MyO – 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

F: conjunto de nombres de cada archivo i, con $i = 1, \ldots, n$.

s_i: tamaño del archivo i en MB, con $i = 1, \ldots, n$.

n: cantidad de archivos.

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

D: conjunto de j discos de tamaño d, con $j = 1, \ldots, m$.

d: tamaño del disco en TB.

x_{ij} binaria: 1 indica si el archivo i está en el disco j, $\forall i = \{1, \ldots, n\}$, $\forall j = \{1, \ldots, m\}$, donde $x_{ij} \in \{0, 1\}$, 0 en el caso de que no esté

y_j binaria: 1 indica que se está usando el disco j.

2.2.2 Modelo

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

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:

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

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 time
import sys
from pyscipopt import Model
from pyscipopt import quicksum
from pyscipopt import SCIP_PARAMSETTING

def distribuir_archivos_1(d_t: int, F: list[str], s: list[int], time_limit=420):
    # model = crear_modelo_1(d_t, F, s, time_limit)
    model, fake_y, fake_x = resolver_modelo_binario_1(d_t, F, s, time_limit)

    try:
        x, obj_x = obtener_solucion_primal_1(model)
    except TypeError:
        x, obj_x = None, None

    sys.stderr.write(f"[Debugging] obj x {obj_x}\n")

    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")

    d = d_t * 10**6

    n = len(F)
    m = n # no se puede tener más discos que archivos

    ###########################################################################
    # 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)]
    #
    ############################################################################

    # minimize disks:
    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])

    # Configurar el límite de tiempo en el solver
    model.setParam("limits/time", time_limit)
    model.setParam("display/freq", 1)

    model.optimize()

    sys.stderr.write(f"[Debugging] [MODELO 1] Time: {model.getSolvingTime()}\n\n")
    sys.stderr.write(f"[Debugging] [MODELO 1] Cantidad sols: {model.getNSols()}\n\n")

    return model, y, x

# Crea el modelo relajado y lo devuelve optimizado
def crear_modelo_1(d_t: int, F: list[str], s: list[int], time_limit=420):
    sys.stderr.write(f"[Debugging] [MODELO 1] Inicio\n\n")
    model = Model("model_part_1")

    d = d_t * 10**6

    n = len(F)
    m = n # no se puede tener más discos que archivos

    ############################################################################
    # CONTINUOUS
```

```
82    # x_{i, j} = 1 si se elige el archivo i para el disco j, 0 si no
83    #
84    x = {}
85    for i in range(n):
86       for j in range(m):
87          x[i, j] = model.addVar(name = f"x_{i}_{j}", lb=0, ub=1, vtype="CONTINUOUS")
88
89    # y_{j} = 1 si se elige el disco j, 0 si no
90    y = [model.addVar(f"y_{j}", lb=0, ub=1, vtype="CONTINUOUS") for j in range(m)]
91    #
92    ############################################################################
93
94    # minimize disks:
95    model.setObjective(quicksum(y), sense="minimize")
96
97    # Que los archivos se elijan solo para un disco
98    for i in range(n):
99       model.addCons(quicksum(x[i, j] for j in range(m)) == 1)
100
101   # Que no se pasen de capacidad los discos
102   for j in range(m):
103      model.addCons(quicksum(x[i, j] * s[i] for i in range(n)) <= d * y[j])
104
105   # Configurar el límite de tiempo en el solver
106   model.setParam("limits/time", time_limit)
107   model.setParam("display/freq", 1)
108
109   # Desactivación temporal de presolve
110   model.setPresolve(SCIP_PARAMSETTING.OFF)
111
112   ########################################################
113   # Esto es la clave para que obj(dual) = obj(primal)
114   # según documentación de pyscipopt
115   #
116   model.setHeuristics(SCIP_PARAMSETTING.OFF)
117   model.disablePropagation()
118   #
119   ########################################################
120
121   model.optimize()
122
123   sys.stderr.write(f"[Debugging] [MODELO 1] Time: {model.getSolvingTime()}\n\n")
124   sys.stderr.write(f"[Debugging] [MODELO 1] Cantidad sols: {model.getNSols()}\n\n")
```

```
125
126     return model
127
128 def obtener_solucion_primal_1(model):
129     sol = model.getBestSol()
130     status = model.getStatus()
131
132     if sol is not None and status in ["optimal", "feasible"]:
133         sys.stderr.write(f"[Debugging] {model.getStatus()}: {model.getBestSol()}\n\n")
134
135         # El nombre de variable x, acá es missleading, es x porque es el primal.
136         # Esta línea devuelve **todas** las variables del modelo, las $x$: archivo seleccionado y las
137         x = [v.getLPSol() for v in model.getVars(False)]
138
139         sys.stderr.write(f"[Debugging]: todas las variables del primal model_part_1 {x}\n\n")
140         return x, model.getObjVal()
141     else:
142         return None
143
144 def obtener_solucion_dual_1(model):
145     # No debería ser necesario si el modelo ya viene optimizado con el presolving off
146     # Aseguarse de apagar el presolving
147     # model.setPresolve(SCIP_PARAMSETTING.OFF)
148
149     # La longitud de y coincide con la cantidad de archivos del input
150     y = [model.getDualSolVal(c) for c in model.getConss(False)]
151
152     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
import os
import random

def generar_archivos(cant_archivos):
    archivos = {}
    contador = 1
    for i in range(cant_archivos):
        archivos["archivo" + str(contador)] = random.randint(1000000, 10000000)
        contador += 1
    return archivos


def generar_configuracion(nombre_archivo):
    capacidad_discos = random.randint(1, 300)
    cant_archivos = random.randint(400, 550)
    archivos = generar_archivos(cant_archivos)

    # ruta_in = os.path.join(os.path.dirname(__file__), "IN", nombre_archivo)
    ruta_in = os.path.dirname(__file__) + "/IN/" + nombre_archivo
    with open(ruta_in, "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")

        f.write(f"\n# Archivos: archivo_id, tamaño (MB) \n")
        for archivo in archivos:
            f.write(archivo + " " + str(archivos[archivo]) + "\n")


def leer_configuracion(nombre_archivo):
    capacidad_disco = 0
    nombres_archivos = []
    tamaños_archivos = []

    # ruta_in = os.path.join(os.path.dirname(__file__), "IN", nombre_archivo)
    ruta_in = os.path.dirname(__file__) + "/IN/" + nombre_archivo
    with open(ruta_in, "r") as f:
        lineas = f.readlines()
```

```
    capacidad_disco = int(lineas[1].strip())

    for i in range(7, len(lineas)):
        if lineas[i].strip():
            archivo = lineas[i].split()
            nombres_archivos.append(archivo[0])
            tamaños_archivos.append(int(archivo[1]))
    return capacidad_disco, nombres_archivos, tamaños_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 sys
from configuracion_1 import *
from generador_output_1 import *
from model_part_1 import *

if len(sys.argv) != 2:
    print(f"Uso: {sys.argv[0]} nombre_archivo")
    sys.exit(1)

archivo = sys.argv[1]
print(f"Utilizando {archivo}\n")

# generar_configuracion(archivo)

capacidad_disco, nombres_archivos, tamaños_archivos = leer_configuracion(f"{archivo}")

solucion = distribuir_archivos_1(capacidad_disco, nombres_archivos, tamaños_archivos, 420)

if solucion is not None:
    generar_output(f"{archivo[:-3]}.out", solucion)
else:
    generar_output_fallido(f"{archivo[:-3]}.out")
```

Utilizamos las siguientes funciones para generar el output.

```python
import os

def generar_output(nombre_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()))

   # La cantidad de discos disponibles es a lo sumo la cantidad de archivos
   number_of_disks = cant_archivos

   ruta_out = os.path.join(os.path.dirname(__file__), ".", "OUT",
                     nombre_archivo)
   with open(ruta_out, "w") as f:
      f.write(
         f"Para la configuración del archivo, {cant_discos} discos son suficientes.\n"
      )
      for j in range(number_of_disks):
         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")


def generar_output_fallido(nombre_archivo):
   ruta_out = os.path.join(os.path.dirname(__file__), ".", "OUT",
```

```
                    nombre_archivo)
    with open(ruta_out, "w") as f:
        f.write(
            f"No se ha encontrado solucion para la configuracion del 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

F: conjunto de nombres de i archivos, con $i = 1, \ldots, n$.

s_i constante: tamaño del archivo i en MB, con $i = 1, \ldots, n$.

I_i constante: indicador de importancia para el archivo i, con $i = 1, \ldots, n$.

n constante: cantidad de archivos.

d constante: tamaño del disco en TB.

x_i binaria: 1 indica si el archivo i está en el disco, $\forall i = \{1, \ldots, n\}$, donde $x_i \in \{0, 1\}$, 0 en el caso de que no esté

3.2.2 Modelo

$$\texttt{Maximize} \quad \sum_{i=1}^{n} I_i \cdot x_i$$

$$\texttt{Subject to} \quad \sum_{i=1}^{n} s_i x_i \leq d, \quad \text{(archivos que entran en el disco)}$$

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

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

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

$$n \geq 0$$

$$d \geq 0$$

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

# Model segunada parte
def distribuir_archivos_2(d_t: int, F: list[str], s: list[int], I: list[float]):
    model, fake_x = resolver_modelo_binario_2(d_t, F, s, I, 420)
    try:
        x, obj_x = obtener_solucion_primal_2(model)
    except TypeError:
        x, obj_x = None, None

    # sys.stderr.write(f"[Debugging] obj x {obj_x}\n")

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

    if solution is not None and status in ["optimal", "feasible"]:
        sys.stderr.write(f"[Debuggin] {status}: {solution}\n\n")
        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)

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

    # maximize importance:
```

```
    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.setParam("display/freq", 1)

    model.optimize()

    sys.stderr.write(f"[Debugging] model_part_2 Time: {model.getSolvingTime()}\n\n")
    sys.stderr.write(f"[Debugging] model_part_2 Cantidad sols: {model.getNSols()}\n\n")

    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)

    ############################################################################
    # CONTINUOUS
    # 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)]
    ############################################################################

    # maximize importance:
    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.setParam("display/freq", 1)

    # Desactivación temporal de presolve
    model.setPresolve(SCIP_PARAMSETTING.OFF)

```

```
    ########################################################
    # Esto es la clave para que obj(dual) = obj(primal)
    # según documentación de pyscipopt
    #
    model.disablePropagation()
    model.setHeuristics(SCIP_PARAMSETTING.OFF)
    #
    ########################################################

    model.optimize()

    sys.stderr.write(f"[Debugging] model_part_2 Time: {model.getSolvingTime()}\n\n")
    sys.stderr.write(f"[Debugging] model_part_2 Cantidad sols: {model.getNSols()}\n\n")

    return model

def obtener_solucion_primal_2(model):
    sol = model.getBestSol()
    status = model.getStatus()

    if sol is not None and status in ["optimal", "feasible"]:
        sys.stderr.write(f"[Debugging] {status}: {sol}\n\n")

        x = [v.getLPSol() for v in model.getVars(False)]

        sys.stderr.write(f"[Debugging]: {x}\n\n")
        return x, model.getObjVal()
    else:
        return None

def obtener_solucion_dual_2(model):
    # No debería ser necesario si el modelo ya viene optimizado con el presolving off
    # Aseguarse de apagar el presolving
    # model.setPresolve(SCIP_PARAMSETTING.OFF)

    # La longitud de y coincide con la cantidad de archivos del input
    y = [model.getDualSolVal(c) for c in model.getConss(False)]

    return y, quicksum(y)
```

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

```python
import os
import random


def generar_archivos(cant_archivos):
    archivos = {}
    contador = 1
    for i in range(cant_archivos):
        # {nombreArchivo: [tamaño, importancia] }
        archivos["archivo" + str(contador)] = [
            random.randint(1000000, 10000000),
            random.randint(1, 10)
        ]
        contador += 1
    return archivos


def generar_configuracion(nombre_archivo):
    capacidad_discos = random.randint(1, 100)
    cant_archivos = random.randint(1, 50)
    archivos = generar_archivos(cant_archivos)

    ruta_in = os.path.join(os.path.dirname(__file__), ".", "IN",
                           nombre_archivo)
    with open(ruta_in, "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")

        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 leer_configuracion(nombre_archivo):
    capacidad_disco = 0
    nombres_archivos = []
    tamaños_archivos = []
```

```
43    importancias_archivos = []
44
45    ruta_in = os.path.join(os.path.dirname(__file__), ".", "IN",
46                           nombre_archivo)
47    with open(ruta_in, "r") as f:
48
49       lineas = f.readlines()
50       capacidad_disco = int(lineas[1].strip())
51       for i in range(7, len(lineas)):
52          if lineas[i].strip():
53             archivo = lineas[i].split()
54             nombres_archivos.append(archivo[0])
55             tamaños_archivos.append(int(archivo[1]))
56             importancias_archivos.append(int(archivo[2]))
57    return capacidad_disco, nombres_archivos, tamaños_archivos, importancias_archivos
```

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

```
#!/usr/bin/env python3

import sys
from configuracion_2 import *
from generador_output_2 import *
from model_part_2 import *

if len(sys.argv) != 2:
   print(f"Uso: {sys.argv[0]} nombre_archivo")
   sys.exit(1)

archivo = sys.argv[1]
print(f"Utilizando {archivo}\n")

generar_configuracion(archivo)
capacidad_disco, nombres_archivos, tamaños_archivos, importancia_archivos = leer_configuracion(f"{a
solucion = distribuir_archivos_2(capacidad_disco, nombres_archivos, tamaños_archivos, importancia_a

if solucion is not None:
   generar_output(f"{archivo[:-3]}.out", solucion)
else:
   generar_output_fallido(f"{archivo[:-3]}.out")
```

Luego, el output se almacena de la siguiente manera.

```python
import os

# solucion = F, model, x, I, s
def generar_output(nombre_archivo, solucion):
    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

    ruta_out = os.path.join(os.path.dirname(__file__), ".", "OUT",
                        nombre_archivo)
    with open(ruta_out, "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: # se eligio el archivo
                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}."
        )

def generar_output_fallido(nombre_archivo):
    with open(nombre_archivo, "w") as f:
        f.write(
            f"No se ha encontrado solucion para la configuracion del archivo.\n"
        )
```

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: cantidad de archivos.

m: cantidad de conjuntos H.

x_j binaria: 1 indica que el conjunto H_j pertenece a I.

y_{ij} **constante** binaria: 1 indica que el conjunto H_j contiene el archivo i.

4.2.2 Modelo

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

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

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

$$y_{i,j} \in \{0,1\}, \quad \forall i \in \{1,\ldots,n\}, \quad \forall j \in \{1,\ldots,m\}$$

$$n \geq 0,$$

$$m \geq 0,$$

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

F: conjunto de nombres de cada archivo i, con $i \in \{1, \ldots, n\}$.

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

n: cantidad de archivos.

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

D: conjunto de j discos de tamaño d, con $j \in \{1, \ldots, m\}$.

S: conjunto de los tamaños de s_i, con $i \in \{1, \ldots, n\}$.

d: tamaño del disco en TB.

t: constante, máxima cantidad de tamaños que entran en un disco.

x_{ij}: binaria, toma el valor 1 si el archivo i está en el disco j, y 0 en caso contrario, para $\forall i \in \{1, \ldots, n\}$ y $\forall j \in \{1, \ldots, m\}$, donde $x_{ij} \in \{0, 1\}$.

y_j: binaria, toma el valor 1 si se está usando el disco j, y 0 en caso contrario, para $\forall j \in \{1, \ldots, m\}$, donde $y_j \in \{0, 1\}$.

z_{sj}: binaria, toma el valor 1 si el tamaño s está en el disco j al menos una vez, y 0 en caso contrario, para $\forall s$ y $\forall j \in \{1, \ldots, m\}$, donde $z_{sj} \in \{0, 1\}$.

M_{si}: constante, toma la cantidad de archivos de tamaño s_i que entran en un disco de tamaño d. Nótese que esto es equivalente a $int(toMB(d)/s)$.

5.2.2 Modelo

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

$$\texttt{Subject to} \quad \sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in \{1, \ldots, n\}, \text{ (el archivo } i \text{ debe estar en un solo disco)}$$

$$\sum_{i=1}^{n} s_i x_{ij} \leq d \cdot y_j \quad \forall j \in \{1, \ldots, m\} \text{(los archivos que entran en el disco } j\text{)}$$

$$z_{sj} \cdot M_{si} \leq \sum_{i=1}^{n} x_{ij} \quad \text{si } S_i = s, \quad \forall s \in S, \forall j \in \{1, \ldots, m\}, \forall i \in \{1, \ldots, n\} \text{(marcamos los tamaños c}$$

$$\sum_{s \in S} z_{sj} \leq t, \quad \forall j \in \{1, \ldots, m\} \text{(en cada disco no entran mas de t tamaños)}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, \ldots, n\}, \ \forall j \in \{1, \ldots, m\}$$

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

$$z_{sj} \in \{0, 1\}, \quad \forall s \in S, \ \forall j \in \{1, \ldots, m\}$$

$$n \geq 0, n \in \mathbb{Z}$$

$$s_i \geq 0, \forall i \in \{1, \ldots, n\}$$

$$m \geq 0, m \leq n, m \in \mathbb{Z}$$

$$d \geq 0$$

$$t \geq 0, t \in \mathbb{Z}$$

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)
   capacidad_disco, nombres_archivos, tamanos_archivos <- leer_configuracion(archivo)

   // PASO 1: Generar los conjuntos iniciales
   conjuntos <- generar_conjuntos(capacidad_disco * 10^6, nombres_archivos, tamanos_archivos)

   duracion <- threshold * 60
   tiempo_inicio <- tiempo_actual()

   MIENTRAS VERDADERO
      // PASO 2 y 3: Resolver el modelo y obtener soluciones primal y dual
      modelo <- crear_modelo_3(nombres_archivos, conjuntos)

      x, obj_x <- obtener_solucion_primal_3(modelo)
      y, obj_y <- obtener_solucion_dual_3(modelo)

      optimo <- es_optimo(modelo, x)

      // PASO 4: Distribuir archivos y generar solucion
      distribucion <- distribuir_archivos_2(capacidad_disco, nombres_archivos, tamanos_archivos, y)
      solucion_modelo_2 <- generar_output_modelo_2(distribucion)

      // PASO 5: Verificar y actualizar conjuntos
      SI suma(solucion_modelo_2[1]) > 1 Y tiempo_actual() - tiempo_inicio <= duracion ENTONCES
         AGREGAR conjunto(solucion_modelo_2[0]) a conjuntos
      SINO
         IMPRIMIR suma(solucion_modelo_2[1])
         SALIR DEL BUCLE
      FIN SI
   FIN MIENTRAS

   x_estrella_int <- obtener_solucion_entera(modelo, x)
   RETORNAR modelo
FIN
```

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 parámetros y configuraciones que permitan optimizar el tiempo de ejecución del código. PySCIPOpt, al estar basado en el solucionador 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 solucionador a las características particulares del problema, maximizando su eficacia y reduciendo el tiempo computacional necesario. En nuestro caso, realizamos las siguientes mejoras al modelo:

8 Resultados de script

caso	cant	cota dual	mejor1	var1	tiempo1	mejor4	var4	tiempo4	mejor5	var5	tiempo5	mejor6
f0015.in	15	8.0	8.0	240	0.0	8.0	240	0.0	7.666666666666667	43	0.0	0
f0018.in	18	10.0	8.0	240	0.0	9.999999544151857	342	98.0	9.166666666666666	69	0.0	0
f0032.in	32	16.5	8.0	240	0.0	9.999999544151857	342	98.0	16.5	285	0.0	0

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.

Este trabajo 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.