Trabajo Fin de Grado
Grado en Estadística Empresarial

Acerca del Problema del Ordenamiento Lineal

Autor:
Alberto Pérez Cánovas

Tutor:
Mercedes Landete Ruiz

Facultad de Ciencias Sociales y Jurídicas de Elche
Universidad Miguel Hernández
Curso 2022/2023

ÍNDICE

1. INTRODUCCIÓN

En el presente trabajo empezaremos explicando el Problema del Ordenamiento Lineal (LOP en inglés) y el modelo que lo resuelve aportando ejemplos para su entendimiento. Del Problema de Ordenamiento Lineal se obtiene el orden de elementos o individuos que mejor se ajusta al conjunto de órdenes de los que iniciamos el problema.

Posteriormente, explicaremos los diferentes algoritmos que hemos creado en R Studio para resolver el LOP. Además, expondremos otro algoritmo que nos sirve para indicar a cada elemento/individuo qué acciones debería tomar para mejorar su posición en el orden que devuelve el Problema del Ordenamiento Lineal.

Finalmente aplicaremos los algoritmos a un caso real, relacionado con el mundo del tenis, de donde saldrán nuestras conclusiones.

Este trabajo de fin de grado está relacionado con varias de las asignaturas que se encuentran en el grado, por lo que los conocimientos adquiridos en ellas nos han servido para poder realizarlo.

Por un lado, está relacionada con Gestión y Planificación de la Producción, Optimización de Recursos y Modelos de Optimización debido a que en cada una de ellas hemos tenido que aprender a formular y resolver modelos lineales de optimización, sobre todo manualmente.

También estaría relacionada con varias asignaturas de la carrera en las que hemos tenido que usar el programa R Studio y comprender su lenguaje para después aplicar los conocimientos teóricos aprendidos, como Estadística, Sistemas de Garantía de Calidad o Simulación de Procesos y Sistemas. Hemos aprendido desde cómo crear un vector, hasta cómo utilizar funciones de distintas librerías para resolver modelos lineales, hacer simulaciones, etc.

La finalidad de este trabajo es conocer el Problema del Ordenamiento Lineal, cómo formular el modelo y saber resolverlo manualmente con matrices pequeñas. Además, formular el algoritmo que permita resolver el modelo del LOP con tantas variables y restricciones que a mano sería casi imposible de realizar.

2. EL PROBLEMA

2.1. EL PROBLEMA DEL ORDENAMIENTO LINEAL

El mundo contemporáneo se encuentra inmerso en una creciente cantidad de datos y, en consecuencia, la necesidad de organizar y procesar eficientemente esta información se ha vuelto esencial. El Problema de Ordenamiento Lineal, *Linear Ordering Problem* (LOP), surge como una pieza fundamental en esta búsqueda por estructurar datos de manera efectiva.

El Problema de Ordenamiento Lineal (LOP) tiene una variedad de aplicaciones en diferentes áreas debido a su naturaleza de optimización combinatoria. Algunas de las aplicaciones más comunes del LOP incluyen:

- Logística y Distribución: En la planificación de rutas de transporte y distribución de recursos, el LOP puede ayudar a determinar el orden óptimo en el que deben visitarse diferentes ubicaciones para minimizar los costos de transporte y tiempo.
- Planificación de Proyectos: En la gestión de proyectos, el LOP puede utilizarse para organizar las tareas de manera secuencial, minimizando el tiempo total de finalización del proyecto o maximizando la eficiencia en la utilización de recursos.
- Genómica Comparativa: En la genómica, el LOP se aplica para comparar secuencias genéticas y determinar el orden lineal de los genes en diferentes especies para comprender mejor las relaciones evolutivas.
- Economía y Finanzas: En la asignación de recursos financieros o activos, el LOP puede ser utilizado para ordenar opciones de inversión de acuerdo con objetivos específicos, como maximización de ganancias o minimización de riesgos.

El LOP parte de un conjunto de órdenes que pueden venir de diferentes criterios o jueces. Este problema decide un único orden que sea lo más coherente posible en base al conjunto de órdenes dado, es decir, que se ajuste al máximo.

La función objetivo del modelo LOP puede variar según el contexto, pero a menudo se relaciona con la minimización o maximización de ciertos criterios, como distancias, costos, tiempos o cualquier otra medida relevante. Esta función

objetivo será mayor cuanto más se ajuste la solución a todo el conjunto de órdenes. Veámoslo con un ejemplo:

Imaginemos los dos siguientes conjuntos de órdenes:

1. {ABC, ABC, ABC}
2. {ABC, ACB, CBA}

En el primer conjunto se ve claro que el orden que mejor se ajusta es ABC y, por lo tanto, el valor de su función objetivo será mayor al del segundo conjunto. El valor de la función objetivo en el segundo conjunto va a ser menor ya que el orden que se saqué será menos coherente que el del primer conjunto.

2.2. MODELO DE OPTIMIZACIÓN DEL LOP

El modelo LOP tiene diversas formulaciones de programación, pero nosotros hemos elegido la siguiente:

$$\text{(LOP) máx } \sum_{(i,j)\in A} c_{ij}\, x_{ij}$$

$$\begin{aligned} \text{s.a} \quad & x_{ij} + x_{ji} = 1 && \forall i,j \in V: i < j \\ & x_{ij} + x_{jk} + x_{ki} \leq 2 && \forall i,j,k \in V: i < j, i < k, j \neq k \\ & x_{ij} \in \{0,1\} && \forall i,j \in V: i \neq j \end{aligned}$$

Utilizamos x_{ij} para cada (i,j) en la matriz A, que es una variable binaria que vale 1 si y solo si la posición de i es anterior que la de j, y 0 en caso contrario.

Lo que queremos obtener en nuestra función objetivo es que la suma de los pesos c_{ij} sea máxima. Es decir, que la suma de los pesos de las variables x_{ij} cuyo valor es 1 sea lo más grande posible. Veámoslo con un ejemplo.

Ejemplo 2.1:

El 70% de los votantes prefieren al candidato 1 antes que al 2, el 40% prefieren al candidato 1 antes que al 3, y el 25% prefieren al 2 antes que al 3. Por lo tanto, se nos queda la siguiente matriz:

$$\begin{pmatrix} 0 & 0.7 & 0.4 \\ 0.3 & 0 & 0.25 \\ 0.6 & 0.75 & 0 \end{pmatrix}$$

Esta matriz nos indica los pesos de cada variable, donde por ejemplo $c_{12} = 0.7$ que indica el porcentaje de votantes que prefieran al 1 por delante del 2 y, por lo tanto, la variable que indica lo contrario sería $c_{21} = 0.3$.

En este caso, la forma de que la suma de los pesos sea máxima, sería sumar los tres números más grandes de la matriz: $0.7 + 0.6 + 0.75 = 2.05$, que dan peso a las variables x_{12}, x_{31}, x_{32}. Esto indica que los votantes prefieren al candidato 3 por encima del 1 y del 2, y que prefieren al candidato 1 por encima del 2, dejando el siguiente orden: 3, 1, 2.

La primera restricción nos indica que solo una de las dos variables puede tener el valor 1. Es decir, o está posicionado el nodo i delante de j o viceversa, pero ambos casos no pueden darse simultáneamente. Veámoslo con el ejemplo anterior:

Ya sabemos que las variables $x_{12} = x_{31} = x_{32} = 1$, ya que con sus pesos maximizan la función objetivo. Por lo tanto, las otras variables valen 0. Si sustituimos los valores en la restricción, vemos que se cumple en los tres casos:

- $x_{12} + x_{21} = 1 + 0 = 1$ ✓
- $x_{13} + x_{31} = 0 + 1 = 1$ ✓
- $x_{23} + x_{32} = 0 + 1 = 1$ ✓

Como podemos observar, esta restricción hace imposible que se dé el caso en el que, por ejemplo, el candidato 1 vaya delante del 2 y viceversa. Sólo se puede dar una de las dos opciones, que en este caso es que el 1 va delante del 2.

La segunda restricción nos muestra la relación de transitividad entre las posiciones de tres nodos. Por ejemplo, si i va antes que j y j antes que k, es decir, $x_{ij} = x_{jk} = 1$, entonces sabemos que i va antes que k por lo que $x_{ik} = 1$ y $x_{ki} = 0$. Veámoslo con el ejemplo anterior:

Al ser una matriz 3x3, solo tendremos una restricción en este caso, que es la siguiente:

- $x_{12} + x_{23} + x_{31} \leq 2 \rightarrow 1 + 0 + 1 \leq 2$ ✓

Esto confirma que al estar el 1 por encima del 2, y el 3 por encima del 1, es imposible que el 2 sea mejor que el 3.

La tercera restricción simplemente nos indica que la variable x_{ij} es binaria, es decir, que su valor puede ser 0 o 1.

A continuación, se muestra otro ejemplo de solución del LOP.

Tabla 2.1: Orden de cinco votantes.

Votante 1:	a	b	d	c
Votante 2:	b	c	d	a
Votante 3:	c	a	b	d
Votante 4:	a	c	b	d
Votante 5:	b	c	d	a

Sea $V = \{a, b, c, d\}$, un conjunto de cuatro candidatos, estos se ordenan según el criterio de cada uno de los cinco votantes. El orden de cada uno de ellos se muestra en la Tabla 2.1. A continuación, en la Tabla 2.2, se encuentra la matriz de preferencias, donde se muestra el valor de los pesos c_{ij} como el número de veces que i está delante de j. Por ejemplo, $c_{ab} = 3$ porque 3 de los 5 votantes (Votante 1, Votante 3 y Votante 4) ponen al candidato a delante del b, y $c_{ba} = 2$ porque los otros 2 votantes restantes (Votante 2 y Votante 5) ponen al candidato b delante del a.

Tabla 2.2: Matriz C de preferencias.

	a	b	c	d
a	0	3	2	3
b	2	0	3	5
c	3	2	0	4
d	2	0	1	0

Teniendo ya está matriz de preferencias, que como podemos ver es cuadrada, es decir, tiene el mismo número de filas que de columnas, en este caso 4x4; procedemos a aplicar el modelo LOP que hemos explicado anteriormente.

-Función objetivo: Todas las combinaciones posibles de $V = \{a, b, c, d\}$. Los pesos c_{ij} están sacados de la Tabla 2.2.

$$3x_{ab} + 2x_{ac} + 3x_{ad} + 2x_{ba} + 3x_{bc} + 5x_{bd} + 3x_{ca} + 2x_{cb} + 4x_{cd} + 2x_{da} + x_{dc}$$

-Primera restricción: Todas las combinaciones posibles de V tal que $i < j$, que en este caso forman 6 restricciones:

- $x_{ab} + x_{ba} = 1$
- $x_{ac} + x_{ca} = 1$
- $x_{ad} + x_{da} = 1$
- $x_{bc} + x_{cb} = 1$
- $x_{bd} + x_{db} = 1$
- $x_{cd} + x_{dc} = 1$

-Segunda restricción: Todas las combinaciones posibles de V que cumplan que $i < j, i < k, j \neq k$; en este caso forman 4 restricciones:

- $x_{ab} + x_{bc} + x_{ca} \leq 2$
- $x_{ab} + x_{bd} + x_{da} \leq 2$
- $x_{ac} + x_{cd} + x_{da} \leq 2$
- $x_{bc} + x_{cd} + x_{db} \leq 2$

-Tercera restricción: Todas las variables x_{ij} son binarias:

$$x_{ab}, x_{ac}, x_{ad}, x_{ba}, x_{bc}, x_{bd}, x_{ca}, x_{cb}, x_{cd}, x_{da}, x_{db}, x_{dc} \in \{0,1\}$$

Tras todo esto, podemos darnos cuenta de que esto es muy complicado de resolver a mano y por eso aplicaremos y explicaremos en el próximo capítulo el algoritmo que hemos creado en R Studio.

Otra forma de resolverlo que no sea por el modelo LOP, es mediante permutaciones en la matriz de preferencias (Tabla 2.2) de forma que dejemos en la matriz superior la máxima suma de sus componentes. Vamos a verlo con la resolución del ejemplo que estamos haciendo.

Partimos de la matriz C de preferencias de la Tabla 2.2, donde el orden tanto en filas como en columnas es $\{a, b, c, d\}$.

$$\begin{pmatrix} \ddots & 3 & 2 & 3 \\ 2 & \ddots & 3 & 5 \\ 3 & 2 & \ddots & 4 \\ 2 & 0 & 1 & \ddots \end{pmatrix}$$

La suma de la diagonal superior es igual a 20, vamos a probar otras combinaciones a ver si superan este número. Por ejemplo, vamos a cambiar las filas y columnas de la *a* por la *b*, quedándose un orden de $\{b, a, c, d\}$.

$$\begin{pmatrix} \ddots & 2 & 3 & 5 \\ 3 & \ddots & 2 & 3 \\ 2 & 3 & \ddots & 4 \\ 0 & 2 & 1 & \ddots \end{pmatrix}$$

En este caso, la suma de la diagonal superior equivale a 19, por lo que es inferior al valor anterior. Vamos a probar con el siguiente orden: $\{b, a, d, c\}$.

$$\begin{pmatrix} \ddots & 2 & 5 & 3 \\ 3 & \ddots & 3 & 2 \\ 0 & 2 & \ddots & 1 \\ 2 & 3 & 4 & \ddots \end{pmatrix}$$

Como podemos ver, la suma de la diagonal superior es de 16, lo que sigue siendo inferior a la primera. Si siguiéramos haciendo todas las permutaciones posibles (en este caso 24) y sacásemos la suma de su diagonal superior, veríamos que el mejor orden es el primero que hemos sacado $\{a, b, c, d\}$ con un valor de 20 en su diagonal superior.

2.3. SENSIBILIDAD DE LAS SOLUCIONES.

Para aplicar posteriormente el LOP a nuestro caso real, queremos ver también como varía la solución si cambiamos una o varias posiciones en los órdenes que nos dan los votantes. Es decir, ¿si un votante intercambia una posición de su orden, cambia el resultado? ¿Y si lo hace en dos posiciones? ¿Y si cambian dos votantes la posición de uno de ellos en su orden?

Estas son las preguntas que nos hemos hecho y con las cuáles, en parte, se basa este trabajo. Para entenderlo mejor vamos a explicarlo continuando con el ejemplo anterior.

Partiendo de la Tabla 2.1, imaginemos que el votante 1 cambia su orden y pone al candidato b por delante del a, quedando la tabla de la siguiente forma:

Tabla 2.3: Orden de cinco votantes cambiando el orden del Votante 1.

Votante 1:	b	a	d	c
Votante 2:	b	c	d	a
Votante 3:	c	a	b	d
Votante 4:	a	c	b	d
Votante 5:	b	c	d	a

Cuando hagamos la nueva matriz C de preferencias, las veces que a está delante de b son 2, y las veces que b está delante de a son 3; quedando la nueva matriz así:

$$\begin{pmatrix} \ddots & 2 & 2 & 3 \\ 3 & \ddots & 3 & 5 \\ 3 & 2 & \ddots & 4 \\ 2 & 0 & 1 & \ddots \end{pmatrix}$$

Si sacamos ahora la suma de la matriz diagonal superior, nos da un valor de 19. Por lo tanto, ha disminuido ahora nuestro valor para el orden $\{a, b, c, d\}$. Al igual que hicimos anteriormente, hacemos todas las permutaciones posibles y vemos que el resultado cambia. Ahora el orden que mejor se ajusta a nuestra matriz C de preferencias es $\{b, c, a, d\}$ con un valor de 21, cuya matriz es la siguiente:

$$\begin{pmatrix} \ddots & 3 & 3 & 5 \\ 2 & \ddots & 3 & 4 \\ 2 & 2 & \ddots & 3 \\ 0 & 1 & 2 & \ddots \end{pmatrix}$$

En definitiva, esto nos hace ver que, sólo cambiando una posición en un orden, puede cambiar de forma significativa el orden que mejor se ajusta al conjunto.

3. ALGORITMOS PARA LA SENSIBILIDAD DEL PROBLEMA DEL ORDENAMIENTO LINEAL.

Viendo los ejemplos del capítulo anterior, podemos intuir que sin una máquina que nos ayude, este modelo sería demasiado extenso para resolverlo a mano ya que, si tuviésemos que aplicarlo al mundo real, las matrices pueden tener miles y miles de columnas y filas.

Por ello, en este capítulo explicaremos el algoritmo que hemos programado en R Studio para que nos resuelva cualquier matriz que nos den y después poder aplicarlo en el último capítulo.

Como hemos explicado anteriormente, hay dos formas de resolver el problema: una de ellas es haciendo permutaciones en la matriz y la otra es aplicando el modelo matemático del LOP. Vamos a ver ambos algoritmos en este capítulo.

3.1. CONSTRUCCIÓN DE LA MATRIZ C DE PREFERENCIAS.

Antes de meternos en los algoritmos, primero tenemos que a partir de órdenes de elementos colocados en distintas filas (Tabla 2.1), sacar la matriz C de preferencias (Tabla 2.2). Lo normal es que a nosotros nos llegue una matriz donde cada fila indica un orden de los elementos a los que nuestro algoritmo tendrá que decir cuál es el orden que mejor se ajusta a todos los que se tienen.

Para ello, creamos en R Studio la función *mat_cuadrada*, que a partir de la matriz que nos den (M), nos saca una matriz cuadrada o matriz C de preferencias (A) que posteriormente es la que usaremos para entrar en las dos formas de resolución del LOP. Esta matriz A lo que hace es comparar los elementos colocándolos en filas y columnas. Esto hace que podamos saber que va a tener la diagonal llena de ceros debido a que el mismo elemento no puedo compararse entre sí.

A continuación, se explica cómo se desarrolla esta función:

1. Lo primero es saber el número de elementos que hay en la matriz, es decir, el número de columnas; y la cantidad de órdenes diferentes, es decir, número de filas de la matriz M. (rojo)

2. Después, lo que hace la función es contar las veces que un elemento está delante de otro y viceversa mirándolo en cada una de las filas u órdenes que hay. (verde)
3. Por último, coloca en una nueva matriz A el resultado de cada cuenta. El número de filas y de columnas de está matriz es la misma y se basa en el número de columnas (elementos) que tiene la matriz M. (azul)

Ejemplo 3.1:

Basándonos en la Tabla 2.1, lo primero que haría nuestro algoritmo es sacar el número de elementos, que son 4 (a, b, c, d); y el número de órdenes, o en este caso votantes, que son 5. Por lo tanto, sabemos que tenemos que ordenar 4 elementos según el orden que nos han dado 5 votantes.

Para seguir, vamos a fijarnos solo en los elementos a y b, ya que se hace igual con todas las combinaciones de elementos.

Nuestra función lo que hará será contar las veces que a está delante de b y viceversa en las 5 filas. En este ejemplo podemos ver que a está delante de b en 3 ocasiones y b está delante de a en 2.

Por último, nuestra función tiene que colocar estos valores en la matriz A. En R Studio los elementos a, b, c, d se transforman a números para poder usarlos. Por lo tanto, la a será el 1 y la b el 2. Entonces, en la matriz el valor 3 que indica las veces que a está delante de b, estará en la posición [1,2] y lo contrario en la posición [2,1].

Como hemos dicho anteriormente, este proceso se aplica igual a todas las combinaciones posibles de elementos, dejándonos una matriz A que sí que podremos usar para resolver el problema.

3.2. ALGORITMO CON PERMUTACIONES.

Primero vamos a explicar este modelo que es más práctico y visual que el modelo matemático del LOP. Como ya hemos visto en el capítulo anterior, este algoritmo lo que hace es permutar cada fila/columna de forma que saca la suma de la matriz superior de todas las combinaciones o permutaciones posibles. Al tener que haber el mismo número de filas que de columnas, debido a que cada

fila/columna indica un elemento concreto, nuestra matriz tiene que ser una matriz cuadrada.

Por ejemplo, en una matriz 3x3, habrá 6 permutaciones posibles:

1, 2, 3	2, 1, 3	3, 1, 2
1, 3, 2	2, 3, 1	3, 2, 1

En una matriz 4x4, habrá 24 posibles permutaciones; en una 5x5 habrá 120; etc.

Para la realización de este algoritmo en R Studio, hemos creado una función llamada *máximo*, que a partir de una matriz cuadrada A nos devuelve el mejor orden y el valor de la matriz superior de ese orden, que será la máxima. Lo que hace este algoritmo dentro de la función es muy sencillo:

1. Primero sacamos a partir del número de filas que hay en la matriz, las permutaciones que va a tener que hacer nuestra función. (rojo)
2. Después, cada vez que hacemos una permutación sacamos la suma de la matriz diagonal y comparamos ese valor con el máximo que tenemos hasta ese momento, y el que sea mayor será el que se guarde como nuevo máximo. (verde)
3. Finalmente, cuando termine de hacer el proceso anterior con todas las permutaciones, la función nos devuelve el orden, que es la permutación, y el valor que se ha guardado como máximo. (azul)

3.3. ALGORITMO BASADO EN EL MODELO MATEMÁTICO DEL LOP.

Para hacer el algoritmo basado en el modelo matemático en R Studio, hemos utilizado dos librerías, *gtools* (amplía herramientas para programar en R) y *lpSolve* (herramienta para resolver problemas de programación lineal y entera).

Por lo tanto, basándonos en el modelo matemático del LOP explicado en el capítulo anterior, hemos creado la función modelo, donde a partir de una matriz A (conjunto de órdenes puestos en filas) nos devuelve el orden y el valor máximo de esa matriz, al igual que sucede con el algoritmo con permutaciones.

En esta ocasión usamos la función *lp* de la librería *lpSolve*, donde tenemos que dar la siguiente información:

- Si nuestra función objetivo es de mínimo o máximo, en nuestro caso es máximo y, en R, "*max*".
- Un vector que forma la función objetivo. (rojo)
- Una matriz que está compuesta por lo que se queda a la izquierda de los signos en la primera y segunda restricción del modelo. (verde)
- Un vector con los signos de la primera y segunda restricción del modelo. (azul)
- Un vector con la parte que se queda a la derecha de la primera y segunda restricción del modelo. (amarillo)
- Un vector para decir que todas las variables del modelo son binarias que es lo que nos dice la tercera restricción de nuestro modelo. (violeta)

Teniendo toda esta información, podemos resolver el modelo (naranja), el cual nos devolverá el valor máximo utilizando el comando $objval y el valor de las variables binarias con el comando $solution. Teniendo el valor de las variables binarias, no sabemos cuál es el orden ya que es un conjunto de 1 y 0. Para ello, creamos en R una forma de que tradujera el conjunto de variables binarias a un orden. Lo mostramos con un ejemplo:

Imaginemos que la solución nos da el siguiente conjunto de variables binarias para una matriz 3x3: $\{0\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\}$.

Lo que hacemos en R es decirle que nos pase eso a una matriz 3x3 donde sabemos que las filas y las columnas indican los elementos que tenemos que ordenar, en este caso $\{1, 2, 3\}$. La matriz se quedaría así:

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Sabemos que los 1 indican que el elemento de la columna es mayor que el de la fila en la que se ubican. Por lo tanto, el que está situado en la posición [2,1] indica que el elemento 2 va por delante del 1, el de la posición [2,3] indica que el elemento 2 va por delante del 3, y el último indica que el elemento 3 va por delante del 1.

Finalmente, podemos decir que el orden que mejor se ajusta según las variables binarias de la solución es $\{2, 3, 1\}$.

3.4. ALGORITMO PARA LA SENSIBILIDAD DE LAS SOLUCIONES.

Por último, queremos ver cómo varía la solución si cambiamos alguna posición de los diferentes órdenes de la matriz C de preferencias. Además, queremos que sea personalizado para el elemento que nosotros queremos que mejore en su posición.

A esta función la hemos llamado *posicion*, y lo que necesita para funcionar son tres cosas: la matriz inicial (M), que es aquella que sus filas indican un orden de elementos; el número del elemento que queremos que mejore (x) y el número de posiciones que queremos que mejore (t).

Un valor de 1 para la t, indica que el elemento mejore una posición en uno de los órdenes o filas que hay en la matriz. Un valor de 2 para la t, indica que hay dos opciones, o se mejora en dos posiciones el elemento en uno de los órdenes, o se mejora en una posición en dos órdenes diferentes. Un valor de 3 implicaría tres opciones posibles: mejorar tres posiciones en una de las filas, mejorar dos posiciones en una fila y una posición en otra, o mejorar una posición en tres filas diferentes. El máximo valor que hemos programado es para mejorar en 5 posiciones.

Esta función dependiendo del número de posiciones que se quieran mejorar, se meterá en un bucle o en otro, pero todos funcionan prácticamente iguales. Vamos a poner de ejemplo el de cambiar una posición por fila:

1. Lo primero que va a hacer nuestro bucle es ver que en la primera fila nuestro elemento no es el primero, ya que, si así fuera, no podría mejorar y, por lo tanto, pasaría a la siguiente fila. (rojo)
2. Cumpliendo el primer paso, lo que hace es cambiar de posición a nuestro elemento con el anterior a él en la misma fila. (verde)
3. Sacamos el nuevo orden global que sale del cambio de posición utilizando el modelo matemático del LOP. Comparamos la posición que tiene el elemento en el nuevo orden con la del orden que tenemos usando la matriz inicial. Si la posición es mejor, seguimos el proceso y guardamos

el número de posiciones que hemos mejorado en el orden global, y en caso contrario pasaríamos a la siguiente fila volviendo al primer paso. (azul)

4. En este paso, comparamos el valor que hemos guardado en el paso anterior con el máximo que haya hasta el momento en el bucle (si es el primero en llegar a este paso el valor del máximo es 0, por lo que sería el nuevo máximo). Si el valor es mayor al máximo, se convierte en el nuevo máximo. Si es igual, comparamos la suma de los pesos de las variables binarias de ambos órdenes, por lo que el que mayor suma tenga será el nuevo máximo. En caso de ser menor se volvería al paso uno con la siguiente fila. (amarillo)
5. Finalmente, cuando el bucle ha recorrido todas las filas, se devuelve en que fila deber mejorar una posición para que tu posición en el orden global mejore lo máximo. También te devuelve la suma de los pesos de las variables binarias. (violeta)

4. APLICACIÓN DEL ALGORITMO A UN CASO REAL.

4.1. APLICACIÓN DEL MODELO LOP.

En este capítulo, vamos a aplicar el algoritmo explicado con anterioridad a un caso real. Hemos elegido el tenis para aplicarlo, cogiendo la base de datos que nos da la ATP, donde sale el ranking con las posiciones de los tenistas, además de las posiciones que tienen en cada una de las estadísticas que recogen en esta base. Las estadísticas que recogen, y en las cuales se basa nuestra matriz inicial, son las siguientes:

- E1: Primer servicio (%).
- E2: Puntos ganados con 1º servicio (%).
- E3: Aces.
- E4: Puntos ganados con 2º servicio (%).
- E5: Puntos ganados sirviendo (%).
- E6: Break points salvados (%).
- E7: Puntos ganados al resto con 1º servicio (%).
- E8: Puntos ganados al resto con 2º servicio (%).
- E9: Break points realizados (%).
- E10: Juegos ganados al resto (%).

Por lo tanto, en la matriz inicial los órdenes de cada una de estas estadísticas serán nuestras filas. Estas estadísticas abarcan a los mejores 20 jugadores profesionales de la ATP a 5 de junio de 2023. Para poder meterlo en una matriz, hemos cambiado sus nombres por la posición que tienen en el Ranking ATP en la fecha anteriormente mencionada:

Tabla 4.1: Top 20 del Ranking ATP a 5/06/2023.

Posición	*Jugador*	*Posición*	*Jugador*
1	Carlos Alcaraz	11	Felix Auger-Aliassime
2	Daniil Medvedev	12	Frances Tiafoe
3	Novak Djokovic	13	Cameron Norrie
4	Stefanos Tsitsipas	14	Hubert Hurkacz

5	Holger Rune	15	Borna Coric
6	Andrey Rublev	16	Tommy Paul
7	Casper Ruud	17	Lorenzo Musetti
8	Taylor Fritz	18	Alex de Miñaur
9	Jannik Sinner	19	Pablo Carreño
10	Karen Khachanov	20	Francisco Cerundolo

Ya tenemos todos los datos necesarios para rellenar nuestra matriz inicial, que en R la hemos llamado *datos*. Como ya sabemos, esta matriz muestra el orden de los jugadores en cada una de las filas, o en este caso estadísticas, que son 10. El número de columnas son el número de jugadores que van a ser ordenados, que como hemos visto son 20. Esto hace que nuestra matriz inicial sea una 10x20.

```
> datos
    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
E1    20    1   10   16    3   13    7    2    4    17    14     5    15     6    12     9     8    11    18    19
E2    14   11    8    4   12    3    6    2    9    15    18    10     5     7     1    13    16    20    17    19
E3    14    8    6    2    4   12   11   10   16     5    13     9     3     7     1    15    20    17    18    19
E4     3    9   10    1    5    7    4    8   12    17    13    16    18     2    15     6    11    14    20    19
E5     4    3   12    8   14   11   10    1    2     5     9     6     7    16    13    15    18    20    17    19
E6     4   12    2    1   10    9   11    5    8    16     3     6    20     7    14    13    18    15    17    19
E7     1    2    3   20   17    9   18   13   10     6     5     7    16    12     4     8    15    11    14    19
E8     9    2    1    3    8   17   18   20   16     6    13     5     7     4    12    10    11    14    15    19
E9     2    1    9   17   10    8   18    4    5     3    13    20     6    15    12    16     7    11    14    19
E10    1    2    9    3   17   18   20   13    5     6    10    16    12     7     4     8    11    15    14    19
```

El primer paso es pasar esta matriz a una matriz cuadrada C de preferencias, por lo que usamos la función que habíamos creado en el capítulo anterior *mat_cuadrada* para sacar una matriz que pueda usarse en el modelo.

```
> mat_cuadrada(datos)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
 [1,]    0    5    6    6    8    8    8    7    6     6     7     6     9     7     9     9    10     9    10     9
 [2,]    5    0    6    6    9    8    8    6    8     7     8     6     8     7    10     8     9     9    10     9
 [3,]    4    4    0    5    7    9   10    6    5     6     7     7     9     8    10     7     9     9    10     9
 [4,]    4    4    5    0    6    6    5    6    5     5     9     8     6     8    10     6     6     6    10     6
 [5,]    2    1    3    4    0    6    9    5    3     2     6     6     6     6     9     7     5     5    10     6
 [6,]    2    2    1    4    4    0    8    3    3     4     7     6     5     6     8     6     4     5    10     5
 [7,]    2    2    0    5    1    2    0    4    1     1     6     4     4     7     8     4     6     5    10     4
 [8,]    3    4    4    4    5    7    6    0    3     4     8     5     7     7     9     7     6     8    10     7
 [9,]    4    2    5    5    7    7    9    7    0     6     7     5     8     6     9     8     8    10    10     8
[10,]    4    3    4    5    8    6    9    6    4     0     7     5     7     7     9     9     6     6    10     6
[11,]    3    2    3    1    4    3    4    2    3     3     0     1     4     6     6     4     4     5    10     5
[12,]    4    4    3    2    4    4    6    5    5     5     9     0     5     7     8     6     5     6    10     5
[13,]    1    2    1    4    4    5    6    3    2     3     6     5     0     6     9     5     5     5    10     5
[14,]    3    3    2    2    4    4    3    3    4     3     4     3     4     0     6     3     4     5    10     4
[15,]    1    0    0    0    1    2    2    1    1     1     4     2     1     4     0     2     4     4    10     4
[16,]    1    2    3    4    3    4    6    3    2     1     6     4     5     7     8     0     5     5    10     5
[17,]    0    1    1    4    5    6    4    4    2     4     6     5     5     6     6     5     0     7    10     4
[18,]    1    1    1    4    5    5    5    2    0     4     5     4     5     5     6     5     3     0    10     6
[19,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0     0     0     0     0
[20,]    1    1    1    4    4    5    6    3    2     4     5     5     5     6     6     5     6     4    10     0
```

Como ya hemos explicado anteriormente, esta matriz lo que te dice es las veces que el jugador que indica la fila está por delante del jugador que indica la columna. Por ejemplo, el jugador 1 (Alcaraz) está 6 veces por delante del jugador 3 (Djokovic).

Con esta matriz podemos aplicar el modelo matemático del LOP y ver cuál es el orden que mejor se ajusta a los órdenes que nos dan estas diez estadísticas.

```
> modelo(mat_cuadrada(datos))
El mejor orden es 2 1 4 9 3 10 8 5 20 17 13 6 12 16 7 11 18 14 15 19 con una puntuación de 1357
```

Como podemos observar, según el modelo del LOP el primero en el ranking debería de ser Medvedev, en vez de Alcaraz que bajaría a la segunda posición.

El orden que nos da el modelo es diferente al del Ranking ATP debido a que los datos en los que se basan son diferentes. El Ranking ATP se basa en los puntos que vayas consiguiendo en cada torneo cada año. Nuestro orden, en cambio, se basa en los órdenes de las diferentes estadísticas que nos da la ATP y mediante un modelo matemático obtenemos el orden que mejor se ajustaría a estos datos.

4.2. APLICACIÓN DEL ALGORITMO PARA LA SENSIBILIDAD DEL MODELO.

Ya teniendo el orden del modelo del LOP, lo que hemos hecho es seleccionar a uno de los jugadores para ver cuánto mejoraría en el orden global si solo mejora en una posición de uno de los órdenes o filas de la matriz inicial.

En este caso, hemos imaginado que el jugador número 15, Borna Coric, quiere saber en qué cosa debe de mejorar una posición para que su puesto en el orden global aumente al máximo. Para ello, aplicamos la función *posicion*. Para ello, metemos en esta función la matriz inicial (*datos*), el jugador que queremos que mejore (15) y el número de posiciones a mejorar (1).

```
> posicion(datos,15,1)
Con un cambio, lo mejor que puedes hacer es mejorar en la fila 8 , donde se te queda un orden de
2 1 4 9 3 10 8 5 20 17 13 6 12 16 7 11 18 15 14 19 aumentando tu posición en 1 con una puntuación
de 1356
```

Por lo visto lo mejor que puede hacer Coric, es mejorar una posición en la fila 8, que es la relativa a los puntos ganados al resto con 2º servicio. Esto hace que su posición en el orden global cambie del puesto 19 al 18, por lo que mejora en una posición.

Imaginemos ahora que, en vez de mejorar solo una posición de cualquiera de los órdenes, quiere mejorar en dos posiciones, lo cual puede ser dos posiciones en el mismo orden o fila, o una posición en dos órdenes diferentes. Para ello, cambiamos en nuestra función *posicion* la última variable, que indica el número de posiciones a mejorar, a 2.

```
> posicion(datos,15,2)

 Con dos cambios, lo mejor que puedes hacer es mejorar dos posiciones en la fila 8 , donde se
te queda un orden de 2 1 4 9 3 10 8 5 20 17 13 6 12 16 7 18 15 11 14 19 aumentando tu posición
en 2 con una puntuación de 1355
```

En este caso, lo mejor que puede hacer es subir dos posiciones en la fila 8, que indica los puntos ganados al resto con 2º servicio. Esto hará que Coric suba dos posiciones en el orden global, pasando del puesto 19 al puesto 17.

Ahora vamos a hacer lo mismo, pero con un jugador diferente. En este caso será Auger-Aliassime (11) el que quiera saber en qué mejorar si puede hacerlo en una o en dos posiciones.

```
> posicion(datos,11,1)
Con un cambio, lo mejor que puedes hacer es mejorar en la fila 9 , donde se te queda un orden de
2 1 4 9 3 10 8 5 20 17 13 6 12 16 11 7 18 14 15 19 aumentando tu posición en 1 con una puntuación
de 1356
> posicion(datos,11,2)

 Con dos cambios, lo mejor que puedes hacer es mejorar dos posiciones en la fila 9 , donde se te
queda un orden de 2 1 4 9 3 10 8 5 20 17 13 6 12 11 16 7 18 14 15 19 aumentando tu posición en 2
con una puntuación de 1355
```

Como se puede observar, lo mejor para Auger-Aliassime si solo puede cambiar una posición, es hacerlo en la fila 9 relativa al porcentaje de break points. Esto hace que suba un puesto en el orden global. En caso de poder cambiar dos posiciones, lo mejor sería hacerlo en la misma fila que hemos mencionado antes y le permitiría subir dos posiciones en el orden global, pasando del puesto 16 al 14.

Vamos a probarlo con otro jugador, en este caso será Hubert Hurkacz (14).

```
> posicion(datos,14,1)
Con un cambio, lo mejor que puedes hacer es mejorar en la fila 0 , donde se te queda un orden de
0 0 0 0 aumentando tu posición en 0 con una puntuación de 0
```

En esta ocasión, con un solo cambio, el programa nos transmite con este mensaje que da igual en que fila u orden cambiemos de posición, que Hurkacz no va a mejorar en el orden global.

```
> posicion(datos,14,2)

 Con dos cambios, lo mejor que puedes hacer es mejorar una posición en las filas 4 y 7 ; donde se
te queda un orden de 2 1 4 9 3 10 8 5 20 17 13 6 12 16 7 18 14 11 15 19 aumentando tu posición en
1 con una puntuación de 1357
```

En cambio, si cambiamos dos posiciones, una en la fila 4 relativa al porcentaje de puntos ganados con 2º servicio y la otra en la fila 7 relacionada con el porcentaje de puntos ganados al resto con 1º servicio; sucede algo que no había sucedido hasta ahora.

Hurkacz (14) sube un puesto, pero también lo hace Alex de Miñaur (18), haciendo que baje dos posiciones Auger-Aliassime (11). Esto nos indica que el hecho de cambiar una posición en el orden, no solo nos afecta a nosotros, sino también a nuestros rivales.

Por último, vamos a ver que le sucedería a Casper Ruud (7).

```
> posicion(datos,7,1)
Con un cambio, lo mejor que puedes hacer es mejorar en la fila 1 , donde se te queda un orden de
2 1 4 9 3 10 8 5 20 17 6 12 16 7 18 13 11 14 15 19 aumentando tu posición en 1 con una puntuación
de 1356
```

Sorprendentemente, con un solo cambio en la fila 1, relacionada con el porcentaje de primeros servicios, hace que cambie bastante el orden global. Si comparamos con el orden inicial, los jugadores 6 (Rublev), 12 (Tiafoe), 16 (Paul) y 7 (Ruud) suben un puesto y el 18 (de Miñaur) sube dos puestos. Esto hace que el 13 (Norrie) caiga 5 posiciones y el 11 (Auger-Aliassime) una posición.

Como ya habíamos dicho antes, un cambio puede producir un efecto en cadena en el resto, haciendo que el orden global que mejor se ajusta cambie por completo.

```
> posicion(datos,7,2)

 Con dos cambios, lo mejor que puedes hacer es mejorar dos posiciones en la fila 9 , donde se te
queda un orden de 2 1 4 9 3 10 8 5 20 17 13 6 7 12 16 11 18 14 15 19 aumentando tu posición en 2
con una puntuación de 1355
```

En este caso, mejorando dos posiciones en la fila 9 relativa al porcentaje de break points, vemos que mejora en el orden global dos posiciones, pero que no afecta al resto de jugadores, a excepción de los dos que pierden una posición.

5. CONCLUSIONES.

El Problema del Ordenamiento Lineal, es un modelo de optimización cuyo objetivo es sacar el orden de un conjunto de elementos que mejor se ajuste a todos los órdenes que forman la matriz.

Lo que hemos hecho nosotros es, utilizando lo que nos da el LOP, sacar la forma de que cada uno de esos elementos puedan mejorar las máximas posiciones en el orden final, cambiando una o más posiciones en el conjunto de órdenes. Esto permite, por ejemplo, que los tenistas sepan que golpe es el que tienen que mejorar para subir su posición en el ranking al máximo.

Por otro lado, hemos visto que cambiar la posición de uno de los jugadores no solo puede afectarle a él, sino que puede afectar, tanto positivamente como negativamente, a varios jugadores. Esto se debe a que al cambiar una posición en una de las estadísticas, el orden global que mejor se ajusta al conjunto de ellas puede cambiar significativamente.

Si se quisiera continuar y ampliar este trabajo, hay que pensar que a un jugador no le va a costar el mismo esfuerzo subir un puesto en una de las estadísticas que en otra, ya que quizás el que está delante tiene mucho mejor porcentaje en esa estadística, o porque ese golpe no es con el que mejor se desenvuelve, etc. Por ello sería razonable pensar que se puede añadir una restricción nueva a nuestro LOP, en la que se ponga el coste/esfuerzo de mejorar posiciones para cada jugador y en cada orden.

Esto haría que, por ejemplo, sin ese coste o esfuerzo lo mejor para el jugador sea mejorar en los aces, mientras que eso es lo que más esfuerzo le supone a él porque es su peor golpe. Entonces, tendría que ir a otro orden en el que mejore lo mismo con menos coste.

6. APÉNDICE.

Algoritmo 3.1: Transformar a matriz C de preferencias.

```
mat_cuadrada = function(M){
n = nrow(M)
v = ncol(M)
A <- matrix(rep(0, v*v), nrow = v)
for (i in 1:v){
  for (j in 1:v){
    if (i<j){
      for (k in 1:n){
        if (which(M[k,]==i) < which(M[k,]==j)){
          A[i,j] = A[i,j] + 1
        }
        if (which(M[k,]==i) > which(M[k,]==j)){
          A[j,i] = A[j,i] + 1
        }
      }
    }
  }

}
return(A)
}
```

Algoritmo 3.2: Permutaciones.

```
máximo = function(A){
  n = nrow(A)
  max = 0
  suma = function(A,n){
    sum = 0
    for (i in 1:n) {
      for (j in 1:n){
        if (j>i){
          sum = sum + A[i,j]
        }
      }
    }
    return(sum)
  }
  per = permutations(n,n,repeats.allowed = FALSE)
  v = nrow(per)
  for (i in 1:v){
    A1 = A[per[i,],per[i,]];
    sum = suma(A1,n)
    if (max<sum){
      max = sum
      orden = per[i,]
    }
  }
  return(cat("El mejor orden es", orden, "con una puntuación de", max))
}
```

Algoritmo 3.3: Modelo matemático del LOP.

```
#Librerías
library(gtools)
library(lpSolve)
modelo = function(A){
  n = ncol(A)
  #Vector Función Objetivo
  obj <- c(rep(0, n*n))
  contador_obj = 1
  for (i in 1:n){
    for (j in 1:n){
      obj[contador_obj] = A[i,j]
      contador_obj = contador_obj + 1
    }
  }
  #Matriz Restr por la izquierda
  columna_ij <- matrix(rep(0,n*n), nrow = n)
  contador = 1
  for (i in 1:n){
    for (j in 1:n){
      columna_ij[i,j] = contador
      contador = contador+1
    }
  }
  #Restriccion x(i,j) + x(j,i) = 1
  contador_H2 = 1
  contador_H1 = 1
  for (i in 1:n){
    for (j in 1:n){
      if (i<j){
        if (i==1 & j==2){
          H <- matrix(0, nrow = 1, ncol = n*n)
          H[contador_H2, columna_ij[i,j]] = 1
```

```
      H[contador_H2, columna_ij[j,i]] = 1
      contador_H1 = contador_H1 + 1
      contador_H2 = contador_H2 + 1
    }
    else{
      H <- rbind(H, rep(0, n*n))
      H[contador_H2, columna_ij[i,j]] = 1
      H[contador_H2, columna_ij[j,i]] = 1
      contador_H1 = contador_H1 + 1
      contador_H2 = contador_H2 + 1
    }
  }
 }
}
#Restriccion x(i,j) + x(j,k) + x(k,i) <= 2
for (i in 1:n){
 for (j in 1:n){
  if (i<j){
   for (k in 1:n){
    if (i<k){
     if (j!=k){
      H <- rbind(H, rep(0, n*n))
      H[contador_H2, columna_ij[i,j]] = 1
      H[contador_H2, columna_ij[j,k]] = 1
      H[contador_H2, columna_ij[k,i]] = 1
      contador_H2 = contador_H2 + 1
     }
    }
   }
  }
 }
}
#Vector Signos de las restricciones
signos <- c(rep("=", contador_H1 - 1), rep("<=", contador_H2 - contador_H1))
```

```
  #Vector Restr por la derecha
  b <- c(rep(1, contador_H1 - 1), rep(2, contador_H2 - contador_H1))

  #Vector Secuencia para decir que x es binaria
  bin <- c(rep(0, (n*n) - n))
  contador_bin = 1
  for (i in 1:n){
    for (j in 1:n){
      if (i!=j){
        bin[contador_bin] = columna_ij[i,j]
        contador_bin = contador_bin + 1
      }
    }
  }
  #Resolvemos el modelo
  solucion <- lp("max", obj, H, signos, b, binary.vec = bin)
  #Sacamos el orden de nuestra solución
  sol = c(rep(0, n))
  for (i in 1:n){
    for (j in 1:n){
      if (solucion$solution[columna_ij[i,j]]==1){
        sol[i] = sol[i] + 1
      }
    }
  }
  #Soluciones de mi modelo
  max = solucion$objval
  orden = order(sol, decreasing = TRUE)
  #Dos formas de devolver la solución. Para el algoritmo final usamos el segundo
  #return(cat("El mejor orden es",orden,"con una puntuación de",max))
  return(c(orden, max))
}
```

Algoritmo 3.4: Sensibilidad soluciones del LOP.

```
#Librerías
library(gtools)
library(lpSolve)
#Función
posicion = function(M, x, t){
M2 = M
max = 0
max_fila = 0
max_fila1 = 0
max_fila2 = 0
max_fila3 = 0
max_fila4 = 0
max_fila5 = 0
max_modelo = 0
max_ord = c(rep(0,4))
max_M2 = M
n = nrow(M)
u = ncol(M)

##### Cambiar 1 posición #####
if (t==1){
#Cambiar 1 posición por fila
for (k in 1:n){
  if (which(M2[k,]==x) > 1){
    z = which(M2[k,]==x)
    M2[k, which(M2[k,]==x)] = M2[k, which(M2[k,]==x) - 1]
    M2[k, which(M2[k,]==M2[k, z])[1]] = x
    ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
    ord2 = modelo(mat_cuadrada(M))[c(1:u)]
    if (which(ord1==x)<which(ord2==x)){
      dif = which(ord2==x) - which(ord1==x)
      if (max<dif){
```

```
        max = dif
        max_fila = k
        max_ord = ord1
        max_M2 = M2
        max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
      }
      else if (max==dif){
        if  (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(max_M2))[-
c(1:u)]){
          max = dif
          max_fila = k
          max_ord = ord1
          max_M2 = M2
          max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
        }
      }
    }
    M2 = M
  }
}
cat("Con un cambio, lo mejor que puedes hacer es mejorar en la fila",max_fila,
    ", donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
    "con una puntuación de",max_modelo)
}

###### Cambiar 2 posiciones ######
max_fila1 = 0
max_fila2 = 0
max_fila = 0
if (t==2){
## Para cambiar 2 veces en la misma fila ##
max = 0
for (k in 1:n){
  if (which(M2[k,]==x) > 2){
```

```
    z = which(M2[k,]==x)
    M2[k, which(M2[k,]==x)] = M2[k, which(M2[k,]==x) - 1]
    M2[k, which(M2[k,]==M2[k, z])[1]] = M2[k, which(M2[k,]==M2[k,z])[1] - 1]
    M2[k, which(M2[k,]==M2[k, z]) - 2] = x
    ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
    ord2 = modelo(mat_cuadrada(M))[c(1:u)]
    if (which(ord1==x)<which(ord2==x)){
      dif = which(ord2==x) - which(ord1==x)
      if (max<dif){
        max = dif
        max_fila = k
        max_ord = ord1
        max_M2 = M2
        max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
      }
      else if (max==dif){
        if  (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(max_M2))[-
c(1:u)]){
          max = dif
          max_fila = k
          max_ord=ord1
          max_M2 = M2
          max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
        }
      }
    }
    M2 = M
  }
}
## Para cambiar 1 vez en 2 filas ##
for (i in 1:n){
  for (j in 1:n){
    if (i<j){
      if (which(M2[i,]==x) > 1){
```

```
      z = which(M2[i,]==x)
      M2[i, which(M2[i,]==x)] = M2[i, which(M2[i,]==x) - 1]
      M2[i, which(M2[i,]==M2[i, z])[1]] = x
      if (which(M2[j,]==x) > 1){
        y = which(M2[j,]==x)
        M2[j, which(M2[j,]==x)] = M2[j, which(M2[j,]==x) - 1]
        M2[j, which(M2[j,]==M2[j, y])[1]] = x
        ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
        ord2 = modelo(mat_cuadrada(M))[c(1:u)]
        if (which(ord1==x)<which(ord2==x)){
         dif = which(ord2==x) - which(ord1==x)
          if (max<dif){
            max = dif
            max_fila1 = i
            max_fila2 = j
            max_ord = ord1
            max_M2 = M2
            max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
          }
          else if (max==dif){
           if(modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(max_M2)
)[-c(1:u)]){
              max = dif
              max_fila1 = i
              max_fila2 = j
              max_ord = ord1
              max_M2 = M2
              max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
           }
          }
        }
      }
    }
  }
```

```
    M2 = M
  }
}
if (max_fila1>0){
  cat("\n Con dos cambios, lo mejor que puedes hacer es mejorar una posición en las filas",max_fila1,"y",
      max_fila2,"; donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
      "con una puntuación de",max_modelo)
}else{
  cat("\n Con dos cambios, lo mejor que puedes hacer es mejorar dos posiciones en la fila",max_fila,
      ", donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
      "con una puntuación de",max_modelo)
}
}
```


Cambiar 3 posiciones

```
max_fila1 = 0
max_fila2 = 0
max_fila3 = 0
max_fila = 0
if (t==3){
```

Para cambiar 3 posiciones en la misma fila

```
for (k in 1:n){
  if (which(M2[k,]==x) > 3){
    z = which(M2[k,]==x)
    M2[k, which(M2[k,]==x)] = M2[k, which(M2[k,]==x) - 1]
    M2[k, which(M2[k,]==M2[k, z])[1]] = M2[k, which(M2[k,]==M2[k, z])[1] - 1]
    M2[k, which(M2[k,]==M2[k, z]) - 2] = M2[k, which(M2[k,]==M2[k, z]) - 3]
    M2[k, which(M2[k,]==M2[k, z]) - 3] = x
    ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
    ord2 = modelo(mat_cuadrada(M))[c(1:u)]
```

```
    if (which(ord1==x)<which(ord2==x)){
      dif = which(ord2==x) - which(ord1==x)
      if (max<dif){
        max = dif
        max_fila = k
        max_ord = ord1
        max_M2 = M2
        max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
      }
      else if (max==dif){
        if  (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(max_M2))[-
c(1:u)]){
          max = dif
          max_fila = k
          max_ord = ord1
          max_M2 = M2
          max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
        }
      }
    }
    M2 = M
  }
}
```

Para cambiar 2 veces en una fila y 1 en otra

```
for (i in 1:n){
  for (j in 1:n){
    if (i!=j){
      if (which(M2[i,]==x) > 2){
        z = which(M2[i,]==x)
        M2[i, which(M2[i,]==x)] = M2[i, which(M2[i,]==x) - 1]
        M2[i, which(M2[i,]==M2[i, z])[1]] = M2[i, which(M2[i,]==M2[i, z])[1] - 1]
        M2[i, which(M2[i,]==M2[i, z]) - 2] = x
        if (which(M2[j,]==x) > 1){
          y = which(M2[j,]==x)
```

```
          M2[j, which(M2[j,]==x)] = M2[j, which(M2[j,]==x) - 1]
          M2[j, which(M2[j,]==M2[j, y])[1]] = x
          ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
          ord2 = modelo(mat_cuadrada(M))[c(1:u)]
          if (which(ord1==x)<which(ord2==x)){
            dif = which(ord2==x) - which(ord1==x)
            if (max<dif){
              max = dif
              max_fila1 = i
              max_fila2 = j
              max_ord = ord1
              max_M2 = M2
              max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
            }
            else if (max==dif){
              if (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(
max_M2))[-c(1:u)]){
                max = dif
                max_fila1 = i
                max_fila2 = j
                max_ord = ord1
                max_M2 = M2
                max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
              }
            }
          }
        }
      }
     M2 = M
     }
   }
}
```

Para cambiar 1 vez en 3 filas

```
for (i in 1:n){
```

```
 for (j in 1:n){
  if (i<j){
   for (l in 1:n){
    if (j<l){
     if (which(M2[i,]==x) > 1){
      z = which(M2[i,]==x)
      M2[i, which(M2[i,]==x)] = M2[i, which(M2[i,]==x) - 1]
      M2[i, which(M2[i,]==M2[i, z])[1]] = x
      if (which(M2[j,]==x) > 1){
       y = which(M2[j,]==x)
       M2[j, which(M2[j,]==x)] = M2[j, which(M2[j,]==x) - 1]
       M2[j, which(M2[j,]==M2[j, y])[1]] = x
       if (which(M2[l,]==x) > 1){
        w = which(M2[l,]==x)
        M2[l, which(M2[l,]==x)]=M2[l, which(M2[l,]==x) - 1]
        M2[l, which(M2[l,]==M2[l, w])[1]] = x
        ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
        ord2 = modelo(mat_cuadrada(M))[c(1:u)]
        if (which(ord1==x)<which(ord2==x)){
         dif = which(ord2==x)-which(ord1==x)
         if (max<dif){
          max = dif
          max_fila1 = i
          max_fila2 = j
          max_fila3 = l
          max_ord = ord1
          max_M2 = M2
          max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
         }
         else if (max==dif){
          if (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(
max_M2))[-c(1:u)]){
           max = dif
           max_fila1 = i
```

```
                max_fila2 = j
                max_fila3 = l
                max_ord = ord1
                max_M2 = M2
                max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
              }
             }
            }
           }
          }
         }
        }
        M2 = M
      }
    }
  }
}
if (max_fila1>0){
  if (max_fila3>0){
    cat("\n Con tres cambios, lo mejor que puedes hacer es mejorar una posición en las filas",max_fila1,",",max_fila2,"y",
        max_fila3,"; donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
        "con una puntuación de",max_modelo)
  }
  else{
    cat("\n Con tres cambios, lo mejor que puedes hacer es mejorar dos posiciones en las fila",max_fila1,"y una posición en la fila",max_fila2,
        "; donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
        "con una puntuación de",max_modelo)
  }
}else{
```

```
  cat("\n Con tres cambios, lo mejor que puedes hacer es mejorar tres posiciones en la fila",max_fila,
      ", donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
      "con una puntuación de",max_modelo)
}
}

###### Cambiar 4 posiciones ######
max_fila1 = 0
max_fila2 = 0
max_fila3 = 0
max_fila4 = 0
max_fila = 0
if (t==4){
## Para cambiar 4 posiciones en la misma fila ##
for (i in 1:n){
  if (which(M2[i,]==x) > 4){
    z = which(M2[i,]==x)
    M2[i, which(M2[i,]==x)] = M2[i, which(M2[i,]==x) - 1]
    M2[i, which(M2[i,]==M2[i, z])[1]] = M2[i, which(M2[i,]==M2[i, z])[1] - 1]
    M2[i, which(M2[i,]==M2[i, z]) - 2] = M2[i, which(M2[i,]==M2[i, z]) - 3]
    M2[i, which(M2[i,]==M2[i, z]) - 3] = M2[i, which(M2[i,]==M2[i, z]) - 4]
    M2[i, which(M2[i,]==M2[i, z]) - 4] = x
    ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
    ord2 = modelo(mat_cuadrada(M))[c(1:u)]
    if (which(ord1==x)<which(ord2==x)){
      dif = which(ord2==x) - which(ord1==x)
      if (max<dif){
        max = dif
        max_fila = k
        max_ord = ord1
        max_M2 = M2
        max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
```

```
      }
      else if (max==dif){
        if  (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(max_M2))[-
c(1:u)]){
          max = dif
          max_fila = k
          max_ord = ord1
          max_M2 = M2
          max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
        }
      }
    }
    M2 = M
  }
}
```

Para cambiar 2 posiciones en 2 filas

```
max = 0
for (i in 1:n){
  for (j in 1:n){
    if (i<j){
      if (which(M2[i,]==x) > 2){
        z = which(M2[i,]==x)
        M2[i, which(M2[i,]==x)] = M2[i,which(M2[i,]==x) - 1]
        M2[i, which(M2[i,]==M2[i, z])[1]] = M2[i, which(M2[i,]==M2[i, z])[1] - 1]
        M2[i, which(M2[i,]==M2[i, z]) - 2] = x
        if (which(M2[j,]==x) > 2){
          z = which(M2[j,]==x)
          M2[j, which(M2[j,]==x)] = M2[j, which(M2[j,]==x) - 1]
          M2[j, which(M2[j,]==M2[j, z])[1]] = M2[j, which(M2[j,]==M2[j, z])[1] - 1]
          M2[j, which(M2[j,]==M2[j, z]) - 2] = x
          ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
          ord2 = modelo(mat_cuadrada(M))[c(1:u)]
          if (which(ord1==x)<which(ord2==x)){
            dif = which(ord2==x) - which(ord1==x)
```

```
            if (max<dif){
              max = dif
              max_fila1 = i
              max_fila2 = j
              max_ord = ord1
              max_M2 = M2
              max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
            }
            else if (max==dif){
              if (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(
max_M2))[-c(1:u)]){
                max = dif
                max_fila1 = i
                max_fila2 = j
                max_ord = ord1
                max_M2 = M2
                max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
              }
            }
          }
        }
      }
    }
    M2 = M
  }
}
## Para cambiar 1 vez en 4 filas ##
for (i in 1:n){
  for (j in 1:n){
    if (i<j){
      for (l in 1:n){
        if (j<l){
          for (p in 1:n){
            if (l<p){
```

```
            if (which(M2[i,]==x) > 1){
              z = which(M2[i,]==x)
              M2[i, which(M2[i,]==x)] = M2[i,which(M2[i,]==x) - 1]
              M2[i, which(M2[i,]==M2[i, z])[1]] = x
              if (which(M2[j,]==x) > 1){
                y = which(M2[j,]==x)
                M2[j, which(M2[j,]==x)] = M2[j, which(M2[j,]==x) - 1]
                M2[j, which(M2[j,]==M2[j, y])[1]] = x
                if (which(M2[l,]==x) > 1){
                  w = which(M2[l,]==x)
                  M2[l, which(M2[l,]==x)] = M2[l, which(M2[l,]==x) - 1]
                  M2[l, which(M2[l,]==M2[l, w])[1]] = x
                  if (which(M2[p,]==x) > 1){
                    x = which(M2[p,]==x)
                    M2[p, which(M2[p,]==x)] = M2[p, which(M2[p,]==x) - 1]
                    M2[p, which(M2[p,]==M2[p, x])[1]] = x
                    ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
                    ord2 = modelo(mat_cuadrada(M))[c(1:u)]
                    if (which(ord1==x)<which(ord2==x)){
                      dif = which(ord2==x) - which(ord1==x)
                      if (max<dif){
                        max = dif
                        max_fila1 = i
                        max_fila2 = j
                        max_fila3 = l
                        max_fila4 = p
                        max_ord = ord1
                        max_M2 = M2
                        max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
                      }
                      else if (max==dif){
                        if (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(
max_M2))[-c(1:u)]){
                          max = dif
```

```
                    max_fila1 = i
                    max_fila2 = j
                    max_fila3 = l
                    max_fila4 = p
                    max_ord = ord1
                    max_M2 = M2
                    max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
                  }
                 }
                }
               }
              }
             }
            }
           }
          M2 = M
         }
        }
       }
      }
     }
}
if (max_fila1>0){
  if (max_fila3>0){
    cat("\n Con cuatro cambios, lo mejor que puedes hacer es mejorar una posición en las filas",max_fila1,",",max_fila2,",",max_fila3,"y",
        max_fila4,"; donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
        "con una puntuación de",max_modelo)
  }
  else{
    cat("\n Con cuatro cambios, lo mejor que puedes hacer es mejorar dos posiciones en las filas",max_fila1,"y",max_fila2,
```

```
      "; donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
      "con una puntuación de",max_modelo)
  }
}else{
  cat("\n Con cuatro cambios, lo mejor que puedes hacer es mejorar cuatro posiciones en la fila",max_fila,
      ", donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
      "con una puntuación de",max_modelo)
}
}

###### Cambiar 5 posiciones ######
max_fila1 = 0
max_fila2 = 0
max_fila3 = 0
max_fila4 = 0
max_fila5 = 0
max_fila = 0
if (t==5){
## Para cambiar 5 posiciones en la misma fila ##
for (i in 1:n){
  if (which(M2[i,]==x) > 5){
    z = which(M2[i,]==x)
    M2[i, which(M2[i,]==x)] = M2[i, which(M2[i,]==x) - 1]
    M2[i, which(M2[i,]==M2[i, z])[1]] = M2[i, which(M2[i,]==M2[i, z])[1] - 1]
    M2[i, which(M2[i,]==M2[i, z]) - 2] = M2[i, which(M2[i,]==M2[i, z]) - 3]
    M2[i, which(M2[i,]==M2[i, z]) - 3] = M2[i, which(M2[i,]==M2[i, z]) - 4]
    M2[i, which(M2[i,]==M2[i, z]) - 4] = M2[i, which(M2[i,]==M2[i, z]) - 5]
    M2[i, which(M2[i,]==M2[i, z]) - 5] = x
    ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
    ord2 = modelo(mat_cuadrada(M))[c(1:u)]
    if (which(ord1==x)<which(ord2==x)){
```

```
      dif = which(ord2==x) - which(ord1==x)
      if (max<dif){
        max = dif
        max_fila = k
        max_ord = ord1
        max_M2 = M2
        max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
      }
      else if (max==dif){
        if  (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(mat_cuadrada(max_M2))[-
c(1:u)]){
          max = dif
          max_fila = k
          max_ord = ord1
          max_M2 = M2
          max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
        }
      }
    }
    M2 = M
  }
}
## Para cambiar 1 vez en 5 filas ##
for (i in 1:n){
  for (j in 1:n){
    if (i<j){
      for (l in 1:n){
        if (j<l){
          for (p in 1:n){
            if (l<p){
              for (s in 1:n){
                if (p<s){
                  if (which(M2[i,]==x) > 1){
                    z = which(M2[i,]==x)
```

```
M2[i, which(M2[i,]==x)] = M2[i, which(M2[i,]==x) - 1]
M2[i, which(M2[i,]==M2[i, z])[1]] = x
if (which(M2[j,]==x) > 1){
  z = which(M2[j,]==x)
  M2[j, which(M2[j,]==x)] = M2[j, which(M2[j,]==x) - 1]
  M2[j, which(M2[j,]==M2[j, z])[1]] = x
  if (which(M2[l,]==x) > 1){
    z = which(M2[l,]==x)
    M2[l, which(M2[l,]==x)] = M2[l, which(M2[l,]==x) - 1]
    M2[l, which(M2[l,]==M2[l, z])[1]] = x
    if (which(M2[p,]==x) > 1){
      z = which(M2[p,]==x)
      M2[p, which(M2[p,]==x)] = M2[p, which(M2[p,]==x) - 1]
      M2[p, which(M2[p,]==M2[p, z])[1]] = x
      if (which(M2[s,]==x) > 1){
        z = which(M2[s,]==x)
        M2[s, which(M2[s,]==x)] = M2[s, which(M2[s,]==x) - 1]
        M2[s, which(M2[s,]==M2[s, z])[1]] = x
        ord1 = modelo(mat_cuadrada(M2))[c(1:u)]
        ord2 = modelo(mat_cuadrada(M))[c(1:u)]
        if (which(ord1==x)<which(ord2==x)){
          dif = which(ord2==x) - which(ord1==x)
          if (max<dif){
            max = dif
            max_fila1 = i
            max_fila2 = j
            max_fila3 = l
            max_fila4 = p
            max_fila5 = s
            max_ord = ord1
            max_M2 = M2
            max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
          }
          else if (max==dif){
```

```
                        if (modelo(mat_cuadrada(M2))[-c(1:u)]>modelo(
mat_cuadrada(max_M2))[-c(1:u)]){
                          max = dif
                          max_fila1 = i
                          max_fila2 = j
                          max_fila3 = l
                          max_fila4 = p
                          max_fila5 = s
                          max_ord = ord1
                          max_M2 = M2
                          max_modelo = modelo(mat_cuadrada(M2))[-c(1:u)]
                        }
                       }
                      }
                     }
                    }
                   }
                  }
                 }
                M2 = M
               }
              }
             }
            }
           }
          }
         }
        }
       }
      }
     }
    }
   }
  }
}
if (max_fila1>0){
  cat("\n Con cinco cambios, lo mejor que puedes hacer es mejorar una posición
en las filas",max_fila1,",",max_fila2,",",max_fila3,",",
      max_fila4,"y",max_fila5,"; donde se te queda un orden
de",max_ord,"aumentando tu posición en",max,
```

```
      "con una puntuación de",max_modelo)
}else{
  cat("\n Con cinco cambios, lo mejor que puedes hacer es mejorar 5 posiciones en la fila",max_fila,
      ", donde se te queda un orden de",max_ord,"aumentando tu posición en",max,
      "con una puntuación de",max_modelo)
}
}
}
```