Programación Lineal - Parte 2

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.

Problemas de Redes

Hasta ahora veníamos viendo problemas desestructurados, es decir, problemas que no tienen una estructura predefinida. En esos casos la formulación del programa lineal puede salir del enunciado rápidamente o con un poco de creatividad. Los problemas de redes son un tipo especial de programación lineal donde las variables de decisión representan flujos entre nodos conectados por arcos. Estos problemas al poder graficarlos nos ayuda mucho a poder visualizar el problema y resolverlo.

1. Problema de Transporte

El problema de transporte es uno de los problemas de redes más comunes. Consiste en determinar cómo transportar unidades desde varios orígenes hasta varios destinos minimizando el costo total, obviamente teniendo restricciones en el camino.

1.1. Ejemplo de Transporte

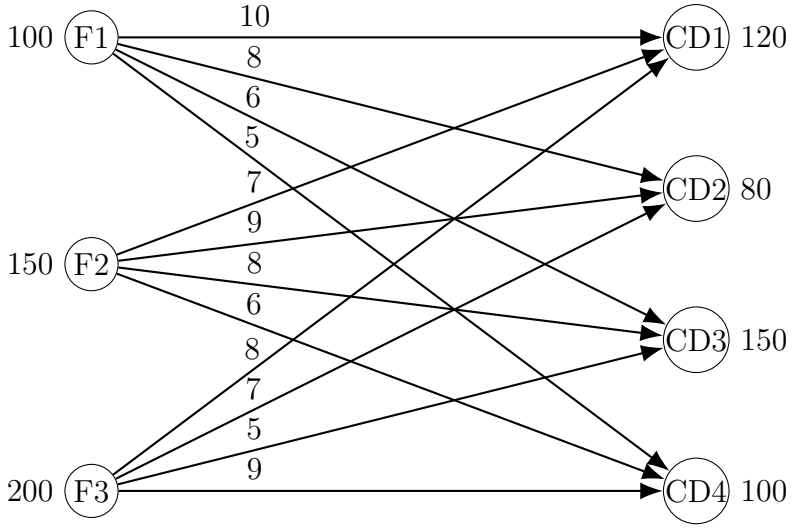
Una empresa tiene 3 fabricas y 4 centros de distribucion. Las fabricas pueden producir las siguientes cantidades: Fabrica 1 puede producir 100 unidades, Fabrica 2 puede producir 150 unidades y Fabrica 3 puede producir 200 unidades. Los centros de distribucion requieren: Centro 1 necesita 120 unidades, Centro 2 necesita 80 unidades, Centro 3 necesita 150 unidades y Centro 4 necesita 100 unidades.

El costo de transporte por unidad se muestra en la siguiente matriz:

	CD1	CD2	CD3	CD4
F1	10	8	6	5
F2	7	9	8	6
F3	8	7	5	9

1.1.1. Gráfico

A la izquierda de cada nodo de fábrica ponemos la capacidad de producción de la fábrica. A la derecha de cada nodo de centro de distribución ponemos la demanda de ese centro de distribución. Trazamos todas las lineas que conectan los nodos de fábricas con los nodos de centros de distribución y encima de ellas ponemos el costo de transporte por unidad. Si hubiera una fábrica que no tiene distribución a un centro, esa línea no se traza.



1.1.2. Formulación

Para las variables de decisión en este tipo de problemas tenemos dos formas de anotarlo. Una sería la forma más mátematica:

- x_{ij}: cantidad transportada desde la fábrica i al centro j

O bien la forma más práctica:

- x_i: cantidad transportada por cada camino i

Con la función objetivo nos pasaría lo mismo. Una es la forma más mátematica:

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

O bien la forma más práctica:

$$\text{Minimizar: } Z = \sum_{i=1}^{12} x_i$$

Con las restricciones, nuevamente, es lo mismo. Si fueramos por la forma más matemática, en donde i es cada fábrica y j es cada centro de distribución, tendríamos:

$$\begin{aligned}
\sum_{j=1}^{4} x_{1j} &= 100 && \text{(capacidad F1)} \\
\sum_{j=1}^{4} x_{2j} &= 150 && \text{(capacidad F2)} \\
\sum_{j=1}^{4} x_{3j} &= 200 && \text{(capacidad F3)} \\
\sum_{i=1}^{3} x_{i1} &= 120 && \text{(demanda CD1)} \\
\sum_{i=1}^{3} x_{i2} &= 80 && \text{(demanda CD2)} \\
\sum_{i=1}^{3} x_{i3} &= 150 && \text{(demanda CD3)} \\
\sum_{i=1}^{3} x_{i4} &= 100 && \text{(demanda CD4)} \\
x_{ij} &\geq 0 && \forall i, j
\end{aligned}$$

Notar que en cada caso de los anteriores, una de las variables queda fija. Por el contrario, si fueramos con que i es cada camino y nada más, las restricciones nos quedan de la siguiente manera:

$$\begin{aligned}
\sum_{i=1}^{4} x_i &= 100 && \text{(capacidad F1)} \\
\sum_{i=5}^{8} x_i &= 150 && \text{(capacidad F2)}
\end{aligned}$$

$$\sum_{i=9}^{12} x_i = 200 \qquad \text{(capacidad F3)}$$
$$x_1 + x_5 + x_9 = 120 \qquad \text{(demanda CD1)}$$
$$x_2 + x_6 + x_{10} = 80 \qquad \text{(demanda CD2)}$$
$$x_3 + x_7 + x_{11} = 150 \qquad \text{(demanda CD3)}$$
$$x_4 + x_8 + x_{12} = 100 \qquad \text{(demanda CD4)}$$
$$x_i \geq 0 \qquad \forall i$$

Ambas formas son válidas y pueden usar la que quieran y se sientan más cómodos.

1.1.3. Implementación

```python
import picos
import numpy as np

# Crear el problema
P = picos.Problem()

# Definir variables
x = picos.RealVariable('x', 12)

# Definir costos
c = np.array([10, 8, 6, 5, 7, 9, 8, 6, 8, 7, 5, 9])

# Definir funcion objetivo
P.set_objective('min', picos.sum([c[i] * x[i] for i in range(12)]))

# Restricciones de capacidad
P.add_constraint(sum(x[0:4]) == 100)  # F1
P.add_constraint(sum(x[4:8]) == 150)  # F2
P.add_constraint(sum(x[8:12]) == 200) # F3

# Restricciones de demanda
P.add_constraint(x[0] + x[4] + x[8] == 120)    # CD1
P.add_constraint(x[1] + x[5] + x[9] == 80)     # CD2
P.add_constraint(x[2] + x[6] + x[10] == 150)   # CD3
P.add_constraint(x[3] + x[7] + x[11] == 100)   # CD4

# Restricciones de no negatividad
P.add_constraint(x >= 0)

```

```python
# Resolver
P.solve(solver='glpk')

# Mostrar resultados
print("Solucion optima:")
for i in range(12):
    print(f"x_{i+1}: {x[i].value:.2f}")
print(f"Costo total: ${P.value:.2f}")
```

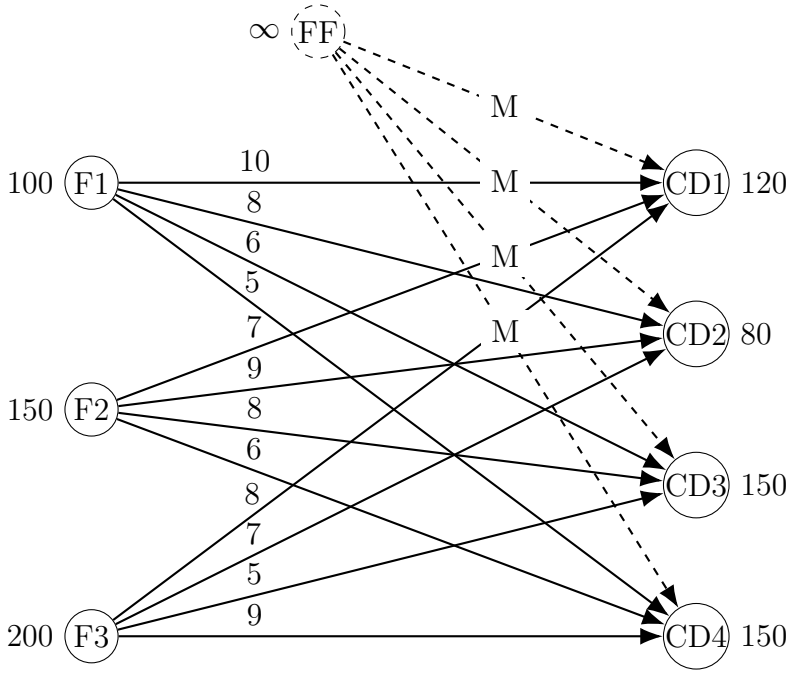
```
Solucion optima:
x_1: 0.00
x_2: 30.00
x_3: 0.00
x_4: 70.00
x_5: 120.00
x_6: 0.00
x_7: 0.00
x_8: 30.00
x_9: 0.00
x_10: 50.00
x_11: 150.00
x_12: 0.00
Costo total: $2710.00
```

2. Problema de Transporte Desbalanceado

En la práctica, es común encontrar situaciones donde la oferta total no coincide con la demanda total. En estos casos, tenemos un problema desbalanceado que requiere la introducción de variables de holgura (slack).

2.1. Ejemplo con Demanda Insatisfecha

Consideremos el ejemplo anterior pero con una demanda total mayor que la oferta. En este caso, debemos introducir una "fábrica ficticia" con un costo de penalización por demanda insatisfecha.



2.1.1. ¿Y qué cambia esto?

Al agregar la fábrica ficticia podemos llegar a suplir la demanda de los centros de distribución. El tema es que por cada unidad que se transporta desde la fábrica ficticia, se tiene un costo de penalización. Esto se agrega a la función objetivo de la siguiente manera:

$$\text{Minimizar: } Z = \sum_{i=1}^{3}\sum_{j=1}^{4} c_{ij}x_{ij} + M\sum_{i=1}^{3} x_i$$

Donde M es un número muy grande que representa el costo de penalización por demanda insatisfecha.

3. Problema de Transshipment

El problema de transshipment es una extensión del problema de transporte donde se permite que los nodos intermedios actúen como puntos de transferencia.

3.1. Sunco Oil Co.

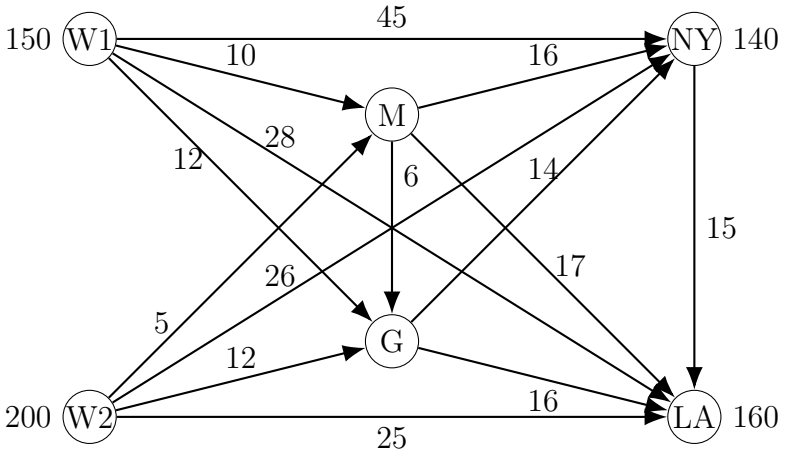
Sunco Oil Co. es una empresa petrolera que tiene dos pozos W1 y W2. La empresa provee de petróleo a la ciudad Nueva York y Los Ángeles. El Pozo W1 puede producir hasta 150.000 barriles de petróleo por día mientras que el W2 puede producir hasta 200.000 barriles de petróleo al día. NY tiene una demanda de 140.000 b/día mientras que LA tiene una demanda de 160.000 b/día. Además de enviar camiones directos a NY y LA Sunco Oil tiene la opción de enviarlo a dos puertos: Mobile y Galveston, donde después llegarán por barco a destino.

		Hasta					
		W1	W2	M	G	NY	LA
Pozo	W1	X	–	10	12	45	28
	W2	–	X	5	12	26	25
Puerto	M	–	–	X	6	16	17
	G	–	–	–	X	14	16
Ciudad	NY	–	–	–	–	X	15
	LA	–	–	–	–	–	X

¿Cómo debe ser el tránsito del petróleo para minimizar los costos y cubrir la demanda de NY y LA?

3.1.1. Gráfico

Hay que ser muy prolijos con el gráfico y no mezclarnos costos de distintos caminos.



3.1.2. Formulación

Como siempre, podemos tener x_i siendo cada camino, pero también lo podemos hacer más prolijo. Para las variables de decision:

- x_{W1M}: cantidad de petroleo transportada desde el pozo W1 al puerto M
- x_{W1G}: cantidad de petroleo transportada desde el pozo W1 al puerto G
- x_{W2M}: cantidad de petroleo transportada desde el pozo W2 al puerto M
- x_{W2G}: cantidad de petroleo transportada desde el pozo W2 al puerto G
- x_{W1NY}: cantidad de petroleo transportada desde el pozo W1 a NY
- x_{W1LA}: cantidad de petroleo transportada desde el pozo W1 a LA
- x_{W2NY}: cantidad de petroleo transportada desde el pozo W2 a NY
- x_{W2LA}: cantidad de petroleo transportada desde el pozo W2 a LA
- x_{MNY}: cantidad de petroleo transportada desde el puerto M a NY
- x_{MLA}: cantidad de petroleo transportada desde el puerto M a LA
- x_{MG}: cantidad de petroleo transportada desde el puerto M al puerto G
- x_{GNY}: cantidad de petroleo transportada desde el puerto G a NY
- x_{GLA}: cantidad de petroleo transportada desde el puerto G a LA
- x_{NYLA}: cantidad de petroleo transportada desde NY a LA

Funcion objetivo:

$$\text{Minimizar: } Z = \sum_{i=1}^{14} c_i x_i$$

donde c_i son los costos de cada camino y x_i son las variables de decision para cada camino.

O bien de forma explícita:

Minimizar: $Z = 10x_{W1M} + 12x_{W1G} + 5x_{W2M} + 12x_{W2G} + 45x_{W1NY} + 28x_{W1LA} + 26x_{W2NY} + 25x_{W2LA} + 16x_{MNY} + 17x_{MLA} + 6x_{MG} + 14x_{GNY} + 16x_{GLA} + 15x_{NYLA}$

Para las restricciones tenemos dos tipos, las de oferta y demanda, que son las siguientes:

$$\begin{aligned} x_{W1M} + x_{W1G} + x_{W1NY} + x_{W1LA} &\leq 150 && \text{(capacidad W1)} \\ x_{W2M} + x_{W2G} + x_{W2NY} + x_{W2LA} &\leq 200 && \text{(capacidad W2)} \\ x_{W1NY} + x_{W2NY} + x_{MNY} + x_{GNY} + x_{LA,NY} &= 140 && \text{(demanda NY)} \\ x_{W1LA} + x_{W2LA} + x_{MLA} + x_{GLA} + x_{NY,LA} &= 160 && \text{(demanda LA)} \\ x_{ij} &\geq 0 && \forall i, j \end{aligned}$$

Por otra parte, tenemos las restricciones de **balance** para asegurarnos de todo lo que llega a M y a G efectivamente se vaya:

$$\begin{aligned} x_{W1M} + x_{W2M} &= x_{MNY} + x_{MLA} + x_{MG} && \text{(balance M)} \\ x_{W1G} + x_{W2G} + x_{MG} &= x_{GNY} + x_{GLA} && \text{(balance G)} \end{aligned}$$

3.1.3. Implementación

```python
import picos
import numpy as np

# Crear el problema
P = picos.Problem()

# Definir variables de decision
x = picos.RealVariable('x', 14)  # 14 caminos posibles

# Definir costos
c = np.array([10, 12, 5, 12, 45, 28, 26, 25, 16, 17, 6, 14, 16, 15])
c = picos.Constant('c', c)

# Definir funcion objetivo
P.set_objective('min', c | x)

# Restricciones de oferta
P.add_constraint(x[0] + x[1] + x[4] + x[5] <= 150)  # W1
```

```
P.add_constraint(x[2] + x[3] + x[6] + x[7] <= 200)  # W2

# Restricciones de demanda
P.add_constraint(x[4] + x[6] + x[8] + x[11] + x[13] == 140)  # NY
P.add_constraint(x[5] + x[7] + x[9] + x[12] + x[13] == 160)  # LA

# Restricciones de balance
P.add_constraint(x[0] + x[2] == x[8] + x[9] + x[10])  # M
P.add_constraint(x[1] + x[3] + x[10] == x[11] + x[12])  # G

# Restricciones de no negatividad
P.add_constraint(x >= 0)

# Resolver
P.solve(solver='glpk')

print("Valores optimos de las variables:")
print(x)
print("\nCosto total minimo:", P.value)
```

3.2. Salida de la consola

```
Valores optimos de las variables:
[ 0.00e+00]
[ 0.00e+00]
[ 2.00e+01]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 2.00e+01]
[ 0.00e+00]
[ 0.00e+00]
[ 0.00e+00]
[ 1.40e+02]

Costo total minimo: 2540.0
```

4. Problema de Flujo Máximo

El problema de flujo máximo busca determinar el máximo flujo posible desde un nodo fuente hasta un nodo sumidero, respetando las capacidades de los arcos. Es como una red de cañerías, donde cada arco tiene una capacidad máxima y el flujo que pasa por cada arco es el que se transporta.

4.1. Sunco Oil Co. (Continuación)

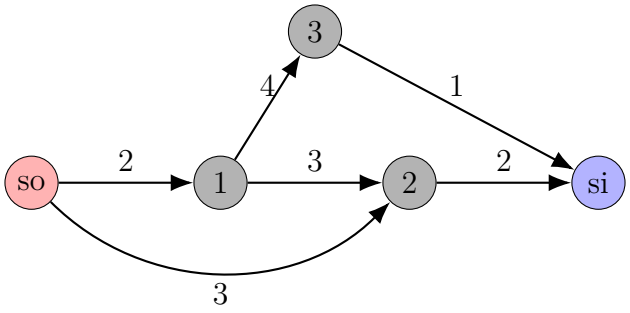
Sunco Oil quiere transportar la máxima cantidad de petróleo posible (por hora) mediante tubos desde el nodo So (source) hasta Si (sink). El petróleo debe si o si pasar a traves de ciertas estaciones de la empresa: 1, 2 y 3. Los distintos arcos en el diagrama representan los tubos, los cuales tienen distintos diametros. El número máximo de barriles de petróleo que pueden ser bombeados a traves de cada arco se muestra en la siguiente Tabla. Cada número se lo denomina capacidad del tubo o del arco. Determinar cuanto es el máximo número de barriles de petróleo por hora que se pueden enviar de So a Si.

Los costos (capacidades) de los arcos se encuentran en la siguiente tabla:

Arco	**Capacidad**
so-1	2
so-2	3
1-2	3
1-3	4
3-si	1
2-si	2

4.1.1. Gráfico

Para armar el gráfico de flujo máximo simplemente creamos cada nodo y luego trazamos los arcos con sus respectivas capacidades prestando atención a no equivocarnos.



4.1.2. Formulación

Las variables de decisión van a ser cada uno de los caminos, y cuánto pasa por ahí:

- x_i: flujo en el arco i

La función objetivo es simplemente maximizar lo que sale de *so* (o lo que llega a *si*):

$$\text{Maximizar: } Z = x_1 + x_2$$

Las restricciones es donde está lo importante en los problemas de flujo máximo. Tenemos que asegurarnos de que lo que entra a cada nodo sea igual a lo que sale. Por ejemplo, en el nodo 1, lo que entra es $x_1 + x_2$ y lo que sale es $x_3 + x_4$. Por lo tanto, tenemos que:

$$\begin{aligned}
x_1 + x_2 &= x_5 + x_6 && \text{(salida igual a llegada)} \\
x_1 + x_2 &= x_3 + x_4 && \text{(conservacion nodo 1)} \\
x_2 + x_4 &= x_6 && \text{(conservacion nodo 2)} \\
x_3 &= x_5 && \text{(conservacion nodo 3)} \\
x_1 &\leq 2 && \text{(capacidad so-1)} \\
x_2 &\leq 3 && \text{(capacidad so-2)} \\
x_3 &\leq 3 && \text{(capacidad 1-2)} \\
x_4 &\leq 4 && \text{(capacidad 1-3)} \\
x_5 &\leq 1 && \text{(capacidad 3-si)} \\
x_6 &\leq 2 && \text{(capacidad 2-si)} \\
x_i &\geq 0 && \forall i
\end{aligned}$$

5. Implementación con PICOS

```python
import picos
import numpy as np

# Crear el problema
P = picos.Problem()

# Definir variables
x = picos.RealVariable('x', 6)

# Definir funcion objetivo
P.set_objective('max', x[0] + x[1])

# Restricciones de conservacion de flujo
P.add_constraint(x[0] + x[1] == x[4] + x[5])  # salida igual a
    llegada
P.add_constraint(x[0] + x[1] == x[2] + x[3])  # conservacion nodo 1
P.add_constraint(x[1] + x[3] == x[5])         # conservacion nodo 2
P.add_constraint(x[2] == x[4])                # conservacion nodo 3

# Restricciones de capacidad
P.add_constraint(x[0] <= 2)  # capacidad so-1
P.add_constraint(x[1] <= 3)  # capacidad so-2
P.add_constraint(x[2] <= 3)  # capacidad 1-2
P.add_constraint(x[3] <= 4)  # capacidad 1-3
P.add_constraint(x[4] <= 1)  # capacidad 3-si
P.add_constraint(x[5] <= 2)  # capacidad 2-si

# Restricciones de no negatividad
P.add_constraint(x >= 0)

# Resolver
P.solve(solver='glpk')

# Mostrar resultados
print("Solucion optima:")
for i in range(6):
    print(f"x_{i+1}: {x[i].value:.2f}")
print(f"Flujo maximo: {P.value:.2f}")
```

5.1. Salida de la consola

```
Solucion optima:
x_1: 2.00
x_2: 0.00
```

```
x_3: 1.00
x_4: 1.00
x_5: 1.00
x_6: 1.00
Flujo maximo: 2.00
```