estos enfoques fueron luego superados por los avances producidos en la optimización metaheurística. Prueba de ello son las soluciones basadas en la búsqueda local (Kernighan y Lin, 1970; Chanas y Kobylanski, 1996), Algoritmos genéticos (Charon y Hudry, 1998) y recientemente, estimación de algoritmos de distribución (Ceberio et al., 2013).

El LOP consiste en ordenar las distintas comparaciones para establecer un orden. De otro modo, dada una matriz $C_{nxn} = (c_{ij})$, se trata de encontrar la mejor permutación *p*, de filas y columnas de la matriz *C*, de modo que se maximice la suma de valores por encima de la diagonal. Imprescindible es destacar que este tipo de problemas se podrán resolver si las matrices iniciales son cuadradas y asimétricas. Si se quiere saber el número de combinaciones que surgen de las filas o columnas de las que disponemos, se debe calcular el factorial del número de filas o columnas.

Charon y Hudry (2007) atribuyeron el LOP a Condorcet (1785), redescubierto por Kemeny (1959), y una formulación diferente a Slater (1961). La primera formulación proviene de los orígenes de la teoría de la elección social. El problema es clasificar un conjunto de candidatos dadas las preferencias de los votantes entre ellos. Aquí, la matriz de matriz B [l, r] contiene el número de votantes que prefieren candidato l al candidato r.

Ejemplo 1

Sea A = {A,B,C} un conjunto de tres concursantes de un concurso de belleza. La matriz cuadrada en la Tabla 1 representa las preferencias, siendo A_{ij} la fracción de las clasificaciones en las que el concursante *i* aparece en la lista antes del concursante *j*. Por ejemplo, A_{21} indica que el concursante B puntúa en el quinceavo puesto al concursante A.

Tabla 1: Matriz {A,B,C} de puntuaciones.

	A	B	C
A	0	15	2
B	7	0	5
C	6	4	0

Fuente: Elaboración propia

La matriz A tiene una dimensión de $3x3$ lo que nos indica que el número de combinaciones que se presentan es de 3! que es equivalente a 6, en este caso, {A,B,C}, {A,C,B}, {B,A,C}, {B,C,A}, {C,A,B}, {C,B,A}.

La combinación {A,B,C} se puede observar en la Tabla 1. Sumando aquellos elementos que están por encima de la diagonal se obtiene un resultado de 22 y de 17 debajo de ésta.

Tabla 2: Matriz {A,C,B} de puntuaciones.

	A	C	B
A	0	2	15
C	6	0	4
B	7	5	0

Fuente: Elaboración propia

La combinación {A,C,B} se muestra en la Tabla 2. Al sumar los elementos por encima de la diagonal se obtiene un valor de 21 y de 18 por debajo de ésta.

Tabla 3: Matriz {B,A,C} de puntuaciones.

	B	A	C
B	0	7	5
A	15	0	2
C	4	6	0

Fuente: Elaboración propia

La combinación {B,A,C} se puede observar en la Tabla 3. Tras sumar los elementos por encima de la diagonal se obtiene un valor de 14 y de 25 por debajo.

Tabla 4: Matriz {B,C,A} de puntuaciones.

	B	C	A
B	0	5	7
C	4	0	6
A	15	2	0

Fuente: Elaboración propia

La combinación {B,C,A} se refleja en la Tabla 4. La suma de los elementos por encima de la diagonal es 18 y por debajo de la misma, 21.

Tabla 5: Matriz {C,A,B} de puntuaciones.

	C	A	B
C	0	6	4
A	2	0	15
B	5	7	0

Fuente: Elaboración propia

La combinación {C,A,B}se puede observar en la Tabla 5. Sumando aquellos elementos que están por encima de la diagonal se obtiene un resultado de 25 y de 14 debajo de ésta.

Tabla 6: Matriz {C,B,A} de puntuaciones.

	C	B	A
C	0	4	6
B	5	0	7
A	2	15	0

Fuente: Elaboración propia

La combinación {C,B,A} se muestra en la Tabla 6. Al sumar los elementos por encima de la diagonal se obtiene un valor de 17 y de 22 por debajo de ésta.

La solución para el LOP, que maximiza el orden en el que los tres concursantes deben estar, es la ordenación {CAB} con una suma de elementos por encima de la diagonal de 25 y una suma por debajo de ésta de 22.

Otra instancia del LOP se originó en el campo de la economía, donde se planteó minimizar la suma de las entradas por debajo de la diagonal en permutaciones de matrices de entrada-salida (Chenery y Watanabe, 1958). Una matriz de entrada y salida representa las relaciones entre los sectores de una economía. Aquí, B[l, r] es una medida monetaria del flujo de bienes del sector l al sector r. El ordenamiento óptimo de los sectores económicos proporciona una visión de la estructura de la economía.

Las matrices *input-output* (5) son un instrumento estadístico que desglosa la Producción Nacional entre los sectores que la han originado y los sectores que la han absorbido; por ello reciben el nombre de *Tablas Intersectoriales*.

La palabra inglesa *output* designa el producto que sale de una empresa o industria mientras que *inputs* son los factores o recursos que se requieren para realizar esa producción.

Tabla 7: Tabla Input-Output

	Sector Agrícola	Sector Industrial	Sector Servicios	Demanda final	Producción total
Producción Agrícola	50	85	25	140	300
Producción Industrial	60	120	50	220	450
Producción de Servicios	30	150	120	250	550

Fuente: ("LAS TABLAS INPUT-OUTPUT", 2019)

En base a la información estadística proporcionada por la Tabla 7, puede elaborarse la Tabla de Coeficientes Técnicos (Tabla 8), que recoge el porcentaje que representa cada uno de los *inputs* sobre la producción final de cada sector.

Tabla 8: Matriz de coeficientes técnicos

	Agricultura	Industria	Servicios
Agricultura	0.17	0.19	0.05
Industria	0.2	0.27	0.09
Servicios	0.1	0.33	0.22

Fuente: ("LAS TABLAS INPUT-OUTPUT", 2019)

2.2. Consideraciones

Para establecer un orden con una matriz dada, se debe prescindir de aquellas matrices simétricas. Una matriz simétrica es aquella que es igual a su transpuesta, es decir, sea A una matriz cuadrada de dimensión mxm, $A = A^T$. Puesto que el objetivo es maximizar la suma de los elementos por encima de la diagonal, si aquellos números son iguales que los que están por debajo de ésta, no podremos establecer un orden. Un ejemplo de matriz simétrica es la que observamos en la Figura 1.

$$\begin{pmatrix} 0 & 2 & 5 \\ 2 & 0 & 8 \\ 5 & 8 & 0 \end{pmatrix}$$

Figura 1: Matriz simétrica Fuente: Elaboración propia

Sumando los números por encima de la diagonal $(2 + 5 + 8 = 15)$, tiene el mismo valor que si sumamos los que están debajo de ésta $(2 + 5 + 8 = 15)$, siendo imposible establecer un orden siguiendo el criterio que estamos analizando hasta ahora.

El orden de las columnas y las filas de una matriz que se quiere ordenar debe ser el mismo. Si las columnas siguen el orden {A,B,C}, las filas deben seguir ese orden, no pueden ir ordenadas de otro modo, como por ejemplo {A,C,B} (Figura 2).

$$\begin{array}{c} \\ A \\ C \\ B \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix} \end{array}$$

Figura 2: Matriz con distinto orden Fuente: Elaboración propia

Otra de las consideraciones es, que las matrices asimétricas pueden disponer de una diagonal con números distintos de cero, sin embargo, estos valores no se deben tener en cuenta a la hora de ordenar. Un ejemplo podría ser la Tabla 8.

2.3. Métodos alternativos

Un método de ordenamiento podría realizarse mediante medias aritméticas. Una media aritmética de un conjunto finito de números es el valor característico de una serie de datos cuantitativos, objeto de estudio que parte del principio de la esperanza matemática o valor esperado, se obtiene a partir de la suma de todos sus valores dividida entre el número de sumandos.

Para comprender mejor este método, podemos observar el siguiente ejemplo.

Sea A = {A,B,C} un conjunto de tres concursantes de un concurso de belleza. La matriz cuadrada en la Tabla 1 representa las preferencias, siendo A_{ij} la fracción de las clasificaciones en las que el concursante *i* aparece en la lista antes del concursante *j*.

El concursante tendría una media de votos de 15,66 {(0 + 2 + 15)/3 = 17/3} que, como se ha comentado anteriormente, es la suma de los elementos que corresponden al concursante A, dividido por el número de votos que tiene dicho concursante.

De igual modo continuamos con el concursante B quien tiene una media de 4 votos {(7 + 5 + 0)/3 = 12/3}. Por último, el concursante C tendría una media de votos de 3,3333 {(6 + 0 + 4)/3 = 10/3}.

La solución para el LOP, que maximiza el orden en el que los tres concursantes deben estar, es la ordenación ABC, ya que el concursante A posee una media de 15,66, el concursante B una media de 4 y el C de 3,333.

Otro método podría, también, abordarse mediante Cadenas de Markov. Una cadena de Markov es un proceso estocástico con espacio de estado discreto, que cumple la propiedad de la pérdida de memoria de Markov. Es decir, la probabilidad del estado actual solo depende del estado anterior y no de toda la evolución.

$$P(x_n = e_n | x_{n-1} = e_{n-1}, \dots, x_0 = e_0) = P(x_n = e_n | x_{n-1} = e_{n-1})$$

Si la probabilidad de que $P(x_{n+1} = j | x_n = i) = P_{ij}$, $\forall n$, es decir, no depende de *n*, entonces la cadena se puede representar por la matriz de transición (Figura 3).

$$\begin{pmatrix} P_{00} & \cdots & P_{0|E1|} \\ \vdots & \ddots & \vdots \\ P_{|E|0} & \cdots & P_{|E||E|} \end{pmatrix}$$

Figura 3: Matriz de transición Fuente: (Simulación de procesos y sistemas)

La suma de cualquier fila de P es 1 y cualquier matriz que cumple que la suma de cada una de sus filas es 1 se llama *estocástica*. Además, esta matriz no puede tener una columna cuyos elementos sean todo ceros.

Para determinar un orden con las Cadenas de Markov, el procedimiento es introducir los estados, la matriz de datos, de forma que la suma de las filas tenga un valor de uno, para después crear la cadena de Markov y calcular los estados de equilibrio. Según el estado de equilibrio que tenga cada una de las variables, éste será el orden óptimo.

Para comprender mejor en qué consiste ordenar con Cadenas de Markov veamos el siguiente ejemplo.

Haciendo uso de la Tabla 1, donde vemos la matriz de preferencias {A,B,C}, vamos a crear la matriz estocástica que hemos mencionado. El procedimiento para hallar esta nueva matriz en sumar todos los números que componen las filas como, por ejemplo, 0 + 15 + 2 = 17 y dividir cada número por ese total. En la posición x_{12}, sería 15/17 = 0.8824.

$$\begin{pmatrix} 0 & 0.8824 & 0.1176 \\ 0.5833 & 0 & 0.4167 \\ 0.6 & 0.4 & 0 \end{pmatrix}$$

Tabla 9: Matriz de transición ejemplo 1 Fuente: Elaboración propia

Habría que calcular, a continuación, la Cadena de Markov y los estados de equilibrio. Profundizaremos con un ejemplo en el capítulo siguiente.

2.4. Formulación general

La programación lineal se ocupa del problema de maximizar una función objetivo lineal sujeta a muchas limitaciones lineales.

Como ya se ha explicado, el LOP se puede resolver como un problema de triangulación. Dada una matriz $C_{nxn} = (c_{ij})$, se trata de encontrar la mejor permutación p, de filas y columnas de la matriz C, de modo que se maximice la suma de valores por encima de la diagonal.

La formulación más utilizada es la siguiente.

Comenzamos haciendo uso de la variable binaria x_{ij} siendo

$$x_{ij} = \begin{cases} 1 \; si \; i \; precede \; a \; j \\ 0 \; en \; caso \; contrario \end{cases}$$

La formulación general del problema es:

(LOP) máx $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij}$

s.a $x_{ij} + x_{ji} = 1 \qquad \forall\, i,j \in V : i < j$ (Restricción 1)

$x_{ij} + x_{jk} + x_{ki} \leq 2 \qquad \forall\, i,j,k \in V : i < j, i < k, j \neq k$ (Restricción 2)

$x_{ij} \in \{0,1\} \qquad \forall\, i,j \in V : i \neq j.$ (Restricción 3)

La Restricción 1, indica que *i* puede ir antes que *j* y *j* delante de *i*, pero ambos casos no pueden ocurrir simultáneamente. Por ello, si finalmente resulta que *i* va antes que *j*, será x_{ij} la variable que tomará valor 1 y $x_{ji}0$ para que se pueda cumplir que ambas sumen 1.

La Restricción 2, describe la relación de transitividad entre las posiciones de tres nodos; así, nos dice que si *j* va antes de *k* y *k* va antes de *i*, es decir, x_{jk}= x_{ki}= 1, entonces j debe ir antes de *i*, es decir, x_{ji}= 1 y x_{ij} = 0.

La Restricción 3, indica que la variable x_{ij} es una variable binaria que puede tomar valor de 1 o 0. Además, indica que *i* y *j* deben tomar distinto valor.

Cualquier solución del sistema (1), (2), (3) es una permutación de los elementos en A.

2.5. Ejemplificación

Como se ha visto en el apartado anterior, ya conocemos la formulación general del LOP. Para tratar de clarificar los conceptos, se presentan los siguientes ejemplos.

Sea A = {A,B,C} un conjunto de tres concursantes de un concurso de belleza. La matriz cuadrada en la Tabla 1 representa las preferencias, siendo A_{ij} la fracción de las clasificaciones en las que el concursante *i* aparece en la lista antes del concursante *j*. Por ejemplo, A_{21} indica que el concursante B puntúa en el quinceavo puesto al concursante A.

La formulación para el ejemplo 1 sería:

(LOP) máx $15\ x_{AB} + 2\ x_{AC} + 7\ x_{BA} + 5\ x_{BC} + 6\ x_{CA} + 4x_{CB}$

s.a $x_{AB} + x_{BA} = 1$

$x_{AC} + x_{CA} = 1$

$x_{BC} + x_{CB} = 1$

$x_{AB} + x_{BC} + x_{CA} \leq 2$

$x_{AC} + x_{CB} + x_{BA} \leq 2$

$x_{AB} \in \{0, 1\}$

$x_{AC} \in \{0, 1\}$

$x_{BA} \in \{0, 1\}$

$x_{BC} \in \{0, 1\}$

$x_{CA} \in \{0, 1\}$

$x_{CB} \in \{0, 1\}$

Figura 4: Formulación específica del LOP, ejemplo 1

Fuente: Elaboración propia

Otro ejemplo que ilustra la formulación general del LOP.

Ejemplo 2

Tabla 10: Ranking de candidatos: lista de cinco órdenes totales.

Votante 1:	a	b	c	d
Votante 2:	a	c	d	b
Votante 3:	c	a	d	b
Votante 4:	d	b	a	c

Votante 5:	b c d a

Fuente: Elaboración propia

Sea V = {a, b, c, d } un conjunto de seis candidatos. Supongamos que cinco votantes los clasifican como muestra la Tabla 11. La matriz cuadrada A en la Tabla 12 representa las preferencias, siendo A_{ij} la fracción de las clasificaciones en las que el candidato *i* aparece en la lista antes del candidato *j*. Por ejemplo, A_{db}= 3/5 porque 3 de cada 5 votantes ordenan *d* antes que a *b*.

Tabla 11: Matriz A de preferencias

	a	b	c	d
a		0.6	0.6	0.6
b	0.4		0.6	0.4
c	0.4	0.4		0.8
d	0.4	0.6	0.2	

Fuente: Elaboración propia

La formulación de este ejemplo sería:

(LOP) máx $0.4\,x_{ab} + 0.8\,x_{ac} + 0.6\,x_{ad} +$

$0.6\,x_{ba} + 0.8\,x_{bc} + 0.8\,x_{bd} +$

$0.4\,x_{ca} + 0.2\,x_{cb} + 0.6\,x_{cd} +$

$0.4\,x_{da} + 0.2\,x_{db} + 0.4\,x_{dc}$

s.a $x_{ab} + x_{ba} = 1$

$x_{ac} + x_{ca} = 1$

$x_{ad} + x_{da} = 1$

$x_{bc} + x_{cb} = 1$

$$x_{bd} + x_{db} = 1$$

$$x_{cd} + x_{dc} = 1$$

$$x_{ab} + x_{bc} + x_{ca} \leq 2$$

$$x_{ab} + x_{bd} + x_{da} \leq 2$$

$$x_{ac} + x_{cd} + x_{da} \leq 2$$

$$x_{bc} + x_{cd} + x_{db} \leq 2$$

$$x_{bd} + x_{dc} + x_{cb} \leq 2$$

$$x_{ab} \in \{0, 1\}, x_{ca} \in \{0, 1\}$$

$$x_{ac} \in \{0, 1\}, x_{cb} \in \{0, 1\}$$

$$x_{ad} \in \{0, 1\}, x_{cd} \in \{0, 1\}$$

$$x_{ba} \in \{0, 1\}, x_{da} \in \{0, 1\}$$

$$x_{bc} \in \{0, 1\}, x_{db} \in \{0, 1\}$$

$$x_{bd} \in \{0, 1\}, x_{dc} \in \{0, 1\}$$

Figura 5: Formulación específica del LOP, ejemplo 2

Fuente: Elaboración propia

CAPÍTULO 3

En este capítulo, aprenderemos a resolver aquellos problemas vistos anteriormente con el lenguaje de programación R, dedicado a la computación estadística. En la Sección 3.1, se introducirá el lenguaje del que haremos uso, R y RStudio. La Sección 3.2, se centra en las librerías del programa necesarias para resolver los LOP. En la Sección 3.3, resolveremos el ejemplo planteado en el capítulo anterior, adjuntando además el código para su resolución en RStudio. Por último, en la Sección 3.4, compararemos los métodos alternativos expuestos en la Sección 2.3 con el problema de ordenamiento lineal.

3.1. Introducción a R y RStudio

3.1.1. Lenguaje de programación R

Para resolver los LOP utilizaremos R. R es un entorno y lenguaje de programación con un enfoque al análisis estadístico.

R nació como una reimplementación de software libre del lenguaje S, adicionado con soporte para alcance estático. Se trata de uno de los lenguajes de programación más utilizados en investigación científica, siendo además muy popular en el campo de la minería de datos, la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con funcionalidades de cálculo y graficación. R es parte del sistema GNU y se distribuye bajo la licencia GNU GPL.

R proporciona un amplio abanico de herramientas estadísticas (modelos lineales y no lineales, test estadísticos, análisis de series temporales, algoritmos de clasificación y agrupamiento, etc.) y gráficas.

Al igual que S, se trata de un lenguaje de programación, lo que permite que los usuarios lo extiendan definiendo sus propias funciones. De hecho, gran parte de las funciones de R están escritas en el mismo R, aunque para algoritmos computacionalmente exigentes es posible desarrollar bibliotecas en C, C++ o Fortran que se cargan dinámicamente.

3.1.2. Entorno de desarrollo integrado RStudio

RStudio es un entorno de desarrollo integrado (IDE) para el lenguaje de programación R, dedicado a la computación estadística y gráficos. Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo.

RStudio está disponible en ediciones de código abierto y comercial y se ejecuta en el escritorio (Windows, Mac y Linux) o en un navegador conectado a RStudio Server o RStudio Server Pro (Debian / Ubuntu, RedHat / CentOS y SUSE Linux).

Sus características principales es que posee un IDE construido en exclusiva para R, es decir, tiene resaltado de sintaxis, auto completado de código, sangría inteligente, se puede ejecutar el código R directamente desde el editor de código fuente y tiene un salto rápido a las funciones definidas.

3.1.3. Cargando archivos en RStudio

Para poder hacer uso de RStudio a la hora de resolver, tanto problemas LOP, como cualquier otro tipo de problemas, es necesario conocer el origen de los datos y el modo en el que se debe cargar en el programa. A continuación, se van a presentar los modos más comunes de cargar datos que son los documentos .csv y los documentos de Excel.

R cuenta con la función genérica *read.table()*, que puede leer cualquier tipo de archivo que contenga una tabla. Acepta un número considerable de argumentos. Los más importantes son los siguientes

file: La ruta del archivo que importaremos, como cadena de texto. Si el archivo se encuentra en nuestro directorio de trabajo, es suficiente dar el nombre del archivo, sin la ruta completa.

header: Si nuestro archivo tiene encabezados, para ser interpretados como nombres de columna, definimos este argumento como TRUE.

sep: El caracter que es usado como separador de columnas. Por defecto es ";".

col.names: Un vector opcional, de tipo caracter, con los nombres de las columnas en la tabla.

stringsAsFactors: Esta función convierte automáticamente los datos de texto a factores. Si este no es el comportamiento que deseamos, definimos este argumento como FALSE.

Un caso particular de las tablas, son los archivos separados por comas, con extensión .csv, por *Comma Separated Values*, sus siglas en inglés. Este es un tipo de archivo comúnmente usado para compartir datos, pues es compatible con una amplia variedad de sistemas diferentes además de que ocupa relativamente poco espacio de almacenamiento. Un ejemplo podría ser el siguiente, donde observamos como sí hay encabezado cuyo nombre es "nombres" y el separador empleado es la coma (",").

```
data <read.table(file = "data.csv", header = TRUE, sep = ",",col.names
 = nombres)
```

Una ventaja de usar documentos con extensión .csv es la posibilidad de usar la función *read.csv()*. Se acepta los mismos argumentos que *read.table()*, pero al usarla con un archivo .csv, en casi todos los casos no hará falta especificar nada salvo la ruta del archivo. Por ejemplo: `data <- read.csv("data.csv").`

Un formato usado con mucha frecuencia para almacenar archivos son las hojas de cálculo, en particular las generadas por el paquete Microsoft Excel.

Para importar datos desde este tipo de archivos, necesitamos instalar el paquete *readxl*, que contiene funciones específicas para realizar esta tarea.

```
install.packages("readxl")
library(readxl)
```

Emplearemos principalmente *read_excel()* con el que se importará archivos .xls y xlsx. Esta función tiene los siguientes argumentos principales:

path: La ruta del archivo a importar. Si no especificamos una ruta completa, será buscado en nuestro directorio de trabajo.

sheet: El nombre de la pestaña a importar. Si no especificamos este argumento, read_excel() intentará leer la primera pestaña de la hoja de cálculo.

range: Cadena de texto con el rango de celdas a importar, escrito con el formato usado en Excel. Por ejemplo, "A1:B:10".

col_names: Con este argumento indicamos si la pestaña que vamos a importar tiene encabezados para usar como nombres de columna. Por defecto su valor es TRUE. Si no tenemos encabezados, podemos dar un vector con nombres para asignar a las columnas.

Un ejemplo podría ser el siguiente.


```
data <- read_excel(path = "data.xlsx", sheet = "Hoja1")
```

Por último, es conveniente ser capaces de importar y exportar datos almacenados en archivos compatibles con paquetes estadísticos comerciales, pues esto nos permitirá usar datos ya existentes compatibles con ellos y colaborar con otras personas. Se hará uso del paquete *haven.*

Todas estas funciones nos piden como argumento *file* la ruta y nombre del archivo a importar, si no especificamos ruta, será buscado en nuestro directorio de trabajo. Todos aquellos datos que se importan lo hacen del modo *data.frame*.

read_spss(): SPSS Statistics, archivos con extensión sav, zsav y por.

read_sav(): SPSS Statistics, sólo archivos sav, zsav.

read_sas(): SAS, archivos sas7bdat.

read_xpt: SAS, archivos xpt.

read_stata(): Stata, archivos dta.

4. Fuente: Elaboración propia

4.1. Librería lpSolve

La instalación básica de R viene equipada con múltiples funciones para la importación de datos, la realización de transformaciones, el ajuste y evaluación de modelos estadísticos, las representaciones gráficas, etc. Sin embargo, el enorme potencial de R deriva de su capacidad de incorporar en cualquier momento nuevas funciones capaces de realizar nuevas tareas.

Un paquete (*package*) es una colección de funciones, datos y código R que se almacenan en una carpeta conforme a una estructura bien definida, fácilmente accesible para R.

Es importante distinguir entre tener un paquete instalado en el ordenador y tenerlo cargado en memoria:

Tenerlo instalado en el ordenador significa simplemente que en algún momento lo hemos bajado de internet y lo hemos copiado en algún directorio, en el que R lo puede localizar.

Tenerlo cargado en memoria significa que durante nuestra sesión de trabajo R ha leído el contenido del paquete y ha incorporado las funciones que contiene a su espacio de trabajo, de tal forma que tales funciones pueden ya ser invocadas y ejecutadas.

El paquete principal que utilizaremos para resolver los problemas de ordenamiento lineal es lpSolve.

El paquete lpSolve fue desarrollado originalmente por Michel Berkelaar en la Universidad de Tecnología de Eindhoven. lpSolve es un solucionador de Programación Lineal Integral Mixta (MILP).

lpSolve es un solucionador de programación lineal (entero) basado en el método *simplex* y el método de derivación y enlace para los números enteros. lpSolve se puede llamar desde R a través de una extensión o módulo. La interfaz completa está escrita en C para que tenga el máximo rendimiento.

Actualmente hay dos paquetes R basados en lpSolve. El paquete lpSolve proporciona funciones de alto nivel para resolver problemas generales lineales / enteros, problemas de asignación y problemas de transporte. El paquete lpSolveAPI proporciona una implementación completa de la API lpSolve. Ambos paquetes están disponibles en CRAN.

Para instalar el paquete mencionado y cargarlo, la sintaxis a emplear en este caso sería la siguiente:

```
#Se instala el paquete lpSolve
    install.packages("lpSolve")

#Se carga el paquete lpSolve
    library(lpSolve)
```

De éste, haremos uso de la librería lp *"Linear/Integer Program"*.

La sintaxis principal de la librería es:

```
lp (direction = "min", objective.in, const.mat, const.dir, const.rhs,transpose.constraints = TRUE, int.vec, presolve=0, compute.sens=0,binary.vec, all.int=FALSE, all.bin=FALSE, scale = 196, dense.const,num.bin.solns=1, use.rw=FALSE)
```

Donde,

direction es una cadena de caracteres que especifica la dirección de optimización: "min" (predeterminado) o "max",

objective.in es un vector numérico que contiene los coeficientes de la función objetivo.

const.mat es una matriz numérica que contiene los coeficientes de restricción, una fila por restricción, una columna por variable de decisión.

const.dir es un vector de caracteres que especifica los tipos de restricciones: cada elemento debe ser uno de "<=", "=" o "> =".

const.rhs es un vector numérico que contiene los lados derechos de las restricciones.

tranpose.constraints es un argumento que está presente sólo para la compatibilidad hacia atrás.

int.vec es un vector numérico de valores enteros positivos que especifican los índices de las variables de decisión que deben ser enteros. La longitud de este vector será, por lo tanto, el número de variables de decisión enteras.

presolve tiene valor por defecto 0 (no); cualquier valor distinto de cero significa "sí".

compute.sens tiene valor por defecto 0 (no); cualquier valor distinto de cero significa "sí".

binary.vec es un vector numérico como *int.vec* que da los índices de variables que se requieren para ser binarios.

all.int indica si todas las variables deben ser enteras. Por defecto es falso.

all.bin indica si todas las variables deben ser binarias. Por defecto es falso.

scale es un valor entero para escalar. Se debe establecer en 0 para no escalar. Por defecto tiene el valor de 196.

dense.const es una matriz de restricción densa de tres columnas. Esto se ignora si se proporciona *const.mat*. De lo contrario, las columnas son número de restricción, número de columna y valor; debe haber una fila para cada entrada distinta de cero en la matriz de restricciones.

num.bin.solns si tiene valor verdadero, el usuario puede solicitar hasta *num.bin.solns* soluciones óptimas para que se devuelvan.

use.rw se usa para vencer un error en algún lugar. Por defecto toma valor falso, pero si quiere encontrar el error, debe tomar valor de verdadero.

A continuación, se van a mostrar las diferentes opciones que *lp* dispone para extraer información del problema y de la solución.

direction devuelve la dirección de la optimización, tal como se ha introducido (min o max).

x.count devuelve el número de variables que hay la función objetivo.

objective devuelve el vector de coeficientes de la función objetivo, según se introducen.

const.count devuelve el número de restricciones que hay en el problema.

constraints devuelve la matriz de restricciones que se ha introducido anteriormente.

int.count devuelve el número de variables enteras.

int.vec devuelve el vector de índices de variables enteras, tal como se introdujo.

objval devuelve el valor de la función objetivo en el óptimo.

solution devuelve el vector de coeficientes óptimo.

num.bin.solns devuelve el indicador numérico del número de soluciones devueltas.

status devuelve 0 en caso de éxito y 2 en caso de que no haya solución factible.

4.2. Ejemplificación

Para comprender la sintaxis que se debe emplear para encontrar un orden óptimo en los LOP, se va a resolver los ejemplos planteados en el apartado 2.5.

En primer lugar, hay que tener en cuenta que no se puede plasmar el problema del mismo modo con formulación general que en el programa que se va a utilizar. Para resolver el problema, hay que seguir el procedimiento establecido en el apartado 3.2 y crear todas aquellas matrices necesarias.

Comenzaremos con el primer ejemplo que hacía referencia a la Tabla 1. La formulación específica del ejercicio es aquella que presenciamos en la Figura 4.

El primer lugar, se deben crear las matrices *problem.coef.obj*, *problem.coef.restr*, *problem.dir.restr* y *problem.term.restr*.

El vector de la función objetivo son aquellos números que componen la matriz inicial, en este caso, la Figura 6.

$$\text{problem.coef.obj} = (15, 2, 7, 5, 6, 4)$$

Figura 6: Vector de la función objetivo Fuente: Elaboración propia

La matriz de los coeficientes de las restricciones se debe formar por aquellos coeficientes que multiplican a cada una de las variables que en ellas aparecen. Cada restricción es una fila de dicha matriz. La matriz correspondiente será la que se muestra en la Figura 7.

$$problem.coef.restr = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Figura 7: Matriz de restricciones Fuente: Elaboración propia

En este caso, hay nueve variables que son $\{ x_{AB}, x_{AC}, x_{BA}, x_{BC}, x_{CA}, x_{CB} \}$ y cinco restricciones, ya que hay tres restricciones del tipo (1) y dos restricciones del tipo (2), por ello la matriz resultante es *5x6*.

El vector de signos con igualdades o desigualdades lo encontramos en la Figura 8. Este vector está formado por las igualdades de las restricciones del tipo (1.3.2) y las desigualdades de inferioridad de las restricciones de tipo (1.3.3).

$$problem.dir.restr = (" = "," = "," = "," <= "," <= ")$$

Figura 8: Vector de signos de igualdades y desigualdades Fuente: Elaboración propia

El vector con los términos independientes de las restricciones es el de la Figura 9. Este vector se compone de aquellos números que están en la parte derecha de la restricción. En general, para cada restricción de tipo (1) debemos encontrar un 1 y en las restricciones de tipo (2) debemos encontrar un 2.

$$problem.term.restr = (1, 1, 1, 2, 2)$$

Figura 9: Vector de términos independientes Fuente: Elaboración propia

Por último, quedaría llamar a la función *lp* antes mencionada siendo ésta la que proporciona los resultados del problema.

De todas las opciones que se han expuesto, para obtener el orden óptimo hemos recurrido a las siguientes funciones: *solution* (para obtener el orden óptimo) y *objval* (para obtener el valor óptimo), dando como resultado

```
solution$solution = 1 0 0 0 1 1

solution$objval = 25
```

Comparando el resultado con las variables que hay en el supuesto, $\{x_{AB}, x_{AC}, x_{BA}, x_{BC}, x_{CA}, x_{CB}\}$, el orden que se establece como óptimo es el de A delante de B, C delante de A y C delante de B, por lo que la solución del LOP que maximiza las preferencias de los votantes es la ordenación {C,A,B} con un valor óptimo de 25 que, como hemos visto anteriormente, es la suma de los elementos por encima de la diagonal.

El código que habría que introducir en RStudio para obtener el resultado anteriormente expuesto es el siguiente.

```
#Definimos un vector con los coeficientes de la función objetivo
   problem.coef.obj <-c(15, 2, 7, 5, 6, 4)

#Definimos una matriz con los coeficientes de las restricciones
   problem.coef.restr <-matrix(c(1, 0, 1, 0, 0, 0,
                                 0, 1, 0, 0, 1, 0,
                                 0, 0, 0, 1, 0, 1,
                                 1, 0, 0, 1, 1, 0,
                                 0, 1, 1, 0, 0, 1), nrow = 5, byrow=TRUE)
   problem.coef.restr

#Definimos un vector de signos que corresponda con el número de restricciones
   problem.dir.restr <-c("=", "=", "=", "<=", "<=")
   problem.dir.restr

#Definimos un vector con los términos independientes de las restricciones
   problem.term.restr <-c(1, 1, 1, 2, 2)

#Creamos una función que llame a todas las variables creadas y además le decimos que todas las variables son enteras
   solution<-lp ("max",problem.coef.obj, problem.coef.restr, problem.dir.restr, problem.term.restr, all.int = TRUE)
```

```
#Obtenemos el orden que maximiza las preferencias de los votantes
  solution$solution

#Obtenemos el valor de la función objetivo
  solution$objval
```

4.3. Comparación entre los métodos alternativos y el LOP.

Para comparar los métodos alternativos con el LOP, vamos a utilizar el ejemplo correspondiente a las Tablas 11 y 12.

El ejemplo decía, sea V = {a, b, c, d} un conjunto de seis candidatos. Supongamos que cinco votantes los clasifican como muestra la Tabla 11. La matriz cuadrada A en la Tabla 12 representa las preferencias, siendo A_{ij} la fracción de las clasificaciones en las que el candidato *i* aparece en la lista antes del candidato *j*.

Comencemos con el método de las medias.

Según este método habría que calcular la media aritmética de cada uno de los candidatos y el que obtuviese un número mayor sería el que estaría en primer lugar y así sucesivamente.

El candidato *a* obtiene una media de 0.6 {(0+0.6+0.6+0.6) /4}.

El candidato *b* obtiene una media de 0.4666666 {(0.4 + 0 + 0.6 +0.4) /4}.

El candidato *c* obtiene una media de 0.5333333 {(0.4 + 0.4 + 0 + 0.8) /4}.

El candidato *d* obtiene una media de 0.4 {(0.4 + 0.6 + 0.2 + 0) /4}.

En conclusión, según este método, el orden óptimo sería *acbd*.

Continuemos con el método de las Cadenas de Markov. Este método consiste en transformar la matriz a una matriz estocástica, calcular la Cadena de Markov y sus estados de equilibrio.

Vamos, en primer lugar, a crear la matriz de transición.

$$\begin{pmatrix} 0 & 0.3333 & 0.3333 & 0.3333 \\ 0.2857 & 0 & 0.4286 & 0.2857 \\ 0.25 & 0.25 & 0 & 0.5 \\ 0.3333 & 0.5 & 0.1667 & 0 \end{pmatrix}$$

Figura 10: Matriz de transición, ejemplo 2 Fuente: Elaboración propia

Ahora en RStudio calcularemos la Cadena de Markov y los estados de equilibrio:

```
#Instalamos los paquetes necesarios
install.packages("markovchain")
install.packages("knitr")
install.packages("Matrix")

#Cargamos los paquetes previamente instalados
library(markovchain)
library(knitr)
library(Matrix)

#Introducimos los estados
estados<-c("a","b","c","d")

#Introducimos la matriz estocástica
matriz<-
matrix(data = c(0,0.3333333333,0.3333333333,0.3333333333, 0.2857142857,0,0.4285
714286,0.2857142857, 0.25,0.25,0,0.5,
0.3333333333,0.5,0.16666666667,0),nrow=4,byrow = TRUE, dimnames=list(estados,e
stados))

#Le damos nombre a las columnas (colnames) y a las filas (rownames)
colnames(matriz)<-c("a","b","c","d")
rownames(matriz)<-c("a","b","c","d")

#Visualizamos la matriz creada
kable(round(matriz,4))

#Creamos la Cadena de Markov
CadenaMarkov<-
new("markovchain",states=estados,byrow=TRUE,transitionMatrix=matriz)
CadenaMarkov
```

```
#Calculamos los estados de equilibrio
steadyStates(CadenaMarkov)
```

La solución que obtenemos al calcular los estados de equilibrio es:

```
steadyStates(CadenaMarkov)
        a         b         c         d
0.2256746 0.2690106 0.2354865 0.2698283
```

El orden óptimo para el ejemplo según las Cadenas de Markov es *dbca*.

Por último, resolvamos el ejemplo con la técnica *linear ordering problem*. Para su resolución emplearemos el código adjunto en el Anexo I. El orden óptimo resultante es *abcd*.

En conclusión, si comparamos el orden óptimo obtenido en los distintos métodos de ordenar podemos observar cómo es de diferente. El método de ordenamiento lineal establece que el mejor orden es *abcd*, el método de las medias aritméticas establece que es *acbd* y el método de las Cadenas de Markov establece que el orden óptimo es el *dbca*.

CAPÍTULO 4

En este capítulo, nos centraremos en un ejemplo real: la fiesta de los caballos del vino de Caravaca de la Cruz, en Murcia. En la Sección 4.1, se introducirá en qué consiste esta fiesta. En la Sección 4.3, se resolverá el problema en RStudio.

4.1. "Los caballos del Vino de Caravaca de la Cruz (Murcia)"

La aplicación real que se va a abordar en esta memoria es el famoso festejo de los caballos del Vino de Caravaca de la Cruz, que se celebra en Murcia los días 1 y 2 de mayo. Está organizado por el Bando de los Caballos del Vino, una federación compuesta por sesenta asociaciones denominadas peñas caballistas. Las sesenta peñas, con sus respectivos caballos, participan en tres concursos de distinta índole.

El festejo de los Caballos del Vino es de los más representativos de Caravaca de la Cruz, se lleva a cabo durante las Fiestas en Honor a la Santísima y Vera Cruz, celebradas del 1 al 5 de mayo, y cada año atraen a la ciudad a más de 100.000 visitantes de todo el mundo. Dichas Fiestas fueron declaradas de Interés Turístico Internacional en 2004 y el festejo de los Caballos del Vino aspira a ser declarado Patrimonio Cultural Inmaterial de la Humanidad por la Unesco.

Cuenta la leyenda, basada en un hecho histórico del siglo XIII, que en el Castillo de Caravaca, donde se refugiaba la población cristiana debido a las constantes batallas que sufría la villa por las tropas musulmanas que habían tonado los alrededores, los alimentos empezaron a escasear, debido a los saqueos y daños que se producían en los huertos de la ciudad. Los cristianos empezaron a enfermar y las reservas de los alimentos a desaparecer, decidiendo ir en busca de víveres y agua, porque incluso las aguas de los aljibes se habían infectado. Tras largas caminatas pudieron encontrar una casa con una bodega en su interior, pero al no encontrar agua, ni alimentos, decidieron cargar el vino, que cargaron en un caballo.

A la llegada al santuario, encontraron un gran cerco musulmán que les impedían el paso. Los templarios se cogieron al caballo, dos delante y dos detrás, defendiendo ese único alimento. Sorteando la defensa enemiga, subiendo a la carrera los odres de vino cargados en el caballo, consiguieron llegar hasta el interior de la fortaleza. Posteriormente el vino fue bendecido por la Santísima Cruz de Caravaca y dado a beber a los enfermos, que milagrosamente sanaron al beberlo.

Esta fiesta muestra su origen con la ceremonia de la bendición del vino y las flores por la Santísima y Vera Cruz de Caravaca. Las primeras referencias documentales aparecen en el siglo XVII. Desde entonces, han sido evolucionado, hasta mostrarse en la actualidad como un triple concurso repleto de fuerza, belleza y emoción: el de caballo a pelo, donde se valorará la figura y el porte del animal; el de enjaezamiento, que premia la belleza y calidad de las piezas y su adecuación al caballo que lo porta; y el de carrera, donde destreza y velocidad se enfrentan al implacable veredicto de cronómetro.

El primero de ellos es el Concurso de Caballo a Pelo. Se trata de la primera jornada de las Fiestas, el 1 de mayo, y la primera en la que se muestran a los caballos de las diferentes peñas, en su totalidad. El animal se muestra en su estado más natural y salvaje, con el cabello al viento, pero aseado, con el pelaje brillante y adornado con trenzas, crin alisada o cualquier otro peinado.

Se trata de un concurso morfológico donde lo que se valora es la belleza y porte del animal. No existe ninguna reglamentación, por lo que la valoración depende únicamente de los criterios subjetivos de los votantes. Durante muchos años la elección de los ganadores estuvo a cargo de un jurado compuesto de expertos en la materia, siendo esto la causa de que se produjeran ciertos problemas y desacuerdos con el resultado, por lo que finalmente se optó por la utilización del mismo sistema que rige el concurso de enjaezamiento, siendo los propios componentes de las peñas los encargados de efectuar la correspondiente votación para establecer el palmarés.

Fuente: “Enjaezamiento”, 2019

Seguidamente, los diez premios de Carrera. Del último al primero, los caballos que más rápido han subido con sus cuatro caballistas asidos al animal. Es frecuente que se trate de peñas con una media de edad joven. Caballistas que, a pesar de su corta edad, ya son expertos en la cuesta y saben lo que significa llevar el caballo hasta el Castillo.

La carrera es una contrarreloj con la que se prueba los caballos determinando cuál de ellos es el más rápido en ascender la cuesta, pero no todo se basa en la velocidad del animal. Las reglas de la carrera exigen que el caballo llegue a la meta con sus 4 caballistas asidos a los flancos, so pena de ser descalificado. Entra en juego, por tanto, la pericia y resistencia de los mozos que controlan a la bestia, los cuales deben prepararla, guiarla y contenerla, si llegara el caso, y todo ello en el menor tiempo posible.

Fuente: “Enjaezamiento”, 2019

A continuación, los premios de enjaezamiento. Aquellos que premian a los caballos mejor vestidos, a las mejores ropas, mejores bordados y mayor originalidad. Las peñas caballistas se dividen en bloques de cara al enjaezamiento, divisiones que marcan la calidad de los trabajos, siendo los del bloque 1 los trabajos más cuidados y tradicionalmente de mayor calidad.

Quedar en los primeros puestos del bloque 3 o 2 da derecho a ascender de categoría; al mismo tiempo, quedar en los últimos puestos del bloque 1 o 2 hace perder la categoría de cara al siguiente año.

Las votaciones las realizan los propios caballistas; las peñas se votan entre ellas al igual que el bando, y entre todos escogen la posición en que queda cada caballo al acabar la jornada.

Fuente: “Enjaezamiento”, 2019

En el caso práctico real nos centraremos en las puntuaciones que se les da a los caballos en el enjaezamiento. Como se ha dicho anteriormente, son las propias peñas quienes establecen las posiciones en las que creen que debería estar cada uno de los caballos y son estas puntuaciones las que queremos ordenar.

Los caballos participantes en el concurso son:

Terry, Campeón, Universo, Sangrino, Caprichoso, Mel-azules, Solteron-Triana, Jupiter, Minipua, Cartujano, Ambicioso, Zambra, Espartaco, Jeque y Luminoso.

El problema consiste en que sea S = {Terry, Campeón, Universo, Sangrino, Caprichoso, Mel-azules, Solteron-Triana, Jupiter, Minipua, Cartujano, Ambicioso, Zambra, Espartaco, Jeque, Luminoso} un conjunto de 15 caballos. Supongamos que los votantes los clasifican como muestra la Tabla adjunta A1. La posición $X_{cd} = 8$ nos indica que Universo vota, según su criterio, que el caballo Sangriento debería estar en octava posición. En el Anexo I se puede ver la transformación de variables que hemos realizado para una mayor comodidad a la hora del análisis del ejemplo.

4.2. Resolución práctica

Vamos a resolver el ejemplo planteado con el código adjunto en el Anexo.

En primer lugar, encontramos la siguiente función objetivo (`problem.coef.obj`)

```
  [1]  2  3  6  1  4 14 10  8 12  5  7 11  9 13  4  1
 [17]  8  3  2 14  9 10 13  6  5  7 12 11  4  2  8  1
 [33]  3 14  9 10 13  5  6  7 12 11  4  2  3  1  5 14
 [49] 13  8 11  6  7 10  9 12  3  1  2  5  4 13 11  8
 [65] 12  7  6 10  9 14  4  2  1  8  3 14 10  9 13  5
 [81]  6  7 12 11  4  2  3  6  1  5 13 10 14  8  7 11
 [97]  9 12  5  3  1  9  2  4 14 10 13  6  7  8 11 12
[113]  4  3  2  8  1  5 14 10 12  6  7 11  9 13  3  5
[129]  2  6  1  8 14 11  9  4  7 12 10 13  4  3  1  7
[145]  2  5 14 12  9 11  6 10  8 13  4  2  3  6  1  5
[161] 14 10  8 12  7 11  9 13  5  2  1  8  4  3 14  9
[177] 10 13  6  7 12 11  4  3  1  8  2  5 14 10  9 12
[193]  6  7 11 13  5  2  1  9  4  3 14 10 11 13  6  7
[209]  8 12
```

Extraemos 105 restricciones del tipo 1 y 910 restricciones del tipo 2, obteniendo como resultado final el siguiente orden óptimo.

```
solution$solution
```

```
 [1] 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1
[51] 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1
```

```
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

[101] 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0

1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1

[151] 0 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 1 1

0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0

[201] 0 0 1 0 0 0 0 0 0 0
```

Y cuyo valor óptimo es de:

```
solution$objval

[1] 1014
```

CONCLUSIONES

En esta memoria se ha estudiado como se agregan las votaciones con la técnica de *linear ordering problem* (LOP). Esta técnica consiste en encontrar el orden óptimo con el que clasificar un conjunto de datos.

A lo largo del documento se ha comparado los LOP con agregaciones más sencillas, como la media aritmética de cada uno de los elementos a ordenar o las Cadenas de Markov mediante los estados de equilibrio, clarificandolos con ejemplos sencillos. Además, se han introducido conceptos básicos de R, RStudio, la librería *lpSolve* y *lp,* para conocer las posibilidades a la hora de resolver un problema. Se ha propuesto un código general en RStudio, con los que resolver problemas de este tipo para cualquier conjunto de datos. Con este código, se han solucionado varios ejemplos sencillos para una mejor comprensión y, además, se ha introducido un problema real de los caballos del vino, de Caravaca de la Cruz, resolviéndose mediante el código general.

Teniendo en cuenta este trabajo y todo aquello aprendido a lo largo del grado, debemos plantearnos diferentes casos de uso para el problema que nos concierna. Sería interesante considerar la agregación de unos datos cuando la población está dividida en clústeres, en estratos o conglomerados, para saber qué orden óptimo establecer según

esa parte de la población a la que pertenece. Por último, sería interesante pensar que, a la hora de ordenar, además de tener en cuenta las consideraciones vistas en esta memoria, cabe la posibilidad de que algunas de las variables que se desean ordenar tengan diferente peso o importancia.

BIBLIOGRAFÍA

1. *Algoritmos de homigas y el problema del viajante.* (28 de Noviembre de 2018). Obtenido de http://www.cs.us.es/~fsancho/?e=71
2. *Algoritmos de Ordenamiento Lineal*. (2017). Obtenido de https://damardiazprogramacion.wordpress.com/acerca-de/
3. Ceberio, J., & Mendiburu, A. A. (2014). The Linear Ordering Problem Revisited.
4. *Coincidencia tridimensional*. (6 de Marzo de 2019). Obtenido de https://en.wikipedia.org/wiki/3-dimensional_matching
5. Coll, J. C. (2019). *Manual básico de economía, la economía de Mercado, las tablas Input-Output.* Obtenido de http://www.juntadeandalucia.es/averroes/centros-tic/14002996/helvia/aula/archivos/repositorio/250/271/html/economia/10/10-a.htm
6. Francisco. (2015). *Gestión de operaciones, Programación Entera, Problema de la mochila en Programación Entera resuelto con OpenSolver*. Obtenido de https://www.gestiondeoperaciones.net/programacion-entera/problema-de-la-mochila-en-programacion-entera-resuelto-con-opensolver/
7. García Nové, E. M. (2018). Nuevos problemas de agregación de rankings: modelos y algoritmos. . *Tesis doctoral. Programa de Doctorado en Estadística, Optimización y Matemática Apliacada. Universidad Miguel Hernández*, 27-37.
8. Garcia Sabater, J. P. (s.f.). *Modelado mediante Optimización Combinatoria.* Obtenido de Universitat Politècnica de València: http://personales.upv.es/jpgarcia/LinkedDocuments/MCOIOptimizacionCombinatoria.pdf

9. Gavara, A. M. (2017). *Problemas de Optimización Combinatoria*. Obtenido de Universitat de València: https://rsmejovenes.blogs.uv.es/programa/sesiones-paralelas/problemas-de-optimizacion-combinatoria/
10. Iranmanesh, E. (2016). *Algorithms for Problems in Voting and Scheduling.* Simon Fraser University.
11. López, B. S. (s.f.). *Problemas de asignación*. Obtenido de https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3n-de-operaciones/problemas-de-asignaci%C3%B3n/
12. others, M. B. (2019). *Interface to "Lp_solve" v. 5.5 to Solve Linear/Integrer Programs.* lpSolve.pdf.
13. Rafel Martí, G. R. (2011). *The linear ordering problem. Exact and Heuristic Methods in Combinatorial Optimization.* Springer; Edición: 2011 (5 de enero de 2011).
14. *Ranking*. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Ranking
15. *RDocumentation*. (s.f.). Obtenido de "lp" programación lineal y entera: https://www.rdocumentation.org/packages/lpSolveAPI/versions/5.5.0.12-3/topics/lp
16. RStudio. (s.f.). *RStudio*. Obtenido de https://www.rstudio.com/products/rstudio/.
17. RStudio. (s.f.). *RStudio*. Obtenido de https://web.archive.org/web/20161203124931/https:/www.rstudio.com/rviews/faq-items/what-license-is-rstudio-available-under/.
18. Santana, A., & Herández, C. N. (s.f.). *Librerías en R. Departamento de Matemáticas.* Obtenido de http://www.dma.ulpgc.es/profesores/personal/stat/cursoR4ULPGC/5-librerias.html
19. Tromble, R. W. (2009). Search and learning for the linear ordering problem with an application to machine translation. *The Johns Hopkins University*.
20. Vega, J. B. (s.f.). *R para principiantes.* 2014. Obtenido de https://bookdown.org/jboscomendoza/r-principiantes4/importar-y-exportar-datos.html
21. Vino., F. C. (s.f.). Obtenido de https://caballosdelvino.org/
22. Vitoriano, B. (2009). Modelos operativos de gestión. *Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid*, 1-39. Obtenido de Facultad

de Ciencias Matemáticas, Universidad Complutense de Madrid: http://www.mat.ucm.es/~bvitoria/Archivos/Apuntes%20MOG_UCM.pdf

ANEXO I

El código de formulación que he desarrollado para resolver los LOP de cualquier base de datos de entrada que cumpla las características que a lo largo de la memoria se han especificado es el siguiente:

```
#Instalamos las librerías necesarias
install.packages("lpSolve")
install.packages("readr")
install.packages("Matrix")

#Cargamos las librerías previamente instaladas
library(lpSolve)
library(readr)
library(Matrix)

#Cargamos la fuente de datos
datos<-read.csv("datos.csv", header = FALSE, sep = ",")
#Extraemos la dimensión de la matriz, cuantas columnas y cuantas filas hay
n<-ncol(datos)

#Calculamos la matriz de coeficientes de la función objetivo
#En primer lugar, para que la matriz de datos inicial se lea en filas, realizamos la transpuesta
Datos1<-t(datos)

#Creamos un vector inicializado a 0 donde se incluirán aquellos números de la matriz inicial que sean distintos de la diagonal
problem.coef.obj <- c(rep(0, (n*n)-n))

#Creamos un bucle que vaya rellenando problem.coef.obj con los números de la matriz inicial, siempre que se cumpla que i != j
t = 1
s = 0
for (i in 1:n){
```

```
  for (j in 1:n){
    s = s+1
    if (i != j){
      problem.coef.obj[t] = datos1[s]
      t = t + 1
    }
  }
}
problem.coef.obj

#Calculamos el número de restricciones de tipo 1
contador1 = 0
for (i in 1:n){
  for (j in 1:n){
    if (i < j){
      contador1 = contador1 + 1
    }
  }
}Contador1

#Calculamos el número de restricciones de tipo 2
contador2 = 0
for (i in 1:n){
  for (j in 1:n){
    for (k in 1:n){
      if (i<j && i < k && j != k){
        contador2 = contador2 + 1
      }
    }
  }
}
contador2

#Mostramos en número de resitricciones que hay del tipo 1 y del tipo 2
dim = contador1 + contador2; dim
dim2<-(n*n)-n; dim2

#Creamos una matriz llamada "vector" inicializada a cero para saber en
la posición en la que se encuentran los números
vector <- matrix(0, nrow = n, ncol = (n))
vector
```

```
#Creamos un bucle con el que obtenemos las posiciones de los números
t = 1
for (i in 1:n){
  for (j in 1:n){
    if (j != i){
      vector[i,j]<- t
      t = t + 1
    }
  }
}
vector

#Inicializamos la matriz "mad" en 0 para rellenarla de unos y ceros y así obtener los coeficientes de las variables que forman las restricciones
mad<-matrix(0, nrow=dim, ncol=(dim2));mad

#Creamos un bucle que recorra dicha matriz y vaya cumpliendo el propósito antes mencionado
t = 1
for (i in 1:n){
  for (j in 1:n){
    if (i < j){
      mad[t,vector[i,j]] = 1
      mad[t,vector[j,i]] = 1
      t = t + 1
    }
  }
}
mad
for (i in 1:n){
  for (j in 1:n){
    for (k in 1:n){
      if (i<j && i<k && j!=k){
        mad[t,vector[i,j]] = 1
        mad[t,vector[j,k]] = 1
        mad[t,vector[k,i]] = 1
        t = t + 1
      }
    }
```

```
  }
}
mad
problem.coef.restr<-mad

#Definimos un vector de signos que corresponden a las restricciones.
rest1<-matrix("=",nrow= 1, ncol=contador1)
rest2<-matrix("<=",nrow= 1, ncol=contador2)
problem.dir.restr<-c(rest1, rest2)
problem.dir.restr

#Definimos un vector con los términos independientes de las
restricciones.
term_rest1<-matrix(1,nrow= 1, ncol=contador1)
term_rest2<-matrix(2,nrow= 1, ncol=contador2)
problem.term.restr<- c(term_rest1, term_rest2)
problem.term.restr

#Creamos una funciónque llame a todas las variables creadas y además
le decimos que todas las variables son int
solution<-
lp ("max",problem.coef.obj, problem.coef.restr, problem.dir.restr, pro
blem.term.restr, all.int = TRUE)

#Obtenemos el orden
solution$solution

#Obtenemos el valor de la función objetivo
solution$objval
```


“Los caballos del Vino de Caravaca de la Cruz (Murcia)”

El ejercicio se realiza con una matriz de datos cuya cabecera son los nombres personales de cada uno de los caballos que vamos a renombrar para una mejor comprensión del ejercicio.

Renombramos los datos:

A	TERRY	I	MINIPUA
B	CAMPEON	J	CARTUJANO
C	UNIVERSO	K	AMBICIOSO
D	SANGRINO	L	ZAMBRA
E	CAPRICHOSO	M	ESPARTACO
F	MEL-AZULES	N	JEQUE
G	SOLTERON-TRIANA	O	LUMINOSO
H	JUPITER		

La matriz de clasificaciones de la que haremos uso en el ejemplo práctico es la siguiente:

Tabla A1 1. Matriz de clasificaciones de los caballos participantes de "Los caballos del Vino"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A	15	2	3	6	1	4	14	10	8	12	5	7	11	9	13
B	4	15	1	8	3	2	14	9	10	13	6	5	7	12	11
C	4	2	15	8	1	3	14	9	10	13	5	6	7	12	11
D	4	2	3	15	1	5	14	13	8	11	6	7	10	9	12
E	3	1	2	5	15	4	13	11	8	12	7	6	10	9	14
F	4	2	1	8	3	15	14	10	9	13	5	6	7	12	11
G	4	2	3	6	1	5	15	13	10	14	8	7	11	9	12
H	5	3	1	9	2	4	14	15	10	13	6	7	8	11	12
I	4	3	2	8	1	5	14	10	15	12	6	7	11	9	13
J	3	5	2	6	1	8	14	11	9	15	4	7	12	10	13
K	4	3	1	7	2	5	14	12	9	11	15	6	10	8	13
L	4	2	3	6	1	5	14	10	8	12	7	15	11	9	13
M	5	2	1	8	4	3	14	9	10	13	6	7	15	12	11
N	4	3	1	8	2	5	14	10	9	12	6	7	11	15	13
O	5	2	1	9	4	3	14	10	11	13	6	7	8	12	15