Modelado y Optimización – Trabajo Práctico

Ximena Ebertz Hernán Rondelli Lucía Soria

2do Semestre 2024

Contenidos

1 Introducción

Este trabajo práctico tiene como objetivo aplicar conceptos de modelado y de optimización, utilizando técnicas de programación lineal, y distintas tecnologías y herramientas. Buscamos, a través de una serie de problemas relacionados con la gestión de archivos en una empresa de datos, diseñar soluciones eficientes que optimicen el uso de recursos, como el almacenamiento en discos y la importancia de los datos.

La actividad se divide en seis partes, cada una con enfoques específicos que abarcan desde el modelado directo en `SCIP`, hasta la implementación de algoritmos en Python con la biblioteca `PySCIPOpt`. A lo largo de este trabajo, se considerarán restricciones como la capacidad limitada de los discos, la importancia de los archivos, y la estructura de conjuntos para maximizar la eficiencia del almacenamiento y la relevancia de los datos seleccionados.

2 Parte 1

2.1 Descripción

Se deben realizar backups de n archivos de la empresa *BigData* en discos idénticos de capacidad d, sin dividir los archivos. Cada archivo es menor que la capacidad de un disco. Se necesita saber cuál es la cantidad mínima de discos necesarios para almacenar todos los archivos, considerando la capacidad de cada disco y el tamaño de cada archivo.

2.2 Modelo Lineal

Para minimizar la cantidad de discos, usamos una variable binaria y_j que nos indica si se está usando el disco j o no, y otra variable binaria x_{ij} para identificar si cada archivo i se guarda en el disco j. Luego, establecimos dos restricciones: la primera se asegura de que cada archivo esté en un solo disco, y la segunda, que los tamaños de los archivos de cada disco no superen la capacidad del disco.

2.2.1 Definición de variables

n: Constante, cantidad de archivos.

s_i: Constante, tamaño del archivo i en MB, $\forall i \in \{1, \ldots, n\}$.

m: Cantidad de discos disponibles; a lo sumo, un archivo por disco, con $m = n$.

d: Constante, tamaño del disco en TB.

x_{ij}: Variable binaria, donde $x_{ij} = 1$ indica que el archivo i está en el disco j, $\forall i \in \{1, \ldots, n\}$, $\forall j \in \{1, \ldots, m\}$, con $x_{ij} \in \{0, 1\}$.

y_j: Variable binaria, donde $y_j = 1$ indica que se está usando el disco j, $\forall j \in \{1, \ldots, m\}$, con $y_j \in \{0, 1\}$.

2.2.2 Modelo

$$\begin{aligned}
\texttt{Minimize} \quad & \sum_{j=1}^{m} y_j \\
\texttt{Subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in \{1, \dots, n\}, \text{ (el archivo } i \text{ debe estar en un solo disco)} \\
& \sum_{i=1}^{n} s_i x_{ij} \leq d \cdot y_j \quad \forall j \in \{1, \dots, m\}, \text{ (los archivos que entran en el disco } j\text{)} \\
& x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\}, \ \forall j \in \{1, \dots, m\} \\
& y_j \in \{0, 1\}, \quad \forall j \in \{1, \dots, m\}
\end{aligned}$$

2.2.3 Restricción: que no se elijan discos vacíos

Inicialmente, habíamos planteado el programa lineal de la Sección 2.2 con una tercera restricción:

$$\sum_{i=1}^{n} x_{ij} \leq n \cdot y_j \quad \forall j \in \{1, \dots, m\}, \text{ (no se puede usar un disco vacío)}$$

En esta sección, explicamos porqué esta restricción—aunque no modificaba el resultado de la función objetivo—no es necesaria, mediante una demostración por el absurdo.

Supongamos que contamos con dos archivos f_1 y f_2, y dos discos, d_1 y d_2. En este caso, una solución óptima hipotética sería $y = (1, 1)$, que indica que se utilizan los dos discos.

Supongamos que esta solución considera que los archivos f_1 y f_2 se almacenan en el disco d_2 (considerando que ambos archivos entran en el disco), quedando el disco d_1 vacío. Si analizamos las restricciones, vemos que todas se cumplen. Es decir:

$$\sum_{j=1}^{2} x_{ij} = 1 \qquad \forall i \in \{1, 2\} \tag{1}$$

$$\sum_{i=1}^{2} s_i x_{ij} \leq d \cdot y_j \qquad \forall j \in \{1, 2\} \tag{2}$$

Vemos que (1) se cumple ya que ambos archivos f_1 y f_2 están en un sólo disco, d_2, por hipótesis. Por otro lado, (2) se cumple, ya que...

(i) $0 \leq d_1 \cdot 1$, ya que el disco está vacío.

(ii) $s_1 + s_2 \leq d_2 \cdot 1$, por hipótesis.

A parte, notar que $x_{ij} \in \{0, 1\}$, $\forall i \in \{1, 2\}$, $\forall j \in \{1, 2\}$, y $y_j \in \{0, 1\}$, $\forall j \in \{1, 2\}$ por hipótesis.

Queremos ver, a partir de esto, que existe una solución x^* mejor, por lo que x no sería una solución óptima.

Consideremos la solución $x^* = (0, 1)$. Vemos que también se cumplen las restricciones:

$$\sum_{j=1}^{2} x_{ij} = 1 \qquad \forall i \in \{1, 2\} \tag{3}$$

$$\sum_{i=1}^{2} s_i x_{ij} \leq d \cdot y_j \qquad \forall j \in \{1, 2\} \tag{4}$$

en (3) se cumple ya que ambos archivos f_1 y f_2 están en un sólo disco, d_2, por hipótesis, y (4) se cumple, ya que...

(i) $0 \leq d_1 \cdot 0$, ya que no se elige el disco 1.

(ii) $s_1 + s_2 \leq d_2 \cdot 1$, por hipótesis.

También se cumplen las condiciones con respecto a los valores que pueden tomar $x_{1,j}$ e y_j, ya que no fueron modificadas. Entonces, x^* es una solución factible.

Para x,

$$\sum_{j=1}^{m} y_j = 2.$$

Por otra parte, para x^*,

$$\sum_{j=1}^{m} y_j = 1.$$

Entonces, x^* es una mejor solución que x, por lo que x no puede ser una solución óptima. Este absurdo surgió de considerar que puede haber un disco vacío en una solución óptima de nuestro problema lineal. Esto implica que la restricción planteada no aporta valor al problema, ya que probamos que se llega a una solución correcta sin ella.

2.2.4 Modelo Zimpl

A continuación presentamos el modelo Zimpl, con los comentarios correspondientes que indican la relación con el modelo lineal de la sección anterior.

```
param input := "IN/a_1.in"; # Path del archivo de input

# Tamaño del disco en terabytes
param d_t := read input as "1n" use 1 comment "#";
param d := d_t * 10**6;

# Conjunto de archivos f_{i} \forall i \in \{1, \ldots, n\}
set F := { read input as "<1s>" skip 2 comment "#" };

param n := card(F); # Cantidad de archivos
param m := n; # Cantidad de discos, a lo sumo, un disco por archivo

# Conjunto de discos
set D := { 1 .. m };

# Tamaños de los archivos f_{i} \forall i \in \{1, \ldots, n\}
param s[F] := read input as "<1s> 2n" skip 2 comment "#";

# 1 si el archivo $i$ está en el disco $j$, 0 en caso contrario
var x[<i, j> in F * D] binary;

# 1 si se usa el disco $j$, 0 en caso contrario
var y[D] binary;

minimize disks: sum<j> in D: y[j];

# Cada archivo debe estar en un solo disco
subto c1:
    forall <i> in F:
        sum<j> in D: x[i, j] == 1;

# Los archivos $i$ que entran en el disco $j$
subto c2:
    forall <j> in D:
        sum<i> in F: s[i] * x[i, j] <= d * y[j];
```

2.2.5 Modelo PySCIPOpt

Como parte del trabajo, incluimos una implementación del modelo utilizando Python, con la librería PySCIPOpt. La siguiente implementación corresponde a una función equivalente al código presentado con Zimpl.

```python
import sys
from pyscipopt import Model
from pyscipopt import quicksum
from pyscipopt import SCIP_PARAMSETTING
from pyscipopt import *

############################################################################
   # d_t: capacidad del disco en TB,
   # F: nombres de los archivos,
   # s: tamaños de los archvios,
   # time_limit: threshold en segundos
############################################################################
def distribuir_archivos_1(d_t: int, F: list[str], s: list[int], time_limit=420):
   model, fake_y, fake_x = resolver_modelo_binario_1(d_t, F, s, time_limit)

   solution = model.getBestSol()
   status = model.getStatus()

   if solution is not None and status in ["optimal", "feasible"]:
      return [F, model, fake_y, fake_x, s]
   else:
      return None

def resolver_modelo_binario_1(d_t: int, F: list[str], s: list[int], time_limit=420):
   model = Model("model_part_1")
   model.setParam("limits/time", time_limit)

   d = d_t * 10**6

   n = len(F)
   m = n

   ############################################################################
   # BINARY
   # x_{i, j} = 1 si se elige el archivo i para el disco j, 0 si no
   #
   x = {}
   for i in range(n):
```

```
        for j in range(m):
            x[i, j] = model.addVar(name=f"x_{i}_{j}", vtype="BINARY")

    # y_{j} = 1 si se elige el disco j, 0 si no
    y = [model.addVar(f"y_{j}", vtype="BINARY") for j in range(m)]
    #
    ############################################################################

    # Minimizar la cantidad de discos
    model.setObjective(quicksum(y), sense="minimize")

    # Que los archivos se elijan solo para un disco
    for i in range(n):
        model.addCons(quicksum(x[i, j] for j in range(m)) == 1)

    # Que no se pasen de capacidad los discos
    for j in range(m):
        model.addCons(quicksum(x[i, j] * s[i] for i in range(n)) <= d * y[j])

    model.setParam("display/freq", 1)
    model.optimize()

    return model, y, x

def obtener_solucion_primal_1(model):
    sol = model.getBestSol()
    status = model.getStatus()

    if sol is not None and status in ["optimal", "feasible"]:
        x = [v.getLPSol() for v in model.getVars(False)]
        return x, model.getObjVal()
    else:
        return None

def obtener_solucion_dual_1(model):
    y = [model.getDualSolVal(c) for c in model.getConss(False)]
    return y, quicksum(y)
```

Esta función recibe los datos leídos de un archivo de configuración creado con datos aleatorios. El siguiente código muestra las funciones necesarias para generar este archivo.

```python
def generar_input_1(ruta_archivo: str):
    generar_input(ruta_archivo, False)
```

```python
def generar_input(ruta_archivo: str, importancia: bool):
    capacidad_discos = random.randint(1, 300)
    cant_archivos = random.randint(400, 550)
    archivos = generar_archivos(cant_archivos, importancia)

    with open(ruta_archivo, "w") as f:

        f.write(f"# Capacidad de dicos en TB (= 1.000.000 MB)\n")
        f.write(str(capacidad_discos) + "\n")

        f.write(f"\n# Cantidad de archivos para backup\n")
        f.write(str(cant_archivos) + "\n")

        if not importancia:
            f.write(f"\n# Archivos: archivo_id, tamaño (MB) \n")
            for archivo in archivos:
                f.write(archivo + " " + str(archivos[archivo]) + "\n")
        else:
            f.write(f"\n# Archivos: archivo_id, tamaño (MB), importancia\n")
            for archivo in archivos:
                f.write(archivo + " " + str(archivos[archivo][0]) + " " + str(archivos[
                    archivo][1]) + "\n")

def generar_archivos(cant_archivos: int, importancia: bool):
    archivos = {}
    contador = 1
    for i in range(cant_archivos):
        if not importancia:
            # {nombreArchivo: tamaño}
            archivos["archivo" + str(contador)] = random.randint(1000000, 10000000)
        else:
            # {nombreArchivo: [tamaño, importancia] }
            archivos["archivo" + str(contador)] = [
            random.randint(1000000, 10000000),
            random.randint(1, 10)
        ]
        contador += 1
    return archivos
```

La solución incluye un script ejecutable que recibe como parámetro el nombre del archivo a generar, genera el archivo y calcula el resultado del modelo. Luego, genera el output correspondiente.

```
#!/usr/bin/env python3
import os
import sys

sys.path.insert(0, "../utils")

import inputs
import configs
import outputs
import model_part_1

if len(sys.argv) != 3:
   print(f'Uso: {sys.argv[0]} OPTION archivo')
   print(f' OPTIONS: -g | -u | -c')
   print(f' -g generar archivo')
   print(f' -u usar archivo ya generado')
   print(f' -c usar configuración')
   sys.exit(1)

archivo = sys.argv[2]
archivos = []
threshold = 7
out_path = 'OUT'

if sys.argv[1] == '-g':
   print(f'Generando {archivo}\n')
   inputs.generar_input_1(os.path.dirname(__file__) + '/IN/' + archivo)
   archivos.append(archivo)

elif sys.argv[1] == '-u':
   print(f'Usando {archivo}\n')
   archivos.append(archivo)

elif sys.argv[1] == '-c':
   print(f'Leyendo configuración {archivo}\n')
   configuraciones = configs.leer_configuracion(os.path.join(os.path.dirname(
       __file__), archivo))
   out_path = configuraciones.get('outPath')[:-1]
   threshold = int(configuraciones.get('threshold', 0))
   archivos = [f for f in os.listdir(configuraciones.get('inPath'))]
```

```
40    archivos.remove('.gitkeep')
41
42
43 for archivo in archivos:
44    capacidad_disco, nombres_archivos, tamaños_archivos = inputs.leer_input_1(os.path
          .dirname(__file__) + '/IN/' + archivo)
45    solucion = model_part_1.distribuir_archivos_1(capacidad_disco, nombres_archivos,
          tamaños_archivos, threshold * 60)
46
47    archivo_out = os.path.join(os.path.dirname(__file__), out_path, f'{archivo[:-3]}.
          out')
48
49    if solucion is not None:
50       print(f'Se encontró solución. Generando {archivo_out}...')
51       outputs.generar_output_1(archivo_out, solucion)
52       print(f'Solución generada en {archivo_out}')
53    else:
54       print(f'No se encontró solución para {archivo}')
55       outputs.generar_output_fallido(archivo_out)
56       print(f'Solución fallida generada en {archivo_out}')
```

Utilizamos la siguiente función para generar el output.

```python
def generar_output_1(ruta_archivo, solucion):
    # solucion = [F, model, y, x, s]
    F = solucion[0]
    model = solucion[1]
    y = solucion[2]
    x = solucion[3]
    s = solucion[4]

    cant_archivos = len(F)
    cant_discos = round(float(model.getObjVal()))
    max_discos = cant_archivos

    with open(ruta_archivo, "w") as f:
        f.write(f"Para la configuracion del archivo, {cant_discos} discos son
            suficientes.\n")
        for j in range(max_discos):
            if model.getVal(y[j]) == 0:
                continue

            archivos_en_disco = []
            espacio_ocupado = 0

            for i in range(cant_archivos):
                if model.getVal(x[i, j]) == 0:
                    continue
                archivos_en_disco.append(f"{F[i]} {s[i]}")
                espacio_ocupado = espacio_ocupado + s[i]

            f.write(f"\nDisco {j+1}: {espacio_ocupado} MB\n")

            for archivo in archivos_en_disco:
                f.write(archivo + "\n")
```

3 Parte 2

3.1 Descripción

Se asigna un indicador de importancia I a cada archivo según características como sensibilidad, frecuencia de uso, y rareza. El objetivo es encontrar un subconjunto de archivos que quepan en un disco de tamaño d, y que la suma de indicadores de importancia sea máxima.

3.2 Modelo Lineal

Para maximizar la importancia de los archivos en un disco, usamos una variable binaria x_i que indica si el archivo i está o no en el disco, y la multiplicamos por la constante I_i que indica la importancia del archivo i. La restricción que utilizamos para este problema lineal, es que la suma de los tamaños de los archivos no superen el tamaño del disco.

3.2.1 Definición de variables

n: Constante, cantidad de archivos.

s_i: Constante, tamaño del archivo i en MB, con $i \in \{1, \ldots, n\}$.

I_i: Constante, indicador de importancia para el archivo i, con $i \in \{1, \ldots, n\}$.

d: Constante, tamaño del disco en TB.

x_i: Variable binaria, donde $x_i = 1$ indica que el archivo i está en el disco y $x_i = 0$ en caso contrario, con $x_i \in \{0, 1\}$, $\forall i \in \{1, \ldots, n\}$.

3.2.2 Modelo

$$
\begin{aligned}
\texttt{Maximize} \quad & \sum_{i=1}^{n} I_i \cdot x_i \\
\texttt{Subject to} \quad & \sum_{i=1}^{n} s_i x_i \leq d \quad \text{(archivos que entran en el disco)} \\
& x_i \in \{0, 1\}, \quad \forall i \in \{1, \ldots, n\}
\end{aligned}
$$

3.2.3 Modelo Zimpl

A continuación presentamos el modelo Zimpl, con los comentarios correspondientes que indican la relación con el modelo lineal de la sección anterior.

```
param input := "IN/b_1.in"; # Path del archivo de input

# Tamaño del disco en terabytes
param d_t := read input as "1n" use 1 comment "#";
param d := d_t * 10**6;

# Conjunto de archivos f_{i} \forall i \in \{1, \ldots, n\}
set F := { read input as "<1s>" skip 2 comment "#" };

param n := card(F); # Cantidad de archivos

# Tamaños de los archivos f_{i} \forall i \in \{1, \ldots, n\}
param s[F] := read input as "<1s> 2n" skip 2 comment "#";

# Importancias de los archivos
param importance[F] := read input as "<1s> 3n" skip 2 comment "#";

# 1 si el archivo $i$ está en el disco, 0 en caso contrario
var x[F] binary;

maximize importances: sum<i> in F: importance[i] * x[i];

# Los archivos $i$ que entran en el disco
subto c1:
    sum<i> in F: s[i] * x[i] <= d;
```

3.2.4 Modelo PySCIPOpt

De la misma manera que en el modelo anterior, generamos el modelo, la configuración y los archivos de output utilizando Python.

El modelo es el siguiente.

```python
import sys
from pyscipopt import Model
from pyscipopt import quicksum
from pyscipopt import SCIP_PARAMSETTING
from pyscipopt import *

############################################################################
   # d_t: capacidad del discoen TB,
   # F: nombres de los archivos,
   # s: tamaños de los archvios,
   # I: importancia de los archivos,
   # time_limit: threshold en segundos
############################################################################
def distribuir_archivos_2(d_t: int, F: list[str], s: list[int], I: list[float],
    time_limit=420):
   model, fake_x = resolver_modelo_binario_2(d_t, F, s, I, time_limit)

   solution = model.getBestSol()
   status = model.getStatus()

   if solution is not None and status in ["optimal", "feasible"]:
      return [F, model, fake_x, I, s]
   else:
      return None

def resolver_modelo_binario_2(d_t: int, F: list[str], s: list[int], I: list[float],
    time_limit=420):
   model = Model("model_part_2")
   d = d_t * 10**6
   n = len(F)

   ############################################################################
   # BINARIA
   # x_{i} = 1 si se elige el archivo i, 0 si no
   x = [model.addVar(f"x_{i}", vtype="BINARY") for i in range(n)]
   ############################################################################

   # Maximizar importancia:
```

```
    model.setObjective(quicksum(x[i] * I[i] for i in range(n)),
                       sense="maximize")

    # Los archivos elegidos deben entrar en el disco
    model.addCons(quicksum(x[i] * s[i] for i in range(n)) <= d)

    # Configurar el límite de tiempo en el solver
    model.setParam("limits/time", time_limit)
    model.optimize()

    return model, x

# Crea el modelo relajado y lo devuelve optimizado
def crear_modelo_2(d_t: int, F: list[str], s: list[int], I: list[float], time_limit
    =420):
    model = Model("model_part_2")

    d = d_t * 10**6
    n = len(F)

    ############################################################################
    # CONTINUA
    # x_{i} = 1 si se elige el archivo i, 0 si no
    x = [
       model.addVar(f"y_{i}", lb=0, ub=1, vtype="CONTINUOUS")
       for i in range(n)
    ]
    ############################################################################

    # MAximizar importancia:
    model.setObjective(quicksum(x[i] * I[i] for i in range(n)),
                       sense="maximize")

    # Loss archivos elegidos deben entrar en el disco
    model.addCons(quicksum(x[i] * s[i] for i in range(n)) <= d)

    # Configurar el límite de tiempo en el solver
    model.setParam("limits/time", time_limit)

    model.optimize()
    return model

def obtener_solucion_primal_2(model):
```

```
79    sol = model.getBestSol()
80    status = model.getStatus()
81
82    if sol is not None and status in ["optimal", "feasible"]:
83       x = [v.getLPSol() for v in model.getVars(False)]
84       return x, model.getObjVal()
85    else:
86       return None
87
88 def obtener_solucion_dual_2(model):
89    y = [model.getDualSolVal(c) for c in model.getConss(False)]
90    return y, quicksum(y)
```

Los archivos de configuración se generan de manera similar al modelo previo.

```python
def generar_input_2(ruta_archivo: str):
    generar_input(ruta_archivo, True)
```

```python
def generar_input(ruta_archivo: str, importancia: bool):
    capacidad_discos = random.randint(1, 300)
    cant_archivos = random.randint(400, 550)
    archivos = generar_archivos(cant_archivos, importancia)

    with open(ruta_archivo, "w") as f:

        f.write(f"# Capacidad de dicos en TB (= 1.000.000 MB)\n")
        f.write(str(capacidad_discos) + "\n")

        f.write(f"\n# Cantidad de archivos para backup\n")
        f.write(str(cant_archivos) + "\n")

        if not importancia:
            f.write(f"\n# Archivos: archivo_id, tamaño (MB) \n")
            for archivo in archivos:
                f.write(archivo + " " + str(archivos[archivo]) + "\n")
        else:
            f.write(f"\n# Archivos: archivo_id, tamaño (MB), importancia\n")
            for archivo in archivos:
                f.write(archivo + " " + str(archivos[archivo][0]) + " " + str(archivos[
                    archivo][1]) + "\n")

def generar_archivos(cant_archivos: int, importancia: bool):
    archivos = {}
    contador = 1
    for i in range(cant_archivos):
        if not importancia:
            # {nombreArchivo: tamaño}
            archivos["archivo" + str(contador)] = random.randint(1000000, 10000000)
        else:
            # {nombreArchivo: [tamaño, importancia] }
            archivos["archivo" + str(contador)] = [
            random.randint(1000000, 10000000),
            random.randint(1, 10)
        ]
        contador += 1
    return archivos
```

El script ejecutable realiza los mismos pasos que el del modelo anterior, pero, considerando la importancia de cada archivo.

```python
#!/usr/bin/env python3
import os
import sys

sys.path.insert(0, "../utils")

import inputs
import configs
import outputs
import model_part_2

if len(sys.argv) != 3:
   print(f'Uso: {sys.argv[0]} OPTION archivo')
   print(f' OPTIONS: -g | -u | -c')
   print(f' -g generar archivo')
   print(f' -u usar archivo ya generado')
   print(f' -c usar configuración')
   sys.exit(1)

archivo = sys.argv[2]
archivos = []
threshold = 7
out_path = 'OUT'

if sys.argv[1] == '-g':
   print(f'Generando {archivo}\n')
   inputs.generar_input_2(os.path.dirname(__file__) + '/IN/' + archivo)
   archivos.append(archivo)

elif sys.argv[1] == '-u':
   print(f'Usando {archivo}\n')
   archivos.append(archivo)

elif sys.argv[1] == '-c':
   print(f'Leyendo configuración {archivo}\n')
   configuraciones = configs.leer_configuracion(os.path.join(os.path.dirname(
       __file__), archivo))
   out_path = configuraciones.get('outPath')[:-1]
   threshold = int(configuraciones.get('threshold', 0))
   archivos = [f for f in os.listdir(configuraciones.get('inPath'))]
   archivos.remove('.gitkeep')
```

```python


for archivo in archivos:
    capacidad_disco, nombres_archivos, tamaños_archivos, importancia_archivos = inputs.leer_input_2(os.path.dirname(__file__) + '/IN/' + archivo)
    solucion = model_part_2.distribuir_archivos_2(capacidad_disco, nombres_archivos, tamaños_archivos, importancia_archivos, threshold * 60)

    archivo_out = os.path.join(os.path.dirname(__file__), out_path, f'{archivo[:-3]}.out')

    if solucion is not None:
        print(f'Se encontró solución. Generando {archivo_out}...')
        outputs.generar_output_2(archivo_out, solucion)
        print(f'Solución generada en {archivo_out}')
    else:
        print(f'No se encontró solución para {archivo}')
        outputs.generar_output_fallido(archivo_out)
        print(f'Solución fallida generada en {archivo_out}')
```

Luego, el output se almacena de la siguiente manera.

```python
def generar_output_2(nombre_archivo, solucion):
    # solución = [F, model, fake_x, I, s]
    F = solucion[0]
    model = solucion[1]
    x = solucion[2]
    I = solucion[3]
    s = solucion[4]

    cant_archivos = len(F)
    cant_archivos_elegidos = sum(1 for i in range(cant_archivos) if model.getVal(x[i
        ]) > 0.5)
    archivos_elegidos = []
    importancia_total = 0

    with open(nombre_archivo, "w") as f:
        f.write(f"Para la configuracion del archivo, se han elegido {
            cant_archivos_elegidos} archivos.\n")

        for i in range(cant_archivos):
            if model.getVal(x[i]) > 0.5:
                archivos_elegidos.append(f"{F[i]} {s[i]} {I[i]}")
                importancia_total += I[i]

        for archivo in archivos_elegidos:
            f.write(archivo + "\n")

        f.write(f"\nLa suma de sus indicadores de importancia da {importancia_total}."
            )
```

4 Parte 3

4.1 Descripción

Dado un conjunto de archivos C y una familia de conjuntos de archivos $\mathcal{H}$, donde cada conjunto de archivos H de $\mathcal{H}$ está incluido en C, y cada archivo de C pertenece a algún conjunto de la familia $\mathcal{H}$, se busca un subconjunto mínimo $\mathcal{I}$ de $\mathcal{H}$ tal que todos los archivos de C estén cubiertos por al menos un conjunto de $\mathcal{I}$.

4.2 Modelo Lineal

4.2.1 Definición de variables

n: Constante, cantidad de archivos.

m: Constante, cantidad de conjuntos H.

y_{ij}: Constante binaria, donde $y_{ij} = 1$ indica que el conjunto H_j contiene el archivo i, con $y_{ij} \in \{0, 1\}$, $\forall i \in \{1, \ldots, n\}$, $\forall j \in \{1, \ldots, m\}$.

x_j: Variable binaria, donde $x_j = 1$ indica que el conjunto H_j pertenece a I, con $x_j \in \{0, 1\}$, $\forall j \in \{1, \ldots, m\}$.

4.2.2 Modelo

$$\texttt{Minimize} \quad \sum_{j=1}^{m} x_j$$

$$\texttt{Subject to} \quad \sum_{j=1}^{m} y_{i,j} \cdot x_j \geq 1 \quad \forall i, \text{ (todos los archivos deben estar en al menos un conjunto elegido)}$$

$$x_j \in \{0, 1\}, \quad \forall j \in \{1, \ldots, m\}$$

5 Parte 4

5.1 Descripción

Se deben realizar backups de n archivos en discos idénticos de capacidad d. Se sabe que los archivos son de pocos tamaños distintos, y que cada archivo es menor que la capacidad de un disco. Se necesita saber cuál es la cantidad mínima de discos necesarios para almacenar todos los archivos, considerando la capacidad de cada disco y los tamaños de archivos.

5.2 Modelo Lineal

5.2.1 Definición de variables

d, constante: tamaño del disco en TB.

n, constante: cantidad de archivos.

t, constante: cantidad de tamaños distintos de archivos.

s_k, constante: tamaño k en MB, con $k \in \{1, \ldots, t\}$.

f_k, constante: cantidad de archivos de tamaño k en MB, con $k \in \{1, \ldots, t\}$.

x_p, entera: cantidad de veces que se usa el patrón p, con $p \in \{1, \ldots, q\}$, donde $x_p \geq 0$.

c_{kp}, constante entera: cantidad de archivos de tamaño k que entran en el patrón p, con $k \in \{1, \ldots, t\}$ y $p \in \{1, \ldots, q\}$.

5.2.2 Modelo

$$
\begin{aligned}
\texttt{Minimize} \quad & \sum_{p=1}^{q} x_p \\
\texttt{Subject to} \quad & \sum_{k=1}^{t} \sum_{p=1}^{q} s_k \cdot c_{kp} \cdot x_{jp} \leq d \qquad \text{(tamaño } k \text{ que entran por disco } d\text{)} \\
& \sum_{p=1}^{q} c_{kp} \cdot x_p \geq f_k \\
& x_p \geq 0, \quad \forall p \in \{1, \ldots, q\}
\end{aligned}
$$

6 Parte 5

6.1 Descripción

El objetivo de este ejercicio es resolver el problema de las Secciones 2 y 5 suponiendo que los tamaños de los archivos pertenecen a un conjunto pequeño. Para ello, se genera una familia de conjuntos H que cubra todos los archivos y quepa en los discos, se plantea un modelo de programación lineal relajado P y se resuelve su dual para asignar valores a los archivos. Usando estos valores como indicadores de importancia, se selecciona un subconjunto de archivos. Si la solución es mejorable, se actualiza H y se repite el proceso, ajustando finalmente la solución para que sea entera.

6.2 Pseudocódigo de alto nivel

```
INICIO obtener_conjuntos(archivo, threshold = infinito)
   # Leer los datos del archivo de entrada
   capacidad_disco, nombres_archivos, tamanos_archivos <- LeerInput5(ruta_archivo)
   tiempo_inicio <- TiempoActual()

   // PASO 1: Generar los conjuntos iniciales
   conjunto_H <- GenerarSubconjuntos(capacidad_disco * 10^6, nombres_archivos,
       tamanos_archivos)
   SI conjunto_H ES NULL ENTONCES
      RETORNAR NULL
   FIN SI

   MIENTRAS HAYA TIEMPO HACER
      // PASO 2 y 3: Resolver el modelo y obtener soluciones primal y dual
      modelo_P <- CrearModelo3(nombres_archivos, conjunto_H, threshold - (
          TiempoActual() - tiempo_inicio))
      x <- ObtenerSolucionPrimal3(modelo_P)
      SI x ES NULL ENTONCES
         TERMINAR CICLO
      FIN SI
      y <- ObtenerSolucionDual3(modelo_P)

      // PASO 4: Distribuir archivos y generar solucion
      solucion_modelo_2 <- ObtenerSolucion2(DistribuirArchivos2(capacidad_disco,
          nombres_archivos, tamanos_archivos, y, threshold - (TiempoActual() -
          tiempo_inicio)))
      SI solucion_modelo_2 ES NULL O NO HAY TIEMPO ENTONCES
         TERMINAR CICLO
      FIN SI

```

```
27      // PASO 5: Verificar y actualizar conjuntos
28      SI EsMejorSolucion(solucion_modelo_2) ENTONCES
29         AgregarNuevoSubconjunto(conjunto_H, solucion_modelo_2)
30      SINO
31         TERMINAR CICLO
32      FIN SI
33   FIN MIENTRAS
34
35   # PASO 6: Arreglar la solucion del paso 3 para que sea una solucion entera para
         nuestro problema.
36   modelo_3_binario <- CrearModeloBinario(nombres_archivos, conjunto_H)
37   x <- ObtenerSolucionPrimal3(modelo_3_binario)
38
39   SI SeEncontroSolucion() O SeAgotoTiempo() ENTONCES
40      RETORNAR [ObtenerConjuntosSeleccionados(x), modelo_3_binario, conjunto_H,
            nombres_archivos, tamanos_archivos, TiempoActual() - tiempo_inicio]
41   FIN SI
42
43   RETORNAR NULL
44 FIN
```

6.3 Modelo P

Este modelo es una relajación del modelo planteado en la Parte 3, considerando a x_j como una variable continua.

6.3.1 Definición de variables

n: Constante, cantidad de archivos.

m: Constante, cantidad de conjuntos H.

a_{ij}: Constante binaria, donde $a_{ij} = 1$ indica que el conjunto H_j contiene el archivo i, con $y_{ij} \in \{0, 1\}$, $\forall i \in \{1, \ldots, n\}$, $\forall j \in \{1, \ldots, m\}$.

x_j: Variable continua, donde $x_j = 1$ indica que el conjunto H_j pertenece a I, con $x_j \in [0, 1]$, $\forall j \in \{1, \ldots, m\}$.

6.3.2 Modelo

$$\texttt{Minimize} \quad \sum_{j=1}^{m} x_j$$

$$\texttt{Subject to} \quad \sum_{j=1}^{m} a_{i,j} \cdot x_j \geq 1 \quad \forall i, \text{ (todos los archivos deben estar en al menos un conjunto elegido)}$$

$$x_j \in \{0,1\}, \quad \forall j \in \{1,\ldots,m\}$$

6.4 Dual de P

Obtenemos el dual de P considerando las restricciones de P.

6.4.1 Definición de variables

n: Constante, cantidad de archivos.

m: Constante, cantidad de conjuntos H.

a_{ij}: Constante binaria, donde $a_{ij} = 1$ indica que el conjunto H_j contiene el archivo i, con $a_{ij} \in \{0,1\}$, $\forall i \in \{1,\ldots,n\}$, $\forall j \in \{1,\ldots,m\}$.

y_i: Variable dual continua asociada a la restricción de que el archivo i debe estar cubierto.

6.4.2 Modelo

$$\texttt{Maximize} \quad \sum_{i=1}^{n} y_i$$

$$\texttt{Subject to} \quad \sum_{i=1}^{n} a_{i,j} \cdot y_i \leq 1 \quad \forall j$$

$$y_i \in \{0,1\}, \quad \forall i \in \{1,\ldots,n\}$$

7 Parte 6

7.1 Descripción

En esta sección, se explican mejoras al algoritmo de la Sección 6. Para agilizar el algoritmo, investigamos en la documentación de PySCIPOpt con el objetivo de identificar los parámetros y configuraciones que nos permitan optimizar el tiempo de ejecución del código.

PySCIPOpt, al estar basado en SCIP, brinda muchas opciones para personalizar su comportamiento, como ajustes en las estrategias de búsqueda, límites en la exploración del árbol y técnicas específicas para encontrar soluciones factibles de manera más eficiente. Estas configuraciones permiten adaptar el rendimiento del solver a las características particulares del problema, maximizando su eficacia y reduciendo el tiempo computacional necesario. En nuestro caso, realizamos las siguientes mejoras al modelo:

```
# Reduce el tiempo empleado en la heurística
model.setHeuristics(SCIP_PARAMSETTING.FAST)

# Para resolver los problemas fáciles más rápidamente
model.setEmphasis(SCIP_PARAMEMPHASIS.EASYCIP)

# Máximo número de procesadores
model.setParam("parallel/maxnthreads", 16)
```

También, analizamos el setting `model.setParam("parallel/mode", 0)`, con 0 para *opportunistic* ó 1 para *deterministic*. Pero no apreciamos cambios significativos en la performance.

8 Resultados de script

En esta sección, listamos los resultados obtenidos.

Caso	**Cant**	**Mejor 1**			**Mejor 4**			**Mejor 5**			**Mejor 6**			**Cota Dual**
		Valor	Var	Tiempo	Valor	Var	Tiempo	Valor	Var	Tiempo	Valor	Var*	Tiempo	
$f(n_1) = \text{f}(17)$	17	9	306	0.0	9	212	0.0	9	22	3.19	9	0	0.01	9
f(32)	32	-	-	420	17	2922	0.0	18	30	5.05	17	0	0.01	17
$f(n_4) = \text{f}(74)$	74	-	-	420	38	159654	150.0	35	77	19.14	38	0	0.02	38
f(512)	512	-	-	420	-	-	420	279	51	22.03	279	0	0.17	279
$f(n_5) = \text{f}(1158)$	1158	-	-	420	-	-	420	595	0	159.04	595	0	0.89	595
f(2048)	2048	-	-	420	-	-	420	1021	14	420.79	1021	0	2.72	1021
$f(n_6) = \text{f}(16384)$	16384	-	-	420	-	-	420	-	-	420	8162	0	385.86	8162

Tabla 1: Resultados del Script.

Nota: Los resultados de la columna `Var6` indican 0 variables debido a que el modelo es resuelto por el presolve.

Nota 2: Los archivos del `Caso f(16384)` fueron generados duplicando un total de 8 veces los archivos del caso `f(2048)`.

9 Conclusiones

Logramos familiarizarnos con el uso de SCIP, y con PySCIPOpt. Pudimos apreciar el alto nivel de abstracción de la sintaxis de Zimpl, utilizado directamente SCIP. Si bien nos llevó más tiempo del pensado, disfrutamos mucho realizar este trabajo ya que nos ayudó a familiarizarnos con las herramientas de modelado, y pudimos relacionarlas y aplicar lo visto en la materia.

Referencias

[1] Koch, T. (2024) *Zimpl User Guide*, Berlin: ZIB.