Programación Entera - Parte 1

Investigación Operativa

¿Qué es la Programación Entera?

Definición:

- Buscamos la mejor solucion a problemas donde las variables deben ser enteras
- Hay dos familias: **pura** (todas enteras) y **mixta** (algunas enteras, otras continuas)
- No es convexa: requiere ramificación y acotación, cortes y heurísticas

¿Cuándo aparece la PE?

- Decisiones si/no (abrir planta, invertir, etc.)
- Ítems indivisibles (personas, máquinas, micros)
- Restricciones lógicas (mutua exclusión, dependencia, al menos/a lo sumo k)

Modelado con variables binarias

Las binarias $y \in \{0, 1\}$ permiten:

- Mutua exclusión: $y_1 + y_2 \leq 1$
- Dependencia: $y_B \leq y_A$
- Costos fijos y variables: $Fy + cx$
- Activaciones: $x \leq My$

Ejemplo: Problema de inversión

Se analizan 6 inversiones, cada una con ganancia y capital requerido. Capital total: 100. Algunas son mutuamente excluyentes y otras dependen de otras.

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

Modelo de inversión: Función objetivo

Variables: $y_j = 1$ si se selecciona la inversión j.

Queremos maximizar la ganancia total:

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

Modelo de inversión: Restricción de capital

No podemos invertir más de lo disponible:

$$38y_1 + 33y_2 + 39y_3 + 45y_4 + 23y_5 + 27y_6 \leq 100$$

(El capital total disponible es 100)

Modelo de inversión: Exclusión

Algunas inversiones son mutuamente excluyentes:

$$y_1 + y_2 \leq 1$$

$$y_3 + y_4 \leq 1$$

(No se pueden hacer ambas a la vez)

Modelo de inversión: Dependencia

Dependencia entre inversiones:

$$y_3 \leq y_1 + y_2$$
$$y_4 \leq y_1 + y_2$$

(Solo se puede hacer la 3 o la 4 si se hizo la 1 o la 2)

Modelo de inversión: Variables binarias

Las variables son binarias:

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

(Cada inversión se toma o no)

Modelo en PICOS: Definicion y restricciones

```
import picos

P = picos.Problem()
y = picos.BinaryVariable('y', 6)

P.set_objective('max', 15*y[0] + 12*y[1] + 16*y[2] + 18*y[3] + 9*y[4] + 11*y[5])
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)
P.add_constraint(y[2] + y[3] <= 1)
P.add_constraint(y[2] <= y[0] + y[1])
P.add_constraint(y[3] <= y[0] + y[1])
```

Modelo en PICOS: Resolucion y resultados

```
P.options.verbosity = 0
P.solve(solver='glpk')
print(y)          # valores optimos de las variables
print(P.value)    # ganancia total optima
```

Salida

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

Ejemplo: Plantas y productos

Cuatro productos, tres plantas, cada una con capacidad y costos distintos. Hay dos variantes: permitir o no permitir división de producción.

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

Demanda: 20, 30, 30, 40

Modelo: Función objetivo

Variables: x_{ij} cantidad de producto j fabricado en planta i.

Queremos minimizar el costo total de producción:

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

Modelo: Restricción de capacidad

Cada planta tiene una capacidad máxima:

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

(No se puede fabricar más de lo que permite la planta)

Modelo: Restricción de demanda

Se debe cubrir la demanda de cada producto:

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

(La suma de lo producido en todas las plantas debe igualar la demanda de cada producto)

Modelo: División o no división

¿Se permite fabricar un producto en más de una planta?

- **División permitida:** No se agrega ninguna restricción extra (modelo de transporte).
- **Sin división:** Cada producto se fabrica en una sola planta (modelo de asignación binaria):

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

donde $y_{ij} = 1$ si el producto j se fabrica en la planta i.

PICOS: modelo de transporte

```python
import picos
import numpy as np

costos = np.array([
    [41, 27, 28, 24],
    [40, 29, 1e6, 23],
    [37, 30, 27, 21]
])
capacidades = [75, 75, 45]
demanda = [20, 30, 30, 40]

plantas, productos = 3, 4
P = picos.Problem()
x = picos.RealVariable('x', (plantas, productos), lower=0)

P.set_objective('min', picos.sum([
    costos[i, j] * x[i, j]
    for i in range(plantas)
    for j in range(productos)
]))
for i in range(plantas):
    P.add_constraint(picos.sum(x[i, :]) <= capacidades[i])
for j in range(productos):
    P.add_constraint(picos.sum(x[:, j]) == demanda[j])
```

PICOS: resultados transporte

```python
P.solve(solver='glpk')
print('Asignacion continua (x):')
print(np.round(x.value, 2))
print('Costo total:', round(P.value, 2))
```

PICOS: modelo de asignacion binaria

```python
costos_lista = costos.tolist()
demanda_lista = list(demanda)

P2 = picos.Problem()
y = picos.BinaryVariable('y', (plantas, productos))

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)
]))
for j in range(productos):
    P2.add_constraint(picos.sum([y[i, j] for i in range(plantas)]) == 1)
for i in range(plantas):
    P2.add_constraint(picos.sum([
        y[i, j] * demanda_lista[j]
        for j in range(productos)
    ]) <= capacidades[i])
```

PICOS: resultados asignacion binaria

```python
P2.solve(solver='glpk')
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))
```

Salida

```
--- 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
```

Ejemplo: Estaciones de bomberos

Ubicar 2 estaciones en 5 sectores. Cada estación atiende a su propio sector y a los que le sean asignados. Queremos minimizar el tiempo de respuesta ponderado por frecuencia de incendios.

	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

Frecuencia: 2, 1, 3, 1, 3

Modelo de bomberos: Función objetivo

Variables:

- y_i: 1 si hay estación en el sector i
- x_{ij}: 1 si el sector j es atendido por la estación ubicada en i

Queremos minimizar el tiempo de respuesta ponderado:

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

donde t_{ij} es el tiempo de respuesta del sector j atendido por la estación en i y f_j es la frecuencia de incendios en el sector j.

Modelo de bomberos: Asignación

Cada sector debe ser atendido por una sola estación:

$$\sum_{i=1}^{5} x_{ij} = 1 \quad \forall j$$

(Cada sector tiene que estar cubierto por alguna estación)

Modelo de bomberos: Cobertura

Solo se puede asignar un sector a una estación si esa estación existe:

$$x_{ij} \leq y_i \quad \forall i, j$$

(No se puede asignar un sector a un lugar donde no hay estación)

Modelo de bomberos: Cantidad de estaciones

Queremos instalar exactamente dos estaciones:

$$\sum_{i=1}^{5} y_i = 2$$

Modelo de bomberos: Variables

Resumen:

- $y_i \in \{0, 1\}$: 1 si hay estación en el sector i
- $x_{ij} \in \{0, 1\}$: 1 si el sector j es atendido por la estación en i

PICOS: modelo de bomberos

```
import picos
import numpy as np

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()
y = picos.BinaryVariable('y', sectores)
x = picos.BinaryVariable('x', (sectores, sectores))

P.set_objective('min', picos.sum([
    tiempos[i, j] * x[i, j] * frecuencias[j]
    for i in range(sectores)
    for j in range(sectores)
]))
for j in range(sectores):
    P.add_constraint(picos.sum([x[i, j] for i in range(sectores)]) == 1)
for i in range(sectores):
    for j in range(sectores):
        P.add_constraint(x[i, j] <= y[i])
P.add_constraint(picos.sum(y) == 2)
```

PICOS: resultados bomberos

```python
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))
```

Salida

```
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
```

Terminamos

¿Dudas?
¿Consultas?