

Modelado y Optimización – Trabajo Práctico

Lucía Soria

Ximena Ebertz

2^{do} Semestre 2024

Contenidos

1	Introducción	3
2	Parte 1	4
2.1	Descripción	4
2.2	Modelo Lineal	4
2.2.1	Definición de variables	4
2.2.2	Modelo	5
2.2.3	Restricción: que no se elijan discos vacíos	5
2.2.4	Modelo Zimpl	7
2.2.5	Modelo PySCIPOpt	8
3	Parte 2	14
3.1	Descripción	14
3.2	Modelo Lineal	14
3.2.1	Definición de variables	14
3.2.2	Modelo	14
3.2.3	Modelo Zimpl	15
3.2.4	Modelo PySCIPOpt	16
4	Parte 3	23
4.1	Descripción	23
4.2	Modelo Lineal	23
4.2.1	Definición de variables	23
4.2.2	Modelo	23
5	Parte 4	24
5.1	Descripción	24
5.2	Modelo Lineal	24
5.2.1	Definición de variables	24
5.2.2	Modelo	24

6	Parte 5	25
6.1	Descripción	25
6.2	Pseudocódigo de alto nivel	25
6.3	Modelo P	26
6.3.1	Definición de variables	26
6.3.2	Modelo	27
6.4	Dual de P	27
6.4.1	Definición de variables	27
6.4.2	Modelo	27
7	Parte 6	28
7.1	Descripción	28
8	Resultados de script	28
9	Conclusiones	30

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 **PySCIP0pt**. 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, $\forall i \in \{1, \dots, n\}$.

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 , $\forall i \in \{1, \dots, n\}$, $\forall j \in \{1, \dots, m\}$, con $x_{ij} \in \{0, 1\}$.

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

2.2.2 Modelo

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

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:

$$\sum_{j=1}^2 x_{ij} = 1 \quad \forall i \in \{1, 2\} \quad (1)$$

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

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:

$$\sum_{j=1}^2 x_{ij} = 1 \quad \forall i \in \{1, 2\} \quad (3)$$

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

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.

```
1 param input := "IN/a_1.in"; # Path del archivo de input
2
3 # Tamaño del disco en terabytes
4 param d_t := read input as "1n" use 1 comment "#";
5 param d := d_t * 10**6;
6
7 # Conjunto de archivos f_{i} \forall i \in \{1, \dots, n\}
8 set F := { read input as "<1s>" skip 2 comment "#" };
9
10 param n := card(F); # Cantidad de archivos
11 param m := n; # Cantidad de discos, a lo sumo, un disco por archivo
12
13 # Conjunto de discos
14 set D := { 1 .. m };
15
16 # Tamaños de los archivos f_{i} \forall i \in \{1, \dots, n\}
17 param s[F] := read input as "<1s> 2n" skip 2 comment "#";
18
19 # 1 si el archivo $i$ está en el disco $j$, 0 en caso contrario
20 var x[<i, j> in F * D] binary;
21
22 # 1 si se usa el disco $j$, 0 en caso contrario
23 var y[D] binary;
24
25 minimize disks: sum<j> in D: y[j];
26
27 # Cada archivo debe estar en un solo disco
28 subto c1:
29     forall <i> in F:
30         sum<j> in D: x[i, j] == 1;
31
32 # Los archivos $i$ que entran en el disco $j$
33 subto c2:
34     forall <j> in D:
35         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.

```
1 import sys
2 from pyscipopt import Model
3 from pyscipopt import quicksum
4 from pyscipopt import SCIP_PARAMSETTING
5 from pyscipopt import *
6
7 #####
8     # d_t: capacidad del disco en TB,
9     # F: nombres de los archivos,
10    # s: tamaños de los archivos,
11    # time_limit: threshold en segundos
12 #####
13 def distribuir_archivos_1(d_t: int, F: list[str], s: list[int], time_limit=420):
14     model, fake_y, fake_x = resolver_modelo_binario_1(d_t, F, s, time_limit)
15
16     solution = model.getBestSol()
17     status = model.getStatus()
18
19     if solution is not None and status in ["optimal", "feasible"]:
20         return [F, model, fake_y, fake_x, s]
21     else:
22         return None
23
24 def resolver_modelo_binario_1(d_t: int, F: list[str], s: list[int], time_limit=420):
25     model = Model("model_part_1")
26     model.setParam("limits/time", time_limit)
27
28     d = d_t * 10**6
29
30     n = len(F)
31     m = n
32
33     #####
34     # BINARY
35     # x_{i, j} = 1 si se elige el archivo i para el disco j, 0 si no
36     #
37     x = {}
38     for i in range(n):
```



```

39     for j in range(m):
40         x[i, j] = model.addVar(name=f"x_{i}_{j}", vtype="BINARY")
41
42     # y_{j} = 1 si se elige el disco j, 0 si no
43     y = [model.addVar(f"y_{j}", vtype="BINARY") for j in range(m)]
44     #
45     #####
46
47     # Minimizar la cantidad de discos
48     model.setObjective(quicksum(y), sense="minimize")
49
50     # Que los archivos se elijan solo para un disco
51     for i in range(n):
52         model.addCons(quicksum(x[i, j] for j in range(m)) == 1)
53
54     # Que no se pasen de capacidad los discos
55     for j in range(m):
56         model.addCons(quicksum(x[i, j] * s[i] for i in range(n)) <= d * y[j])
57
58     model.setParam("display/freq", 1)
59     model.optimize()
60
61     return model, y, x
62
63 def obtener_solucion_primal_1(model):
64     sol = model.getBestSol()
65     status = model.getStatus()
66
67     if sol is not None and status in ["optimal", "feasible"]:
68         x = [v.getLPSol() for v in model.getVars(False)]
69         return x, model.getObjVal()
70     else:
71         return None
72
73 def obtener_solucion_dual_1(model):
74     y = [model.getDualSolVal(c) for c in model.getConss(False)]
75     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.

```
1 def generar_input_1(ruta_archivo: str):
2     generar_input(ruta_archivo, False)

1 def generar_input(ruta_archivo: str, importancia: bool):
2     capacidad_discos = random.randint(1, 300)
3     cant_archivos = random.randint(400, 550)
4     archivos = generar_archivos(cant_archivos, importancia)
5
6     with open(ruta_archivo, "w") as f:
7
8         f.write(f"# Capacidad de discos en TB (= 1.000.000 MB)\n")
9         f.write(str(capacidad_discos) + "\n")
10
11        f.write(f"\n# Cantidad de archivos para backup\n")
12        f.write(str(cant_archivos) + "\n")
13
14        if not importancia:
15            f.write(f"\n# Archivos: archivo_id, tamaño (MB) \n")
16            for archivo in archivos:
17                f.write(archivo + " " + str(archivos[archivo]) + "\n")
18        else:
19            f.write(f"\n# Archivos: archivo_id, tamaño (MB), importancia\n")
20            for archivo in archivos:
21                f.write(archivo + " " + str(archivos[archivo][0]) + " " + str(archivos[
22                    archivo][1]) + "\n")
23
24 def generar_archivos(cant_archivos: int, importancia: bool):
25     archivos = {}
26     contador = 1
27     for i in range(cant_archivos):
28         if not importancia:
29             # {nombreArchivo: tamaño}
30             archivos["archivo" + str(contador)] = random.randint(1000000, 10000000)
31         else:
32             # {nombreArchivo: [tamaño, importancia] }
33             archivos["archivo" + str(contador)] = [
34                 random.randint(1000000, 10000000),
35                 random.randint(1, 10)
36             ]
37         contador += 1
38     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.

```
1 #!/usr/bin/env python3
2 import os
3 import sys
4
5 sys.path.insert(0, "../utils")
6
7 import inputs
8 import configs
9 import outputs
10 import model_part_1
11
12 if len(sys.argv) != 3:
13     print(f'Uso: {sys.argv[0]} OPTION archivo')
14     print(f'OPTIONS: -g | -u | -c')
15     print(f'-g generar archivo')
16     print(f'-u usar archivo ya generado')
17     print(f'-c usar configuración')
18     sys.exit(1)
19
20 archivo = sys.argv[2]
21 archivos = []
22 threshold = 7
23 out_path = 'OUT'
24
25 if sys.argv[1] == '-g':
26     print(f'Generando {archivo}\n')
27     inputs.generar_input_1(os.path.dirname(__file__) + '/IN/' + archivo)
28     archivos.append(archivo)
29
30 elif sys.argv[1] == '-u':
31     print(f'Usando {archivo}\n')
32     archivos.append(archivo)
33
34 elif sys.argv[1] == '-c':
35     print(f'Leyendo configuración {archivo}\n')
36     configuraciones = configs.leer_configuracion(os.path.join(os.path.dirname(
37         __file__), archivo))
38     out_path = configuraciones.get('outPath')[:-1]
39     threshold = int(configuraciones.get('threshold', 0))
40     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.

```
1 def generar_output_1(ruta_archivo, solucion):
2     # solucion = [F, model, y, x, s]
3     F = solucion[0]
4     model = solucion[1]
5     y = solucion[2]
6     x = solucion[3]
7     s = solucion[4]
8
9     cant_archivos = len(F)
10    cant_discos = round(float(model.getObjVal()))
11    max_discos = cant_archivos
12
13    with open(ruta_archivo, "w") as f:
14        f.write(f"Para la configuracion del archivo, {cant_discos} discos son
15            suficientes.\n")
16        for j in range(max_discos):
17            if model.getVal(y[j]) == 0:
18                continue
19
20            archivos_en_disco = []
21            espacio_ocupado = 0
22
23            for i in range(cant_archivos):
24                if model.getVal(x[i, j]) == 0:
25                    continue
26                archivos_en_disco.append(f"{F[i]} {s[i]}")
27                espacio_ocupado = espacio_ocupado + s[i]
28
29            f.write(f"\nDisco {j+1}: {espacio_ocupado} MB\n")
30
31        for archivo in archivos_en_disco:
32            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 $i \in \{1, \dots, n\}$.

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

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 $x_i \in \{0, 1\}$, $\forall i \in \{1, \dots, n\}$.

3.2.2 Modelo

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

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.

```
1 param input := "IN/b_1.in"; # Path del archivo de input
2
3 # Tamaño del disco en terabytes
4 param d_t := read input as "1n" use 1 comment "#";
5 param d := d_t * 10**6;
6
7 # Conjunto de archivos f_{i} \forall i \in \{1, \dots, n\}
8 set F := { read input as "<1s>" skip 2 comment "#" };
9
10 param n := card(F); # Cantidad de archivos
11
12 # Tamaños de los archivos f_{i} \forall i \in \{1, \dots, n\}
13 param s[F] := read input as "<1s> 2n" skip 2 comment "#";
14
15 # Importancias de los archivos
16 param importance[F] := read input as "<1s> 3n" skip 2 comment "#";
17
18 # 1 si el archivo $i$ está en el disco, 0 en caso contrario
19 var x[F] binary;
20
21 maximize importances: sum<i> in F: importance[i] * x[i];
22
23 # Los archivos $i$ que entran en el disco
24 subto c1:
25     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.

```
1 import sys
2 from pycscipopt import Model
3 from pycscipopt import quicksum
4 from pycscipopt import SCIP_PARAMSETTING
5 from pycscipopt import *
6
7 #####
8     # d_t: capacidad del disco en TB,
9     # F: nombres de los archivos,
10    # s: tamaños de los archivos,
11    # I: importancia de los archivos,
12    # time_limit: threshold en segundos
13 #####
14 def distribuir_archivos_2(d_t: int, F: list[str], s: list[int], I: list[float],
15    time_limit=420):
16     model, fake_x = resolver_modelo_binario_2(d_t, F, s, I, time_limit)
17
18     solution = model.getBestSol()
19     status = model.getStatus()
20
21     if solution is not None and status in ["optimal", "feasible"]:
22         return [F, model, fake_x, I, s]
23     else:
24         return None
25
26 def resolver_modelo_binario_2(d_t: int, F: list[str], s: list[int], I: list[float],
27    time_limit=420):
28     model = Model("model_part_2")
29     d = d_t * 10**6
30     n = len(F)
31
32     #####
33     # BINARIA
34     # x_{i} = 1 si se elige el archivo i, 0 si no
35     x = [model.addVar(f"x_{i}", vtype="BINARY") for i in range(n)]
36     #####
37
38     # Maximizar importancia:
```



```

37 model.setObjective(quicksum(x[i] * I[i] for i in range(n)),
38                     sense="maximize")
39
40 # Los archivos elegidos deben entrar en el disco
41 model.addCons(quicksum(x[i] * s[i] for i in range(n)) <= d)
42
43 # Configurar el límite de tiempo en el solver
44 model.setParam("limits/time", time_limit)
45 model.optimize()
46
47 return model, x
48
49 # Crea el modelo relajado y lo devuelve optimizado
50 def crear_modelo_2(d_t: int, F: list[str], s: list[int], I: list[float], time_limit
51                   =420):
52     model = Model("model_part_2")
53
54     d = d_t * 10**6
55     n = len(F)
56
57     #####
58     # CONTINUA
59     # x_{i} = 1 si se elige el archivo i, 0 si no
60     x = [
61         model.addVar(f"y_{i}", lb=0, ub=1, vtype="CONTINUOUS")
62         for i in range(n)
63     ]
64     #####
65
66     # Maximizar importancia:
67     model.setObjective(quicksum(x[i] * I[i] for i in range(n)),
68                       sense="maximize")
69
70     # Loss archivos elegidos deben entrar en el disco
71     model.addCons(quicksum(x[i] * s[i] for i in range(n)) <= d)
72
73     # Configurar el límite de tiempo en el solver
74     model.setParam("limits/time", time_limit)
75
76     model.optimize()
77     return model
78
79 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.

```
1 def generar_input_2(ruta_archivo: str):
2     generar_input(ruta_archivo, True)

1 def generar_input(ruta_archivo: str, importancia: bool):
2     capacidad_discos = random.randint(1, 300)
3     cant_archivos = random.randint(400, 550)
4     archivos = generar_archivos(cant_archivos, importancia)
5
6     with open(ruta_archivo, "w") as f:
7
8         f.write(f"# Capacidad de discos en TB (= 1.000.000 MB)\n")
9         f.write(str(capacidad_discos) + "\n")
10
11        f.write(f"\n# Cantidad de archivos para backup\n")
12        f.write(str(cant_archivos) + "\n")
13
14        if not importancia:
15            f.write(f"\n# Archivos: archivo_id, tamaño (MB) \n")
16            for archivo in archivos:
17                f.write(archivo + " " + str(archivos[archivo]) + "\n")
18        else:
19            f.write(f"\n# Archivos: archivo_id, tamaño (MB), importancia\n")
20            for archivo in archivos:
21                f.write(archivo + " " + str(archivos[archivo][0]) + " " + str(archivos[
22                    archivo][1]) + "\n")
23
24 def generar_archivos(cant_archivos: int, importancia: bool):
25     archivos = {}
26     contador = 1
27     for i in range(cant_archivos):
28         if not importancia:
29             # {nombreArchivo: tamaño}
30             archivos["archivo" + str(contador)] = random.randint(1000000, 10000000)
31         else:
32             # {nombreArchivo: [tamaño, importancia] }
33             archivos["archivo" + str(contador)] = [
34                 random.randint(1000000, 10000000),
35                 random.randint(1, 10)
36             ]
37         contador += 1
38     return archivos
```

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

```
1 #!/usr/bin/env python3
2 import os
3 import sys
4
5 sys.path.insert(0, "../utils")
6
7 import inputs
8 import configs
9 import outputs
10 import model_part_2
11
12 if len(sys.argv) != 3:
13     print(f'Uso: {sys.argv[0]} OPTION archivo')
14     print(f'OPTIONS: -g | -u | -c')
15     print(f'-g generar archivo')
16     print(f'-u usar archivo ya generado')
17     print(f'-c usar configuración')
18     sys.exit(1)
19
20 archivo = sys.argv[2]
21 archivos = []
22 threshold = 7
23 out_path = 'OUT'
24
25 if sys.argv[1] == '-g':
26     print(f'Generando {archivo}\n')
27     inputs.generar_input_2(os.path.dirname(__file__) + '/IN/' + archivo)
28     archivos.append(archivo)
29
30 elif sys.argv[1] == '-u':
31     print(f'Usando {archivo}\n')
32     archivos.append(archivo)
33
34 elif sys.argv[1] == '-c':
35     print(f'Leyendo configuración {archivo}\n')
36     configuraciones = configs.leer_configuracion(os.path.join(os.path.dirname(
37         __file__), archivo))
38     out_path = configuraciones.get('outPath')[:-1]
39     threshold = int(configuraciones.get('threshold', 0))
40     archivos = [f for f in os.listdir(configuraciones.get('inPath'))]
```

```

41
42
43 for archivo in archivos:
44     capacidad_disco, nombres_archivos, tamaños_archivos, importancia_archivos =
         inputs.leer_input_2(os.path.dirname(__file__) + '/IN/' + archivo)
45     solucion = model_part_2.distribuir_archivos_2(capacidad_disco, nombres_archivos,
         tamaños_archivos, importancia_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_2(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}')

```

Luego, el output se almacena de la siguiente manera.

```
1 def generar_output_2(nombre_archivo, solucion):
2     # solución = [F, model, fake_x, I, s]
3     F = solucion[0]
4     model = solucion[1]
5     x = solucion[2]
6     I = solucion[3]
7     s = solucion[4]
8
9     cant_archivos = len(F)
10    cant_archivos_elegidos = sum(1 for i in range(cant_archivos) if model.getVal(x[i]
11                                     ]) > 0.5)
12    archivos_elegidos = []
13    importancia_total = 0
14
15    with open(nombre_archivo, "w") as f:
16        f.write(f"Para la configuracion del archivo, se han elegido {
17                cant_archivos_elegidos} archivos.\n")
18
19        for i in range(cant_archivos):
20            if model.getVal(x[i]) > 0.5:
21                archivos_elegidos.append(f"{F[i]} {s[i]} {I[i]}")
22                importancia_total += I[i]
23
24        for archivo in archivos_elegidos:
25            f.write(archivo + "\n")
26
27    f.write(f"\nLa suma de sus indicadores de importancia da {importancia_total}."
28           )
```

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 $y_{ij} \in \{0, 1\}$, $\forall i \in \{1, \dots, n\}$, $\forall j \in \{1, \dots, m\}$.

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

4.2.2 Modelo

$$\begin{aligned} & \text{Minimize} && \sum_{j=1}^m x_j \\ & \text{Subject to} && \sum_{j=1}^m y_{i,j} \cdot x_j \geq 1 \quad \forall i, \text{ (todos los archivos deben estar en al menos un conjunto elegido)} \\ & && x_j \in \{0, 1\}, \quad \forall j \in \{1, \dots, m\} \end{aligned}$$

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 $k \in \{1, \dots, t\}$.

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

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

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

5.2.2 Modelo

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

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

```
1 INICIO obtener_conjuntos(archivo, threshold = infinito)
2   # Leer los datos del archivo de entrada
3   capacidad_disco, nombres_archivos, tamanos_archivos <- LeerInput5(ruta_archivo)
4   tiempo_inicio <- TiempoActual()
5
6   // PASO 1: Generar los conjuntos iniciales
7   conjunto_H <- GenerarSubconjuntos(capacidad_disco * 10^6, nombres_archivos,
8     tamanos_archivos)
9   SI conjunto_H ES NULL ENTONCES
10     RETORNAR NULL
11   FIN SI
12
13   MIENTRAS HAYA TIEMPO HACER
14     // PASO 2 y 3: Resolver el modelo y obtener soluciones primal y dual
15     modelo_P <- CrearModelo3(nombres_archivos, conjunto_H, threshold - (
16       TiempoActual() - tiempo_inicio))
17     x <- ObtenerSolucionPrimal3(modelo_P)
18     SI x ES NULL ENTONCES
19       TERMINAR CICLO
20     FIN SI
21     y <- ObtenerSolucionDual3(modelo_P)
22
23     // PASO 4: Distribuir archivos y generar solucion
24     solucion_modelo_2 <- ObtenerSolucion2(DistribuirArchivos2(capacidad_disco,
25       nombres_archivos, tamanos_archivos, y, threshold - (TiempoActual() -
26       tiempo_inicio)))
27     SI solucion_modelo_2 ES NULL O NO HAY TIEMPO ENTONCES
28       TERMINAR CICLO
29     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() 0 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 $y_{ij} \in \{0, 1\}$, $\forall i \in \{1, \dots, n\}$, $\forall j \in \{1, \dots, m\}$.

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

6.3.2 Modelo

$$\begin{aligned}
& \text{Minimize} && \sum_{j=1}^m x_j \\
& \text{Subject to} && \sum_{j=1}^m a_{i,j} \cdot x_j \geq 1 \quad \forall i, \text{ (todos los archivos deben estar en al menos un conjunto elegido)} \\
& && x_j \in \{0, 1\}, \quad \forall j \in \{1, \dots, m\}
\end{aligned}$$

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 $a_{ij} \in \{0, 1\}$, $\forall i \in \{1, \dots, n\}$, $\forall j \in \{1, \dots, m\}$.

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

6.4.2 Modelo

$$\begin{aligned}
& \text{Maximize} && \sum_{i=1}^n y_i \\
& \text{Subject to} && \sum_{i=1}^n a_{i,j} \cdot y_i \leq 1 \quad \forall j \\
& && y_i \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\}
\end{aligned}$$

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	
$f(n_1) = f(17)$	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
$f(n_4) = f(74)$	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
$f(n_5) = f(1158)$	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
$f(n_6) = f(16384)$	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.