Práctica: *Programación Lineal.*
Heurística y Optimización
Curso 2025-2026

1. Objetivo

El objetivo de esta práctica es que el alumno aprenda a modelar y resolver tareas de Programación Lineal de dos maneras diferentes: hojas de cálculo y desarrollo de programas que integren librerías de modelización y resolución.

2. Enunciado del problema

Una compañía nacional de autobuses intenta minimizar el impacto de cualquier incidencia en sus autobuses y, para ello, desea resolver diferentes problemas de optimización que mejoren sus prestaciones al público, al mismo tiempo que maximizan sus beneficios.

2.1. Parte 1: Modelo básico en Calc

La compañía dispone, en una de sus regiones, de hasta 5 talleres numerados como t_1, t_2, ..., t_5, cada uno de los cuales se encuentra a una determinada distancia del comienzo de la línea de cinco autobuses diferentes, a_1 a_2, ..., a_5. La Tabla 1 muestra estas distancias donde la celda c_{ij} es la distancia, medida en kilómetros, a la que se encuentra el taller t_i de la posición del autobús a_j.

En previsión de que uno, o más autobuses, tuvieran alguna avería al inicio del servicio, se desea realizar una asignación de autobuses a talleres, de tal modo que la distancia recorrida por todos los autobuses, desde su posición hasta el taller asignado, sea la mínima posible. Cada autobús debe ser asignado a un taller, y ningún taller puede tener asignado más de un autobús.

Se pide:

1. Modelar el problema como un problema de Programación Lineal entera, indicando claramente el tipo de problema de qué se trata, el significado de cada variable de decisión, el propósito de cada restricción, y la formalización de la función objetivo.

2. Implementar el modelo en una hoja de cálculo (LibreOffice), y resolverlo indicando claramente el valor de todas las variables de decisión, y de la función objetivo en la solución óptima encontrada.

2.2. Parte 2: Modelo avanzado en GLPK

Obviamente, la organización de la compañía no es igual en todo el territorio nacional, de modo que a continuación se pide resolver dos problemas de optimización diferentes cuya solución debe darse, cada una, con dos ficheros:

	a_1	a_2	a_3	a_4	a_5
t_1	206	66	263	104	154
t_2	285	86	111	300	110
t_3	164	251	109	220	177
t_4	264	175	217	147	149
t_5	141	75	218	87	228

Tabla 1: Distancia (en kms) de cada autobús a cada taller.

- Fichero de modelo (`.mod`): contiene el modelo *general* que resuelve la tarea de optimización, **para cualquier caso**.
- *Script* desarrollado en Python 3.8 (o posterior) que sirva para generar el fichero de datos `.dat` específico para cada tarea de optimización, de acuerdo a los valores de un fichero de entrada, y que resuelva óptimamente el problema usando, para ello, el fichero de datos generado, junto con el modelo general implementado.

2.2.1. Minimización del impacto de averías

Como en el caso del apartado 2.1, se desea minimizar la pérdida que pueda resultar de averías inesperadas al inicio del servicio pero, a diferencia de aquel caso, sólo se debe considerar un único taller t que ofrece hasta n franjas diferentes (cada una de los cuales es lo suficientemente amplia como para atender cualquier incidencia) numeradas como s_1, s_2, ..., s_n. Asimismo, el modelo debe considerar la existencia de una cantidad m de autobuses: a_1, a_2, ..., a_m, de cada uno de los cuales se sabe:

- d_i: la distancia a la que se encuentra el autobús a_i del taller t, $1 \leq i \leq m$.
- p_i: la cantidad de pasajeros que han contratado el servicio del autobús i-ésimo, a_i, $1 \leq i \leq m$.

Nótese que n y m no tienen por qué ser iguales y, en particular, puede que el taller oferte menos franjas de atención, n, que el número de autobuses que pueden llegar a requerir atención, m. Por ello, el problema define, además, dos tipos diferentes de costes por autobús:

- $k^d d_i$: donde k^d es una constante que representa la cantidad de euros por kilómetro que se deben asumir, en caso de que el autobús a_i sea asignado a alguna franja horaria del taller, $1 \leq i \leq m$.
- $k^p p_i$: donde k^p es una constante cualquiera que representa la penalización en euros por pasajero, en caso de no asignar al autobús i-ésimo alguna franja del taller, $1 \leq i \leq m$.

Obviamente, cada franja ofertada por el taller sólo puede ser ocupada por un autobús como máximo o, de lo contrario, no se garantizaría la reparación necesaria.

La resolución del problema debe implementarse con GLPK usando un script en Python 3.8 (o posterior), que se debe llamar `gen-1.py` que recibe exactamente dos argumentos: `fichero-entrada`, y `fichero-salida`. A continuación se muestran diferentes casos de uso:

```
$ ./gen-1.py ejemplo.in ejemplo.dat
$ ./gen-1.py /tmp/ejemplo.in ejemplo.dat
$ ./gen-1.py ejemplo.in ../../datos/ejemplo.dat
$ ./gen-1.py /tmp/ejemplo.in ../../datos/ejemplo.dat
```

Como se ve, cada fichero puede indicarse con una ruta absoluta o relativa, y si no contiene ninguna entonces se asume aquella en la que se encuentra el *script*, que debe:

- Generar un fichero de datos, con el nombre y la extensión indicada como segundo argumento, que codifique la información dada en el fichero de entrada, de acuerdo a las decisiones de diseño tomadas en el modelo implementado en el fichero `.mod`. Este fichero no debe ser borrado por el *script*.
- Resolver óptimamente la tarea correspondiente al caso de entrada, invocando el *solver* de GLPK con el modelo y el fichero de datos generado.
- Mostrar en pantalla la solución obtenida **indicando en la primera línea el valor óptimo de la función objetivo, así como el número de variables de decisión y restricciones totales creadas**, y en las siguientes líneas la solución óptima calculada indicando de manera legible qué autobuses se asignan a qué franjas horarias del taller, y cuáles se quedan sin asignar —en caso de que el número de franjas, n, sea menor que el número de autobuses, m.

El fichero de entrada, indicado en primer lugar, debe tener exactamente el siguiente formato:

<número de franjas, n><número de autobuses, m>
$<k^d><k^p>$
$<d_1, d_2, \ldots, d_m>$
$<p_1, p_2, \ldots, p_m>$

Se pide:

1. Modelar la tarea de optimización con un modelo general, con Programación Lineal entera, que minimice el coste total.

   Está expresamente prohibida la escritura de restricciones, o la función objetivo con operadores que no sean puramente lineales.

2. Implementar un modelo general que sirva para resolver la tarea de optimización propuesta, que se debe llamar `parte-2-1.mod`.

3. Desarrollar un *script* en Python, de acuerdo a las indicaciones dadas, que sirva para generar ficheros de datos `.dat`, y para resolver óptimamente la tarea de Programación Lineal resultante.

2.2.2. Maximización de la satisfacción de los pasajeros

En vez de minimizar las pérdidas ocasionadas por incidencias inesperadas al inicio de cualquier línea, en este segundo modelo se desea minimizar el impacto sobre los clientes de estas posibles averías, en una región de la empresa en la que puede haber una cantidad u de talleres t_1, t_2, …, t_u, y hasta m autobuses a_1, a_2, …, a_m. Como en la sección 2.2.1, cada taller oferta la misma disponibilidad de atención a incidencias en n franjas s_1, s_2, …, s_n, de modo que cada autobús puede ser asignado a una y sólo una franja, *si está disponible*, de cualquiera de los talleres.

Para la resolución de esta tarea, se debe conocer:

- c_{ij}: número de pasajeros que han contratado simultáneamente los servicios de los autobuses i- y j-ésimo, $1 \leq i, j \leq m$.
- o_{ij}: indica si la franja i-ésima está disponible en el taller j-ésimo, $1 \leq i \leq n, 1 \leq j \leq u$. Si bien cada uno de los u talleres disponibles ofertan hasta n franjas de atención, algunas podrían estar reservadas para otros usos si este parámetro lo indica explícitamente así —y esta indicación puede hacerse como se desee.

Obviamente, cada franja ofertada por cada taller sólo puede ser ocupada por un autobús como máximo o, de lo contrario, no se garantizaría la reparación necesaria, y en ningún caso deben asignarse aquellas franjas que no estén disponibles de acuerdo al parámetro o_{ij}. Además, todos los autobuses deben ser asignados a una franja (y sólo una) de algún taller, de modo que todos deben disponer de un taller asignado en caso de avería.

La resolución del problema debe implementarse con GLPK usando un script en Python 3.8 (o posterior), que se debe llamar `gen-2.py` que recibe exactamente dos argumentos: `fichero-entrada`, y `fichero-salida`. A continuación se muestran diferentes casos de uso:

```
$ ./gen-2.py ejemplo.in ejemplo.dat
$ ./gen-2.py /tmp/ejemplo.in ejemplo.dat
$ ./gen-2.py ejemplo.in ../../datos/ejemplo.dat
$ ./gen-2.py /tmp/ejemplo.in ../../datos/ejemplo.dat
```

Como se ve, cada fichero puede indicarse con una ruta absoluta o relativa, y si no contiene ninguna entonces se asume aquella en la que se encuentra el *script*, que debe:

- Generar un fichero de datos, con el nombre y la extensión indicada como segundo argumento, que codifique la información dada en el fichero de entrada, de acuerdo a las decisiones de diseño tomadas en el modelo implementado en el fichero `.mod`. Este fichero no debe ser borrado por el *script*.
- Resolver óptimamente la tarea correspondiente al caso de entrada, invocando el *solver* de GLPK con el modelo y el fichero de datos generado.
- Mostrar en pantalla la solución obtenida **indicando en la primera línea el valor óptimo de la función objetivo, así como el número de variables de decisión y restricciones totales creadas**, y en las siguientes líneas la solución óptima calculada indicando de manera legible qué autobuses se asignan a qué franjas de qué talleres.

El fichero de entrada, indicado en primer lugar, debe tener exactamente el siguiente formato:

<número de franjas, n><número de autobuses, m><número de talleres, u>
$<c_{11}c_{12}\ldots c_{1m}>$
$<c_{21}c_{22}\ldots c_{2m}>$
$\ldots$
$<c_{m1}c_{m2}\ldots c_{mm}>$
$<o_{11}o_{12}\ldots o_{1u}>$
$<o_{21}o_{22}\ldots o_{2u}>$
$\ldots$
$<o_{n1}o_{n2}\ldots o_{nu}>$

Se pide:

1. Modelar la tarea de optimización con un modelo general, con Programación Lineal entera, que minimice el número total de usuarios que están asignados a la misma franja horaria en talleres diferentes.

   Está expresamente prohibida la escritura de restricciones, o la función objetivo con operadores que no sean puramente lineales.

2. Implementar un modelo general que sirva para resolver la tarea de optimización propuesta, que se debe llamar `parte-2-2.mod`.

3. Desarrollar un *script* en Python, de acuerdo a las indicaciones dadas, que sirva para generar ficheros de datos `.dat`, y para resolver óptimamente la tarea de Programación Lineal resultante.

2.3. Parte 3: Análisis de Resultados

En este apartado se deben analizar todos los resultados obtenidos de cada parte, describiendo la solución obtenida (comprobando que cumple con las restricciones del enunciado) y analizando qué restricciones están limitando la solución del problema.

Análisis de la complejidad del problema: ¿cuántas variables y restricciones has definido?, crea diferentes casos con el uso del *script*, variando los parámetros, y explica cómo estas modificaciones afectan a la dificultad de resolución del problema resultante.

3. Directrices para la Memoria

La memoria debe entregarse en formato .pdf y tener un máximo de 15 hojas en total, incluyendo la portada, contraportada e índice. Debe tratarse de un documento técnico que contenga, al menos:

1. Breve introducción explicando los contenidos del documento.

2. Descripción de los modelos, argumentando las decisiones tomadas, usando para ello una notación algebraica correcta, clara y concisa.

3. Análisis de los resultados, según se indica en la sección 2.3.

La memoria **no debe incluir código fuente** en ningún caso.

4. Evaluación

La evaluación de la práctica se realizará sobre 10 puntos. Para que la práctica sea evaluada deberá realizarse al menos la primera mitad de la segunda parte (sección 2.2.1), esto es, tanto la resolución con GLPK usando un *script* en Python, como la entrega de la memoria de esa parte. La distribución de puntos es la siguiente:

1. Parte 1, sección 2.1 (2 puntos)
    - Modelización del problema (1 punto)
    - Implementación del modelo (1 punto)
2. Parte 2 (7 puntos)
    - Parte 1, sección 2.2.1 (3 puntos)
        - Modelización del problema (1,5 puntos)
        - Implementación del modelo (1,5 puntos)
    - Parte 2, sección 2.2.2 (4 puntos)
        - Modelización del problema (2 puntos)
        - Implementación del modelo (2 puntos)
3. Parte 3, sección 2.3 (1 punto):
    - Parte 1, sección 2.2.1 (0,5 puntos)
    - Parte 2, sección 2.2.2 (0,5 puntos)

En la evaluación de la modelización del problema, un modelo correcto supondrá la mitad de los puntos. Para obtenerse el resto de puntos, la modelización del problema deberá:

- Ser formalizada correctamente en la memoria, usando para ello una notación algebraica correcta, clara y concisa.
- Detallar claramente la utilidad de cada variable y restricción, así como las colecciones, si las hubiera, de parámetros, y conjuntos.
- Justificar detalladamente en la memoria todas las decisiones de diseño tomadas.

En la evaluación de la implementación del modelo, un modelo correcto supondrá la mitad de los puntos. Para obtenerse el resto de puntos, la implementación del problema deberá:

- Hacer uso (en la implementación) de las capacidades que ofrecen las herramientas para que hacer/actualizar el modelo sea lo más sencillo posible (por ejemplo, utilizar `sumaproducto` si es posible en el caso de la hoja de cálculo o el uso de `sets` en MathProg).
- Mantener el código (hoja de cálculo o ficheros de MathProg) correctamente organizado y comentado. Los nombres deben ser descriptivos. Deberán añadirse comentarios en los casos en que sea preciso mejorar la legibilidad.

Importante: los modelos implementados en la hoja de cálculo y GLPK deben ser correctos. Esto es, han de funcionar y obtener soluciones óptimas a cada problema propuesto. En ningún caso se obtendrá una calificación superior a 1 punto por un modelo que no lo haga. Por tanto, si la parte 1 no funciona correctamente, la nota máxima, en esa parte, será de 1 punto; y, si cualquiera de las otras dos no lo estuviera, como mucho se obtendrá un punto en cada una.

Asimismo, se debe tener en cuenta que se considera que la práctica no funciona correctamente, si bien el modelo no es capaz de resolver un caso razonable, o si el *script* en Python no funciona correctamente, por ejemplo, porque genera ficheros de datos erróneos, o en rutas diferentes de la indicada, o no escribe correctamente la solución óptima, etc.

IA generativa: el uso de herramientas de Inteligencia Artificial para realizar la práctica está expresamente prohibido, si bien es posible usarlas con el propósito de mejorar o razonar sobre modelos desarrollados previamente.
En particular, la sospecha del uso fraudulento de herramientas de este tipo, o cualquier otra consideración que anime a creer que algún miembro de un equipo, o ambos, no han realizado la práctica íntegramente, es motivo suficiente **para hacer una evaluación oral paralela a la corrección de esta práctica**.

5. Entrega

Se tiene de plazo para entregar la práctica hasta el 31 de Octubre, Viernes, a las 23:55. Este límite es fijo y no se extenderá de ningún modo.

Sólo un miembro de cada pareja debe subir los siguientes ficheros:

- Un único fichero `.zip` a través del punto de entrega de 'Aula Global' llamado "*Primera práctica*".

  El fichero debe nombrarse `p1-NIA1-NIA2.zip`, donde `NIA1` y `NIA2` son los últimos 6 dígitos del NIA (rellenando con `0`s por la izquierda si fuera preciso) de cada miembro de la pareja.

  Ejemplo: `p1-054000-671342.zip`.

- La memoria debe subirse separadamente a través del punto de entrega Turnitin de 'Aula Global' llamado "*Primera práctica (sólo pdf)*".

  La memoria debe llamarse `NIA1-NIA2.pdf` —después de sustituir adecuadamente los NIAs de cada estudiante como en el caso anterior.

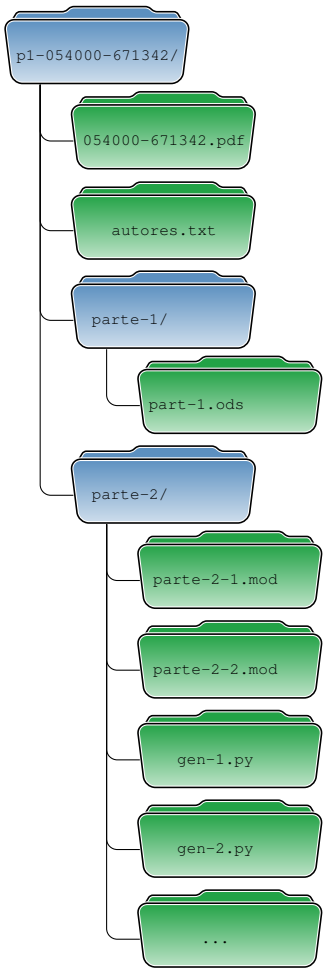
  Ejemplo: `054000-671342.pdf`.

La descompresión del fichero `.zip` descrito en el primer punto debe producir un directorio llamado `p1-NIA1-NIA2`, donde `NIA1` y `NIA2` son los últimos 6 dígitos del NIA de cada estudiante rellenando con `0`s si fuera preciso. Este directorio debe contener: primero, la misma memoria entregada a través del segundo enlace descrito anteriormente que debe nombrarse como `NIA1-NIA2.pdf` —después de sustituir convenientemente los NIAs de cada estudiante; segundo, un fichero llamado `autores.txt` que identifica a los miembros de cada pareja, con una línea por cada uno con el siguiente formato: `NIA Surname, Name`. Por ejemplo:

```
054000 Von Neumann, John
671342 Turing, Alan
```

Además, este directorio debe contener un directorio por cada parte realizada llamados "`parte-1`" y "`parte-2`". Las soluciones de cada parte (tanto el código fuente como la hoja de cálculo generada) deben incluirse en sus respectivos directorios.

La siguiente figura muestra una distribución posible de los ficheros después de descomprimir el fichero `.zip`:



Como se puede ver, el directorio "`parte-2`" puede contener otros ficheros adicionales (indicados en la figura con puntos suspensivos). Son ejemplos válidos: ficheros adicionales de Python necesarios para la ejecución de los scripts, o ficheros de datos de ejemplo de casos resueltos que se discutan en la memoria, pero en ningún caso deben entregarse ficheros generados por el solver de GLPK con la solución óptima.

Importante: no seguir las normas de entrega puede suponer una pérdida de hasta 1 punto en la calificación final de la práctica.