DEPARTAMENTO DE INGENIERÍA INDUSTRIAL
Optimización Avanzada 202420 – Tarea 2
PROFESOR: Andrés Medaglia
ASISTENTE: Laura Juliana Sánchez
MONITOR: Felipe de Jesús Liévano

Apellidos	Nombres	Código	Login	Quién entrega (Bloque Neón)
Bayona Vergara	William Andres	202011494	w.bayona	X
Vasquez Cristancho	Juan Martin	202113314	j.vasquezc	

Problema 1: Opti-drones

A.

1.1. Formulación matemática:

Conjuntos:

$$L: Conjunto\ de\ lugares$$
$$A: \{Hangar, Cultivo\ 1, Cultivo\ 2, \dots, Cultivo\ 25\}$$

$$A: Conjunto\ de\ arcos\ (i,j)\ posibles\ (Sin\ arcos\ donde\ i = j)$$
$$A: \{(Hangar, Cultivo\ 1), (Cultivo\ 1, Cultivo\ 2), \dots, (Cultivo\ 2, Cultivo\ 24), \dots\}$$

Parámetros:

$$X_l: Coodernada\ X\ en\ la\ que\ esta\ ubicado\ el\ lugar\ l \in L$$

$$Y_l: Coodernada\ Y\ en\ la\ que\ esta\ ubicado\ el\ lugar\ l \in L$$

$$M_{ij}: Distancia\ Manhattan\ entre\ dos\ ubicaciones$$
$$M_{ij} = (|X_i - X_j| + |Y_i - Y_j|)$$

$$F_l: Número\ de\ fotos\ que\ deben\ tomarse\ en\ el\ lugar\ l \in L$$

Variables de decisión:

$$x_{ij}: \begin{cases} 1, si\ el\ arco\ que\ conecta\ de\ ir\ del\ lugar\ i \in L\ al\ lugar\ j \in L\ existe \\ 0, dlc \end{cases}$$

Función Objetivo:

$$Min \sum_{i \in L} \sum_{j \in L} x_{ij} * M_{ij}$$

Restricciones:

1. Todo cultivo debe ser visitado por un dron

$$\sum_{j|(i,j) \in A} x_{ij} = 1 \ \ \forall\, i \in L | \, i \neq 0$$
$$\sum_{i|(i,j) \in A} x_{ij} = 1 \ \ \forall\, j \in L | \, j \neq 0$$

2. Se deben utilizar todos los drones disponibles (5 drones entran y salen de hangar)

$$\sum_{i|(i,0) \in A} x_{i0} = 5$$

$$\sum_{j|(0,j) \in A} x_{j0} = 5$$

3. No deben existir ciclos entre un par de ubicaciones

$$x_{ij} + x_{ji} \leq 1 \quad \forall\, i,j|(i,j) \in A, j \neq i$$

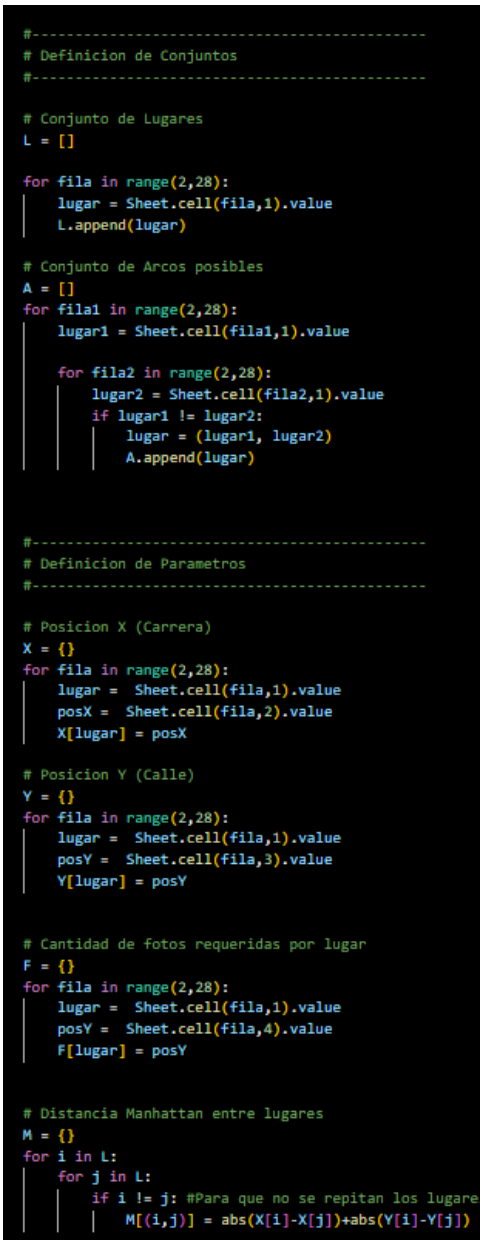
4. Naturaleza de las variables

$$x_{ij} \ \in \{0,1\}$$

1.2. Implementación del problema en Python-Gurobi y Resultados

A partir de la formulación anterior se planteó en Python-Gurobi la siguiente implementación.

En primer lugar, se crean las estructuras de datos leyéndolas desde el Excel que fue provisto:



```
#---------------------------------------------
# Definicion de Conjuntos
#---------------------------------------------

# Conjunto de Lugares
L = []

for fila in range(2,28):
    lugar = Sheet.cell(fila,1).value
    L.append(lugar)

# Conjunto de Arcos posibles
A = []
for fila1 in range(2,28):
    lugar1 = Sheet.cell(fila1,1).value

    for fila2 in range(2,28):
        lugar2 = Sheet.cell(fila2,1).value
        if lugar1 != lugar2:
            lugar = (lugar1, lugar2)
            A.append(lugar)



#---------------------------------------------
# Definicion de Parametros
#---------------------------------------------

# Posicion X (Carrera)
X = {}
for fila in range(2,28):
    lugar =  Sheet.cell(fila,1).value
    posX =  Sheet.cell(fila,2).value
    X[lugar] = posX

# Posicion Y (Calle)
Y = {}
for fila in range(2,28):
    lugar =  Sheet.cell(fila,1).value
    posY =  Sheet.cell(fila,3).value
    Y[lugar] = posY


# Cantidad de fotos requeridas por lugar
F = {}
for fila in range(2,28):
    lugar =  Sheet.cell(fila,1).value
    posY =  Sheet.cell(fila,4).value
    F[lugar] = posY


# Distancia Manhattan entre lugares
M = {}
for i in L:
    for j in L:
        if i != j: #Para que no se repitan los lugare
            M[(i,j)] = abs(X[i]-X[j])+abs(Y[i]-Y[j])
```

Después se implementan únicamente las restricciones planteadas en el enunciado:

```
#-------------------------------------------
# Restricciones del problema
#-------------------------------------------


#1. Todos los lugares deben ser visitados por un dron
for i in L:
    if i != 'Hangar':
        m.addConstr(quicksum(x[i,j] for j in L if i != j) == 1)

for j in L:
    if j != 'Hangar':
        m.addConstr(quicksum(x[i,j] for i in L if i != j) == 1)

#2. Restricción para que los drones salgan y lleguen al hangar
m.addConstr(quicksum(x['Hangar',j] for j in L if j != 'Hangar') == 5)
m.addConstr(quicksum(x[i,'Hangar'] for i in L if i != 'Hangar') == 5)


#3. Restricción para evitar ciclos de dos nodos
for i in L :
    for j in L :
        if i !=j:
            m.addConstr(x[i,j] + x[j,i] <= 1)
```

Después solo resta implementar la función objetico y optimizar el problema.

```
#F.O
m.setObjective(quicksum(x[a]*M[a] for a in A),GRB.MINIMIZE)
m.update()
m.setParam("Outputflag",0)
m.optimize()
```

```
328.0
+-------+-----------------------------------------------------------------------------------------------------------+---------------+--------+
| #Ruta |                                                 Secuencia                                                 | Tiempo (Horas) | Fotos |
+-------+-----------------------------------------------------------------------------------------------------------+---------------+--------+
|   1   |                                ['Cultivo 12', 'Cultivo 4', 'Cultivo 14', 'Cultivo 12']                    |      4.54     |  140   |
|   2   |                             ['Cultivo 6', 'Cultivo 3', 'Cultivo 23', 'Cultivo 21', 'Cultivo 6']           |      6.05     |  230   |
|   3   | ['Cultivo 19', 'Cultivo 5', 'Cultivo 16', 'Cultivo 25', 'Cultivo 15', 'Cultivo 7', 'Cultivo 10', 'Cultivo 2', 'Cultivo 19'] |     12.09     |  275   |
|   4   |                                ['Cultivo 1', 'Cultivo 17', 'Hangar', 'Cultivo 1']                         |      3.1      |   60   |
|   5   |                                ['Cultivo 9', 'Cultivo 22', 'Hangar', 'Cultivo 9']                         |      3.06     |  130   |
|   6   |                               ['Cultivo 11', 'Cultivo 8', 'Hangar', 'Cultivo 11']                         |      3.09     |   95   |
|   7   |                               ['Cultivo 20', 'Hangar', 'Cultivo 24', 'Cultivo 20']                        |      3.08     |   70   |
|   8   |                               ['Cultivo 18', 'Cultivo 13', 'Hangar', 'Cultivo 18']                        |      3.04     |  165   |
+-------+-----------------------------------------------------------------------------------------------------------+---------------+--------+
```

De esta manera se obtienen los siguientes resultados:

#Ruta	Secuencia	Tiempo (Horas)	Fotos
1	['Cultivo 14', 'Cultivo 12', 'Cultivo 4', 'Cultivo 14']	4.54	140
2	['Cultivo 3', 'Cultivo 23', 'Cultivo 21', 'Cultivo 6', 'Cultivo 3']	6.05	230
3	['Cultivo 2', 'Cultivo 19', 'Cultivo 5', 'Cultivo 16', 'Cultivo 25', 'Cultivo 15', 'Cultivo 7', 'Cultivo 10', 'Cultivo 2']	12.09	275
4	['Hangar', 'Cultivo 1', 'Cultivo 17', 'Hangar']	3.1	60
5	['Hangar', 'Cultivo 9', 'Cultivo 22', 'Hangar']	3.06	130
6	['Hangar', 'Cultivo 11', 'Cultivo 8', 'Hangar']	3.09	95
7	['Hangar', 'Cultivo 18', 'Cultivo 13', 'Hangar']	3.04	165
8	['Hangar', 'Cultivo 24', 'Cultivo 20', 'Hangar']	3.08	70

Tabla 1. Tabla solución problema inicial

Los cuales se ven de la siguiente forma al ser graficados:

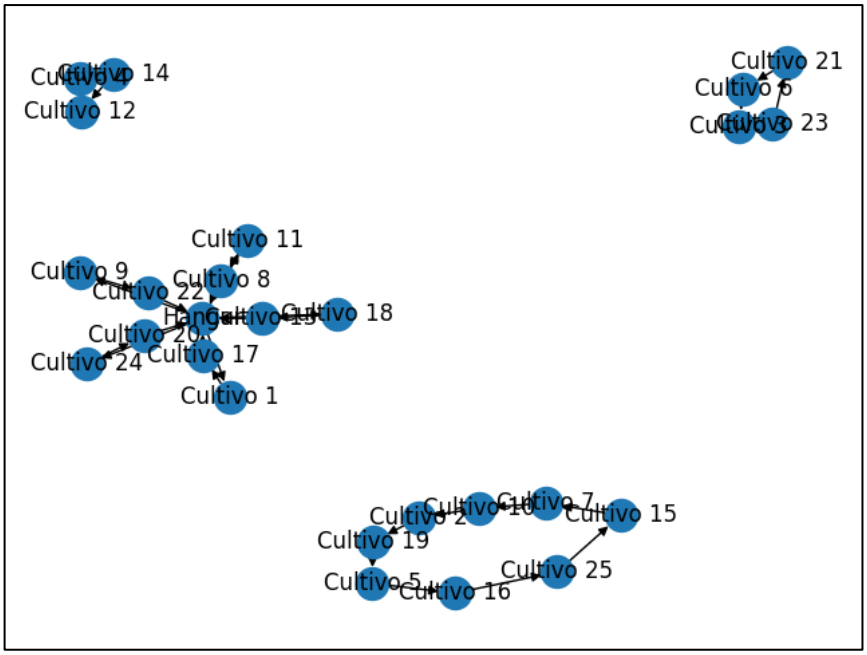


Figura 1. Grafo solución problema inicial

Este resultado cuenta con una función objetivo de **328.000 m (Distancia total recorrida).**

1.3. Conclusiones

Es muy aparente en la imagen 1 como esta solución no es factible en términos del problema dado. En primer lugar, hay más de 5 rutas, en total 8. Estas 3 rutas extras no pasan por el hangar (las rutas 1,2,3), lo cual las hace infectables. Además, se tiene el caso de la ruta 3 en la cual de llevarse a cabo terminaría con el dron en el piso pues, supera las 12 horas de autonomía del dron. Ninguna ruta supera la capacidad de fotografía de los drones (300 fotos).

B. Implementación de cortes para la

1.1. Formulación matemática

Los parámetros, conjuntos y función objetico de la anterior implementación se mantienen. Sin embargo, se agrega al modelo las restricciones planteadas en el enunciado que actuaran de manera iterativa sobre los siguientes subconjuntos del conjunto que acumula las rutas (o ciclos dentro del grafo) con problemas (uno para cada una de las restricciones de cortes iterativas).

Conjuntos:

Se define el conjunto rutas como un conjunto de listas en las cuales están los

lugares (en orden) de todas las rutas que se generaron en la parte A.

$$R: Conjunto\ de\ listas\ de\ lugares\ en\ el\ cual\ cada\ lista\ representa\ una\ ruta$$
$$R: \{[0,2,3], \dots, [\dots, 4,5,3,1]\}$$

Sub Conjuntos: Estos subconjuntos van cambiando con cada iteración donde se realizan los cortes

$$R_f: Subconjuto\ de\ dl\ conjunto\ R\ que\ acumila\ unicamente\ las\ rutas\ que\ sobrepasan\ las\ 300\ fotos$$
$$R_f: \{[0,1,3,4], \dots, [0, \dots, 4]\}$$
$$R_H: Subconjuto\ de\ dl\ conjunto\ R\ que\ acumila\ unicamente\ las\ rutas\ que\ no\ pasan\ por\ le\ hangar$$
$$R_f: \{[2,1,3,7], \dots, [5, \dots, 4]\}$$
$$R_A: Subconjuto\ de\ dl\ conjunto\ R\ que\ acumila\ unicamente\ las\ rutas\ que\ sobrepasan\ las\ 12\ horas\ en\ ruta$$
$$R_f: \{[0,6,8,6], \dots, [0, \dots, 5]\}$$

Restricciones:

1. Un dron puede tomar hasta 300 fotos:Esta restricción se ejecuta sobre cada ruta (o ciclo) que no cumple la condición. Se obliga a las rutas con mas de 300 fotos en ellas a reestructurase. Se itera sobre los nodos del subconjunto y hasta que la sumatoria de fotos sea inferior a las 300.

$$\sum_{j \in C} \sum_{i \in C} x_{ij} * F_i \leq 300\ \forall\ C \in R_f$$

2. Un dron tiene una autonomía de 12 horas:

   Esta restricción obliga a las rutas que superan 12 horas de recorrido a reestructurarse para cumplir con el tiempo de autonomía; utilizamos un parámetro auxiliar w_{ij} el cual toma el valor de 0 si estamos tratando con el hangar dentro de la ruta y de 1 en el caso contrario, esto debido a que ese tiempo de 1.5horas solo se considera en los lugares que no son el hangar; pues en el hangar los drones no se detienen a tomar fotografías.

$$\sum_{j \in C} \sum_{i \in C} x_{ij} * \frac{M_{ij}}{600} + 1.5 w_{ij} \leq 12\ \forall\ C \in R_A$$

Donde

$$w_j \begin{cases} 0\ si\ j = Hangar\ \vee\ i = Hangar \\ 1, dlc \end{cases}$$

3. Todas las rutas deben pasar por el hangar:

   Esta restricción aplica la lógica de obligar a conectar el clico que en este momento se encuentra separado de los demás (fuera del alcance del nodo pivote 'Hangar') a algún otro nodo fuera del ciclo (que a final será 'Hangar' pues esta restricción se aplicará hasta que el ciclo pase por dicho nodo.

$$\sum_{i \in C \mid i \neq 0} \sum_{j \in L \mid j \notin C} x_{ij} \geq 1 \quad \forall\, C \in R_H$$

1.2. Soluciones iterativas

De un total de 18 iteraciones se reporta el grafico de las siguientes 4. En las cuales se puede apreciar la naturaleza iterativa del problema como poco a poco las rutas son mejores:

Solución 0

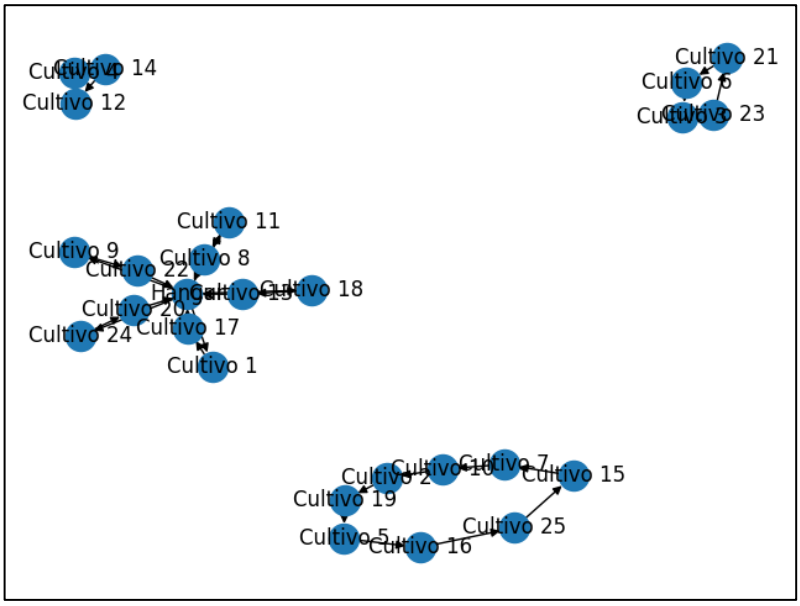


Figura 2. Grafo solución iteración 0

Solución 5

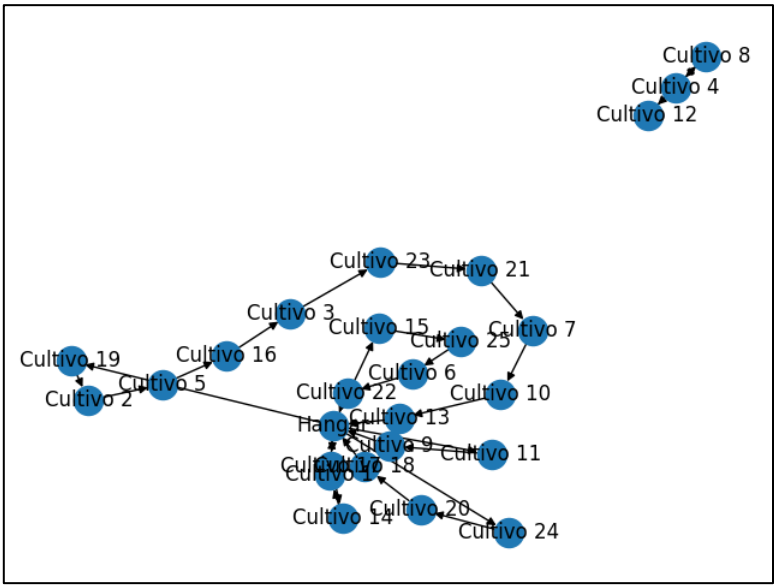


Figura 3. Grafo solución iteración 5

Solución 10

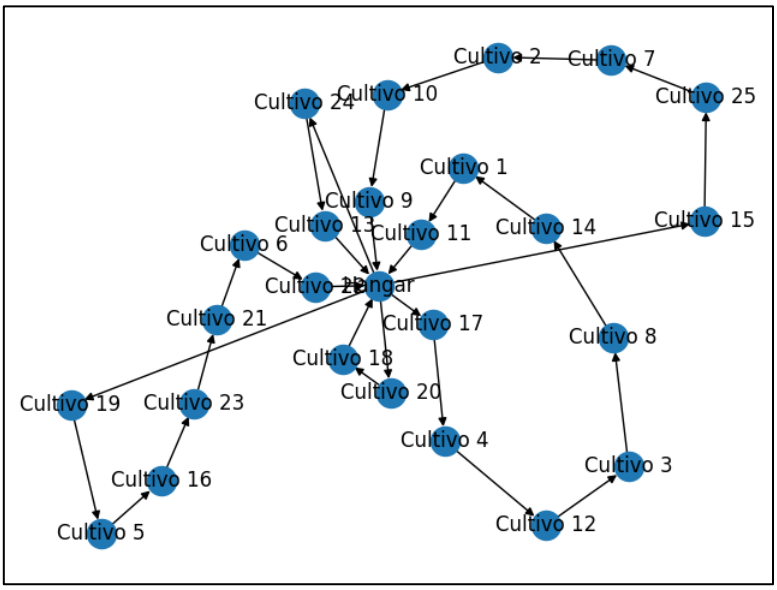


Figura 3. Grafo solución iteración 10

Solución 15

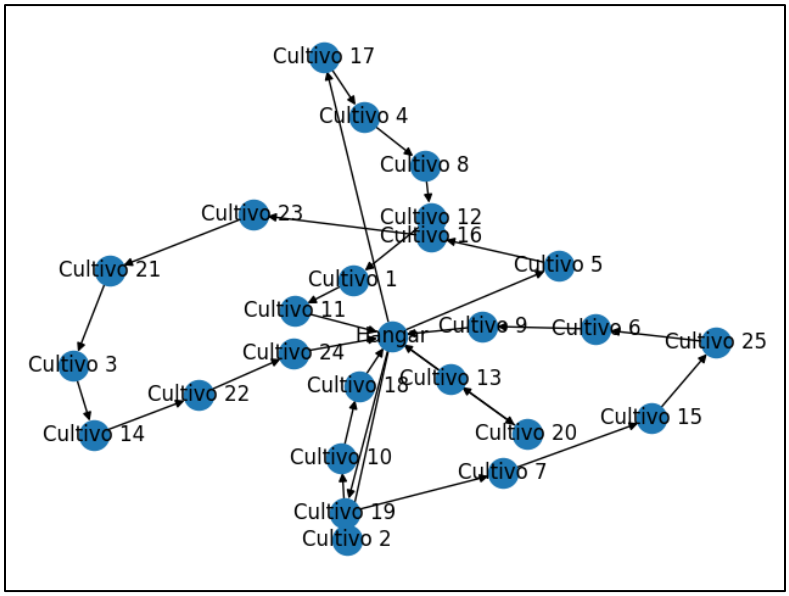


Figura 4. Grafo solución iteración 15

1.3. Reporte solución final

Después de las 18 iteraciones se llegó a la siguiente solución, en la cual todas las restricciones se cumplen.

#Ruta	Secuencia	Tiempo (Horas)	Fotos
1	['Cultivo 5', 'Cultivo 16', 'Cultivo 23', 'Cultivo 3', 'Cultivo 8', 'Cultivo 11', 'Hangar', 'Cultivo 2', 'Cultivo 5']	10.73	265
2	['Cultivo 9', 'Hangar', 'Cultivo 19', 'Cultivo 7', 'Cultivo 10', 'Cultivo 9']	6.16	205
3	['Cultivo 1', 'Cultivo 17', 'Cultivo 13', 'Hangar', 'Cultivo 15', 'Cultivo 25', 'Cultivo 6', 'Cultivo 1']	9.2	250
4	['Cultivo 22', 'Hangar', 'Cultivo 4', 'Cultivo 12', 'Cultivo 21', 'Cultivo 14', 'Cultivo 22']	7.71	285
5	['Cultivo 18', 'Hangar', 'Cultivo 24', 'Cultivo 20', 'Cultivo 18']	4.58	160

Tabla 2. Tabla solución problema inicial

El valor final de la función objetivo es de **528.000 m**

1.4. El algoritmo (pseudo código)

El algoritmo implementado sigue la siguiente lógica

En primer lugar, se implementan varias funciones auxiliares:

```
Timpo_secuencia -> numero #Tiempo que toma una secuencia
Cantiada_fotos -> numero "número de fotos en una secuencia
```

```
Nodos_ciclo -> lista "Nodos que contiene un ciclo"
Graficar -> void "Grafica la tabla y el grafo"
```

Después, a partir de los resultados de la parte A se define la función iterativa. La lógica detrás de esta función reconocer las rutas con problemas, e iterar sobre ellas agregando grupos de restricciones al modelo. Para después re optimizar y repetir el proceso hasta poder hallar rutas que cumplan con todas las restricciones del problema a la vez, para eso haciendo uso de un while. Una vez que se encuentra una solucion sale del ciclo:

```
Verificar_y_restringir ->

While true: #hasta que haya una solucion factible
G = GenerarGrafo(resultadosParteA) #se genera un grafo a partir de
la primera solucion (parte A)

Ciclos = G.getCiclos() #con una funcion de la librería se obtienen
todos los ciclos (rutas) actuales del grafo

Graficar () #se guarda el grafico de la solución actual

CiclosPasadosDeFotos = [ciclo for ciclo in ciclos if
cantidadFotos(ciclo)> 300] # obtención de ciclos que toman mas de
300 fotos
For ciclo in CiclosPasadosDeFotos:
     Modelo.agregarRestriccion(RestricciónPara300Fotos)     #la
restricción que fue planteada al inicio para la fotos

     CiclosPasadosDeTiempo = [ciclo for ciclo in ciclos if
TiempoSecuenia(ciclo) > 12)] # obtención de ciclos que tardan mas
de 12 horas
For ciclo in CiclosPasadosDeTiempo:
     Modelo.agregarRestriccion(RestricciónPara12Horas)#la
restricción que fue planteada al inicio para resolver problemas de
tiempo

CiclosSinHangar = [ciclo for ciclo in ciclos if 'Hangar' not in
ciclo] # obtención de ciclos que toman mas de 300 fotos
For ciclo in CiclosSinHangar:
     Modelo.agregarRestriccion(RestricciónPara300Fotos)     #la
restricción que fue planteada al inicio

If CiclosConProblemas == []: #no quden ciclos con problemas
     Break #solucion encontrada
```

```
Modelo.update()
Modelo.optimize()

actualizarSolucion() #se actualizan las soluciones para la
siguiente iteración crear el grafo
```

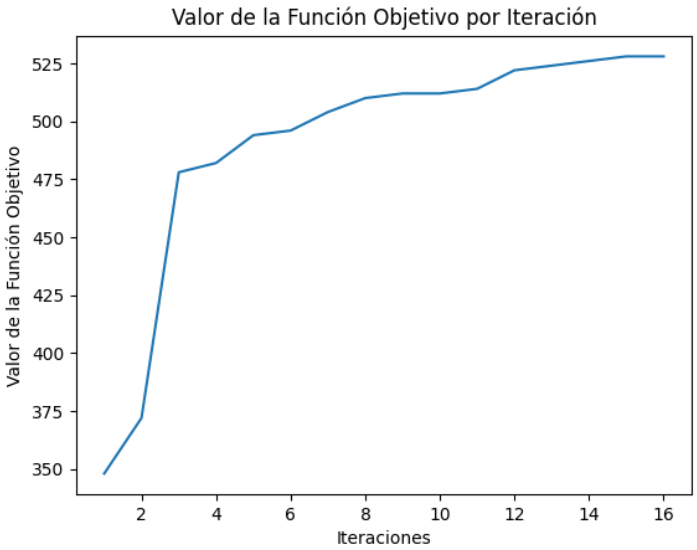
Esta función es usada de igual forma de manera iterativa dentro de un ciclo en el que se comparan soluciones, para asegurar que la solución actual sea la correcta, la invocación de la función se da de la siguiente manera:

```
SolucionActual -> list #itera sobre las las variables del problema
y las guarda

While true:

SolucionAnterior = SolucionActual()
Verificar_y_restringir()
arcos_reales = SolucionActual() #como se corrió de nuevo el
optimizador la solucion actual cambia
if arcos_reales == solucion_anterior:
   break

graficar()
```

De esta manera se implemento en el código para poder llegar a la respuesta.

1.5. Discusión función objetivo

La función objetivo va cambiando a lo largo de las iteraciones, mas en especifico este va creciendo conforme las rutas mejoran. Esto se debe a que cada vez que se agregan grupos de restricciones (cortes) se limita mas el problema lo cual hace que la función objetico crezca pues este es un problema de minimización.



Como dato adicional, por la manera como tenemos formulado nuestro código (Generar primero restricciones de Fotos, luego restricciones de Tiempo y finalmente restricciones de Hangar y ahí si ejecutar el optimizador). Abrió la posibilidad a obtener diferentes valores de la función objetivo basado en el orden en que se generan estas restricciones. Bajo este caso, se evaluó todas las posibles combinaciones del orden de creación de los diferentes tipos de restricciones para utilizar aquella que minimiza la FO.

En la siguiente tabla se encuentran los valores de las Funciones Objetivo basado en el orden en que se ejecutaron las funciones referentes a cada tipo de restricción.
H – Restricción Hangar
T – Restricción Tiempo
F – Restricción Foto

Orden	Valor FO
H T F	532
H F T	538
T H F	538
T F H	540
F H T	560
F T H	528