Programación Entera - Parte 1

Investigación Operativa, Universidad de San Andrés

Si encuentran algún error en el documento o hay alguna duda, mandenmé un mail a rodriguezf@udesa.edu.ar y lo revisamos.

1. Introducción a la Programación Entera

La **programación entera** (PE) es una rama de la optimización matemática en la cual se busca la mejor solución a problemas cuyas variables de decisión deben tomar valores enteros. Dentro de la PE distinguimos dos grandes familias:

- **Programación Entera Pura**: *todas* las variables son enteras.
- **Programación Mixta Entera**: solo una parte de las variables son enteras (generalmente binarias) y el resto pueden ser continuas.

A diferencia de la Programación Lineal (PL), la PE suele requerir algoritmos de *ramificación y acotación*, *cortes* y otras heurísticas. Esto se debe a que la región factible deja de ser convexa una vez que exigimos integralidad.

1.1. ¿Cuándo aparece la PE?

La necesidad de variables enteras surge, por ejemplo, cuando:

- Tomamos decisiones de tipo *sí / no* (abrir o no una planta, invertir o no en un proyecto, etc.).
- Trabajamos con ítems indivisibles (personas, máquinas, micros, etc.).
- Modelamos relaciones lógicas (mutua exclusión, dependencia, *al menos / a lo sumo* k, etc.).

2. Modelado típico con variables binarias

Las variables binarias $y \in \{0, 1\}$ permiten representar un amplio abanico de restricciones lógicas.

2.1. Mutuamente excluyentes

Para que dos actividades x_1 y x_2 no puedan realizarse simultáneamente, basta con exigir lo siguiente:

$$y_1 + y_2 \leq 1, \qquad y_j \in \{0, 1\}.$$

2.2. Dependencia (precedencia)

Si la actividad B solo puede realizarse cuando también se realiza la A lo planteamos de esta manera:

$$y_B \leq y_A.$$

2.3. Costos fijos de puesta en marcha

Para abrir un turno, una planta o cualquier recurso con costo fijo F y costo variable c, debemos pensarlo así:

$$\begin{aligned} Fy + cx \quad &\text{(costo)} \\ x \leq My \quad &\text{(capacidad)} \end{aligned}$$

3. Ejemplos prácticos

3.1. Problema de inversión

Los integrantes de una mesa directiva analizan 6 inversiones. Cada una puede realizarse a lo sumo una vez y difieren en el valor presente neto (VPN) que generan y el capital inicial requerido.

Inversión	1	2	3	4	5	6
Ganancia estimada	15	12	16	18	9	11
Capital requerido	38	33	39	45	23	27

El capital disponible es sólo de 100 millones. Las oportunidades 1 y 2 son mutuamente excluyentes, al igual que las 3 y 4. Además, ni la opción 3 ni la 4 pueden hacerse a menos que se lleve a cabo la 1 o la 2. El objetivo es, obviamente, maximizar la ganancia total.

3.1.1. Resolución

Una posible formulación utiliza $y_j \in \{0, 1\}$ indicando si se selecciona la inversión j. La función objetivo consiste en multiplicar la ganancia de cada proyecto por la variable binaria que indica si se selecciona o no.

$$\text{Max } Z = 15y_1 + 12y_2 + 16y_3 + 18y_4 + 9y_5 + 11y_6$$

Las restricciones quedan de la siguiente manera:

$$\begin{aligned}
&38y_1 + 33y_2 + 39y_3 + 45y_4 + 23y_5 + 27y_6 \leq 100 \quad \text{(capital disponible)} \\
&y_1 + y_2 \leq 1 \quad \text{(mutuamente excluyentes)} \\
&y_3 + y_4 \leq 1 \quad \text{(mutuamente excluyentes)} \\
&y_3 \leq y_1 + y_2 \quad \text{(si se hace la 1 o la 2, se puede hacer la 3 o la 4)} \\
&y_4 \leq y_1 + y_2 \quad \text{(si se hace la 1 o la 2, se puede hacer la 3 o la 4)} \\
&y_j \in \{0, 1\} \quad \forall j \quad \text{(variables binarias)}
\end{aligned}$$

3.1.2. Resolución con PICOS

```python
import picos

# Crear el problema
P = picos.Problem()

# Variables binarias (una por inversion)
y = picos.BinaryVariable('y', 6)

# Funcion objetivo
P.set_objective('max', 15*y[0] + 12*y[1] + 16*y[2] + 18*y[3] + 9*y
    [4] + 11*y[5])

# Restricciones
P.add_constraint(38*y[0] + 33*y[1] + 39*y[2] + 45*y[3] + 23*y[4] +
    27*y[5] <= 100)
P.add_constraint(y[0] + y[1] <= 1)  # inversiones 1 y 2 excluyentes
P.add_constraint(y[2] + y[3] <= 1)  # inversiones 3 y 4 excluyentes
P.add_constraint(y[2] <= y[0] + y[1])  # precedencia
```

```
17 P.add_constraint(y[3] <= y[0] + y[1])  # precedencia
18
19 # Resolver
20 P.options.verbosity = 0
21 P.solve(solver='glpk')
22
23 print(y)          # valores optimos de las variables
24 print(P.value)    # ganancia total optima
```

3.1.3. Salida de la consola

```
[ 1.00e+00]
[ 0.00e+00]
[ 1.00e+00]
[ 0.00e+00]
[ 1.00e+00]
[ 0.00e+00]
40.0
```

3.2. Plantas y Productos

Una compañía producirá cuatro productos empleando tres plantas con capacidad ociosa. El esfuerzo productivo por unidad es comparable en todas las plantas; por lo tanto, la capacidad disponible se mide en unidades de cualquier producto por día.

	Costo unitario ($/u)				Capacidad
	1	2	3	4	disponible (u/día)
Planta 1	41	27	28	24	75
Planta 2	40	29	–	23	75
Planta 3	37	30	27	21	45

La demanda diaria proyectada es:

Producto 1: 20 Producto 2: 30 Producto 3: 30 Producto 4: 40

La dirección debe decidir la asignación con dos criterios:

a) **Permitir división**: un producto puede fabricarse en más de una planta (modelo de transporte).

b) **No permitir división**: cada producto se fabrica en una sola planta (modelo de asignación binaria).

3.2.1. Resolución

La función objetivo entonces es minimizar el costo total de producción, el cual se compone de la cantidad de producto fabricado por el costo unitario de producción. No tenemos un costo por poner en marcha la fábrica. Siendo c_{ij} el costo unitario de producción del producto j en la planta i y x_{ij} la cantidad de producto j fabricado en la planta i, nos queda la siguiente función objetivo:

$$\text{Min } Z = \sum_{i=1}^{3} \sum_{j=1}^{4} c_{ij} x_{ij}$$

Vamos entonces ahora a pensar las restricciones. En primer lugar tenemos que pensar las restricciones de capacidad de las plantas.

$$\sum_{j=1}^{4} x_{ij} \leq cap_i \quad \forall i$$

En segundo lugar, vamos a pensar en las restricciones de demanda.

$$\sum_{i=1}^{3} x_{ij} = demanda_j \quad \forall j$$

Por último, vamos a pensar en las restricciones de división. Si no permitimos división, entonces cada producto se fabrica en una sola planta. Si permitimos división, entonces cada producto puede fabricarse en más de una planta. Basta entonces con tener o no esta restricción para controlar eso.

$$\sum_{i=1}^{3} x_{ij} = 1 \quad \forall j$$

3.2.2. Resolución con PICOS

```
import picos
import numpy as np

costos = np.array([
    [41, 27, 28, 24],
    [40, 29, 1e6, 23],  # 1e6 (1000000) es un costo prohibitivo para producto 3 en planta 2
    [37, 30, 27, 21]
])
capacidades = [75, 75, 45]
demanda = [20, 30, 30, 40]

plantas, productos = 3, 4

# ---- a) Permitir division (modelo de transporte) ----
P = picos.Problem()
x = picos.RealVariable('x', (plantas, productos), lower=0)

# Objetivo
P.set_objective('min', picos.sum([
    costos[i, j] * x[i, j]
    for i in range(plantas)
    for j in range(productos)
]))

# Restricciones de capacidad
for i in range(plantas):
    P.add_constraint(picos.sum(x[i, :]) <= capacidades[i])

# Restricciones de demanda
for j in range(productos):
    P.add_constraint(picos.sum(x[:, j]) == demanda[j])

P.solve(solver='glpk')
print('--- Modelo de transporte (division permitida) ---')
print("Asignacion continua (x):")
print(np.round(x.value, 2))
print("Costo total:", round(P.value, 2))

# ---- b) No permitir division (modelo de asignacion binaria) ----
costos_lista = costos.tolist()  # conversion segura para picos
demanda_lista = list(demanda)

P2 = picos.Problem()
```

```
y = picos.BinaryVariable('y', (plantas, productos))

# Objetivo
P2.set_objective('min', picos.sum([
    costos_lista[i][j] * demanda_lista[j] * y[i, j]
    for i in range(plantas)
    for j in range(productos)
]))

# Cada producto se fabrica en una sola planta
for j in range(productos):
    P2.add_constraint(picos.sum([y[i, j] for i in range(plantas)])
    == 1)

# Restricciones de capacidad
for i in range(plantas):
    P2.add_constraint(picos.sum([
        y[i, j] * demanda_lista[j]
        for j in range(productos)
    ]) <= capacidades[i])

P2.solve(solver='glpk')
print('--- Modelo de asignacion binaria (sin division) ---')
print("Asignacion binaria (y):")
asignacion = np.array([[int(round(y[i, j].value)) for j in range(
    productos)] for i in range(plantas)])
print(asignacion)
print("Costo total:", round(P2.value, 2))
```

3.2.3. Salida de la consola

```
--- Modelo de transporte (division permitida) ---
Asignacion continua (x):
[[ 0. 30. 30.  0.]
[ 0.  0.  0. 15.]
[20.  0.  0. 25.]]
Costo total: 3260.0
--- Modelo de asignacion binaria (sin division) ---
Asignacion binaria (y):
[[0 1 1 0]
[1 0 0 0]
[0 0 0 1]]
Costo total: 3290.0
```

3.3. Estaciones de Bomberos

Se pretende ubicar dos estaciones de bomberos en una comunidad dividida en cinco sectores. No se pueden tener más de una estación en un mismo sector. Una estación solo puede atender a su propio sector y a los otros que le sean asignados. La matriz muestra el tiempo medio de respuesta (minutos) si una estación ubicada en el sector i atiende un incendio en el sector j.

	1	2	3	4	5
1	5	12	30	20	15
2	20	4	15	10	25
3	15	20	6	15	12
4	25	15	25	4	10
5	10	25	15	12	5

La frecuencia promedio de incendios es de 2 por día en el sector 1, 1 por día en el sector 2, 3 por día en el sector 3, 1 por día en el sector 4 y 3 por día en el sector 5.

3.3.1. Resolución

La forma de pensarlo es que tenemos las siguientes variables:

→ y_i: 1 si se instala una estación en el sector i. Variable binaria.

→ x_{ij}: 1 si el sector j es atendido por la estación ubicada en i. Variable binaria.

Además, tenemos un vector de frecuencias:

→ $f = (2, 1, 3, 1, 3)$: Frecuencia promedio de incendios por sector.

Lo que queremos minimizar es el tiempo de respuesta a cada sector, dados por la matriz de tiempos de respuesta que llamamos t. Nos queda entonces que la función objetivo es:

$$\text{Min } Z = \sum_{i=1}^{5} \sum_{j=1}^{5} t_{ij} \cdot x_{ij} \cdot f_j$$

Para poner las restricciones debemos pensar en que en primer lugar no se puede tener más de una estación en un mismo sector, y en segundo lugar una estación solo puede atender a su propio sector y a los otros que le sean asignados (es decir, son mutuamente excluyentes). Además, queremos poner exactamente dos estaciones.

$$
\begin{aligned}
x_{ij} &\leq y_i \quad \forall i, j \\
\sum_{j=1}^{5} x_{ij} &= 1 \quad \forall i \\
\sum_{i=1}^{5} y_i &= 2
\end{aligned}
$$

3.3.2. Resolución con PICOS

```python
import picos
import numpy as np

# Matriz de tiempos de respuesta
tiempos = np.array([
    [5, 12, 30, 20, 15],
    [20, 4, 15, 10, 25],
    [15, 20, 6, 15, 12],
    [25, 15, 25, 4, 10],
    [10, 25, 15, 12, 5]
])

frecuencias = np.array([2, 1, 3, 1, 3])
sectores = 5

P = picos.Problem()

# Variables binarias
y = picos.BinaryVariable('y', sectores)  # y[i]: 1 si hay estacion en sector i
x = picos.BinaryVariable('x', (sectores, sectores))  # x[i,j]: 1 si sector j es atendido por estacion en i

# Objetivo: minimizar tiempo de respuesta ponderado por frecuencia
P.set_objective('min', picos.sum([
    tiempos[i, j] * x[i, j] * frecuencias[j]
    for i in range(sectores)
```

```
    for j in range(sectores)
]))

# Restriccion: cada sector es atendido por una sola estacion
for j in range(sectores):
    P.add_constraint(picos.sum([x[i, j] for i in range(sectores)])
    == 1)

# Restriccion: solo puede atender si hay estacion en i
for i in range(sectores):
    for j in range(sectores):
        P.add_constraint(x[i, j] <= y[i])

# Restriccion: exactamente dos estaciones
P.add_constraint(picos.sum(y) == 2)

P.solve(solver='glpk')
print('Asignacion (x):')
print(np.array([[int(round(x[i, j].value)) for j in range(sectores)]
    for i in range(sectores)]))
print('Estaciones ubicadas (y):')
print(np.array([int(round(y[i].value)) for i in range(sectores)]))
print('Tiempo total ponderado:', round(P.value, 2))
```

3.3.3. Salida de la consola

```
Asignacion (x):
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 1 1 0 0]
 [0 0 0 0 0]
 [1 0 0 1 1]]
Estaciones ubicadas (y):
[0 0 1 0 1]
Tiempo total ponderado: 85.0
```

Es decir, la estación en el sector 3 atiende a los sectores 2 y 3, y la estación en el sector 5 atiende a los sectores 1, 4 y 5.