

Programación Lineal - Pt 2

Problemas de Redes

Investigación Operativa



¿Qué son los Problemas de Redes?

Definición:

- Tipo especial de programación lineal
- Variables representan flujos entre nodos
- Nodos conectados por arcos con capacidades
- Visualización gráfica facilita el análisis

Ventajas:

- Estructura predefinida
- Fácil interpretación
- Algoritmos especializados más eficientes


Tipos de Problemas de Redes

Principales categorías:

 **Transporte:** Envío desde orígenes a destinos

 **Transshipment:** Permite nodos intermedios

 **Flujo Máximo:** Maximizar flujo en la red

 **Asignación:** Matching uno a uno

Problema de Transporte

Características:

- Múltiples orígenes con capacidades
- Múltiples destinos con demandas
- Costo por unidad transportada
- Objetivo: minimizar costo total

Aplicaciones:

- Distribución de productos
- Logística de almacenes
- Asignación de recursos

Ejemplo: Fábricas y Centros de Distribución

Situación:

- 3 fábricas con capacidades: 100, 150, 200 unidades
- 4 centros con demandas: 120, 80, 150, 100 unidades
- Costos de transporte variables

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

Representación Gráfica

100 (F1)

(CD1) 120

150 (F2)

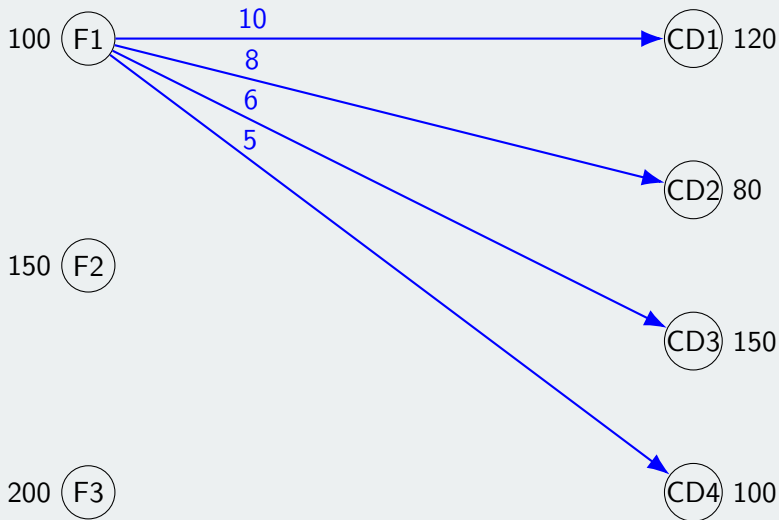
(CD2) 80

200 (F3)

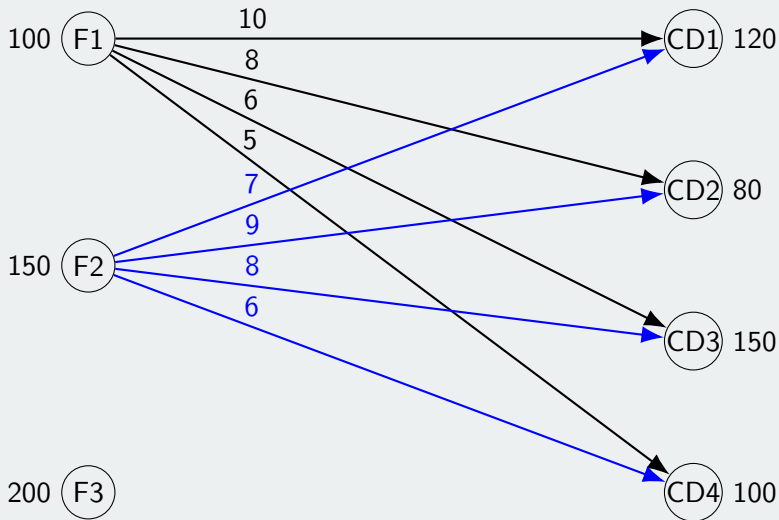
(CD3) 150

(CD4) 100

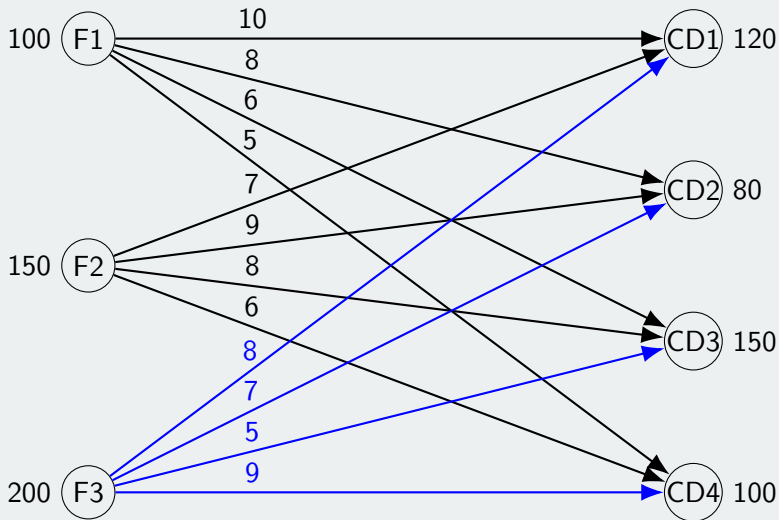
Representación Gráfica



Representación Gráfica



Representación Gráfica



Formulación del Problema de Transporte

Variables de decisión:

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

Formulación del Problema de Transporte

Variables de decisión:

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

Función objetivo:

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

Formulación del Problema de Transporte

Variables de decisión:

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

Función objetivo:

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

Restricciones:

$$\sum_{j=1}^4 x_{ij} = \text{capacidad}_i \quad \forall i$$

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

$$x_{ij} \geq 0 \quad \forall i, j$$

Implementación en PICOS

```
1 import picos
2 import numpy as np
3
4 # Crear el problema
5 P = picos.Problem()
6
7 # Definir variables
8 x = picos.RealVariable('x', 12)
9
10 # Definir costos
11 c = np.array([10, 8, 6, 5, 7, 9, 8, 6, 8, 7, 5, 9])
12
13 # Funcion objetivo
14 P.set_objective('min', picos.sum([c[i] * x[i] for i in range(12)]))
15
16 # Restricciones de capacidad
17 P.add_constraint(sum(x[0:4]) == 100) # F1
18 P.add_constraint(sum(x[4:8]) == 150) # F2
19 P.add_constraint(sum(x[8:12]) == 200) # F3
20
21 # Restricciones de demanda
22 P.add_constraint(x[0] + x[4] + x[8] == 120) # CD1
23 P.add_constraint(x[1] + x[5] + x[9] == 80) # CD2
24 P.add_constraint(x[2] + x[6] + x[10] == 150) # CD3
25 P.add_constraint(x[3] + x[7] + x[11] == 100) # CD4
26
27 P.solve(solver='glpk')
```

¿Qué pasa cuando oferta \neq demanda?

- **Exceso de oferta:** Introducir destino ficticio

¿Qué pasa cuando oferta \neq demanda?

- **Exceso de oferta:** Introducir destino ficticio
- **Exceso de demanda:** Introducir origen ficticio

¿Qué pasa cuando oferta \neq demanda?

- **Exceso de oferta:** Introducir destino ficticio
- **Exceso de demanda:** Introducir origen ficticio
- **Costo de penalización:** Valor M muy grande

¿Qué pasa cuando oferta \neq demanda?

- **Exceso de oferta:** Introducir destino ficticio
- **Exceso de demanda:** Introducir origen ficticio
- **Costo de penalización:** Valor M muy grande

Origen/Destino ficticio:

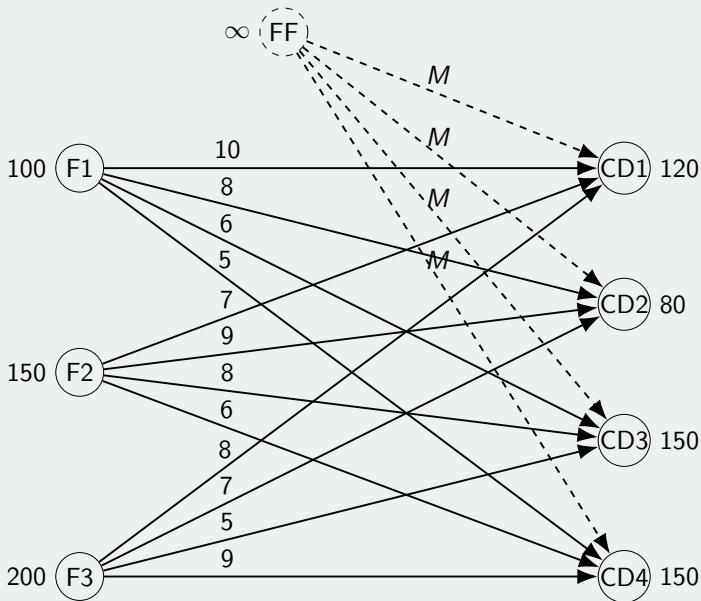
- Capacidad/demanda = $|\text{oferta} - \text{demanda}|$
- Costo = M (penalización)
- Representa demanda insatisfecha o capacidad ociosa

Ejemplo: Demanda Insatisfecha

Situación desbalanceada:

- Oferta total: $100 + 150 + 200 = 450$ unidades
- Demanda total: $120 + 80 + 150 + 150 = 500$ unidades
- Déficit: 50 unidades

Gráfico Demanda Insatisfecha



Función Objetivo con Penalización

Función objetivo original:

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

Función objetivo con fábrica ficticia:

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

Donde:


- M es un número muy grande (ej: 1000)
- $x_{FF,j}$ es la demanda insatisfecha del centro j
- La penalización hace que se evite la demanda insatisfecha

Problema de Transshipment


Extensión del problema de transporte:

- Permite nodos intermedios
- Los nodos pueden recibir y enviar
- Más flexible que transporte puro

Características adicionales:

 **Nodos de tránsito:** Hubs, puertos, almacenes

 **Restricciones de balance:** $\text{Entrada} = \text{Salida}$

 **Múltiples rutas:** Caminos alternativos

Ejemplo: Sunco Oil Co.

Situación:

- 2 pozos (W1: 150k, W2: 200k barriles/día)
- 2 ciudades (NY: 140k, LA: 160k barriles/día)
- 2 puertos intermedios (Mobile, Galveston)
- Costos variables por ruta

Objetivo:

- Minimizar costo total de transporte
- Satisfacer demanda de ambas ciudades
- Usar puertos como nodos de tránsito

Red de Transshipment: Nodos

150 (W1)

(NY) 140

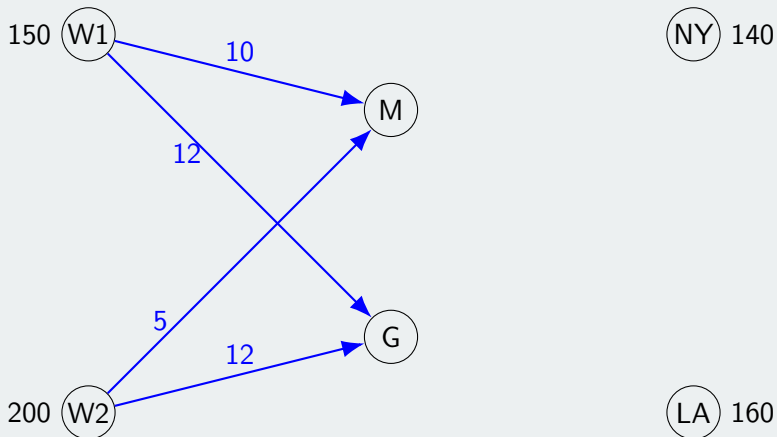
(M)

(G)

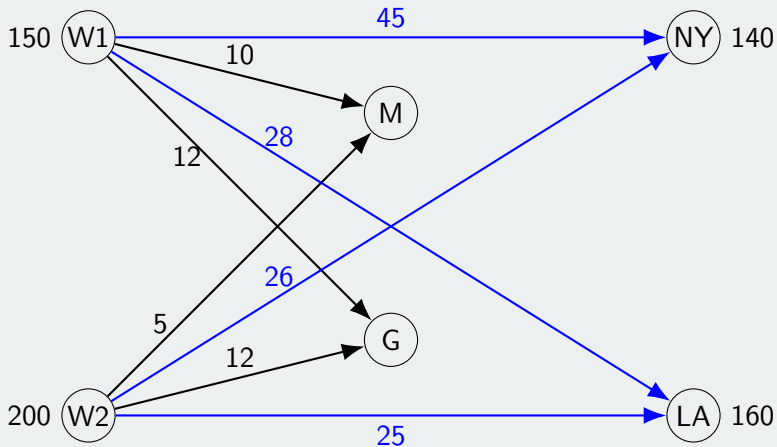
200 (W2)

(LA) 160

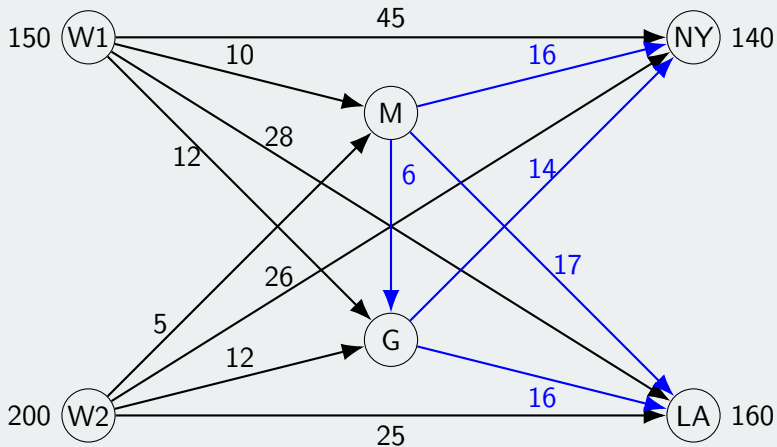
Red de Transshipment



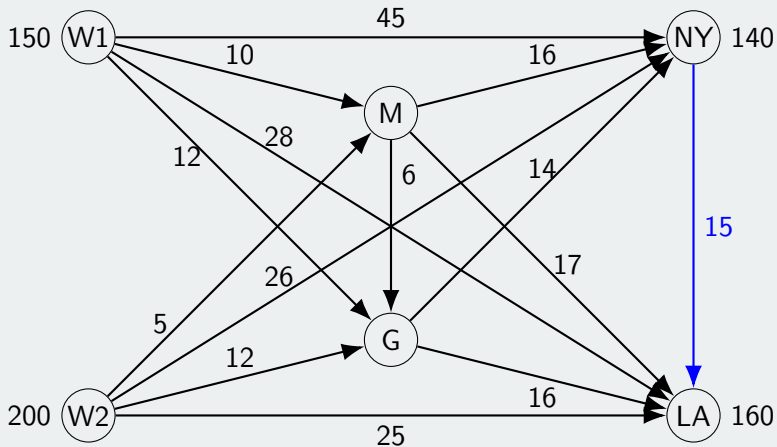
Red de Transshipment



Red de Transshipment



Red de Transshipment



Formulación Transshipment

Variables: x_{ij} = flujo desde nodo i a nodo j

Restricciones de balance:

$$\text{Entrada} - \text{Salida} = \text{Oferta neta}$$

Para cada nodo:

- **Pozos:** $\text{Salida} \leq \text{Capacidad}$
- **Ciudades:** $\text{Entrada} = \text{Demanda}$
- **Puertos:** $\text{Entrada} = \text{Salida}$ (balance)

Problema de Flujo Máximo

Objetivo: Maximizar flujo desde fuente hasta sumidero

Características:

- Un nodo fuente (source)
- Un nodo sumidero (sink)
- Capacidades en los arcos
- Conservación de flujo en nodos intermedios

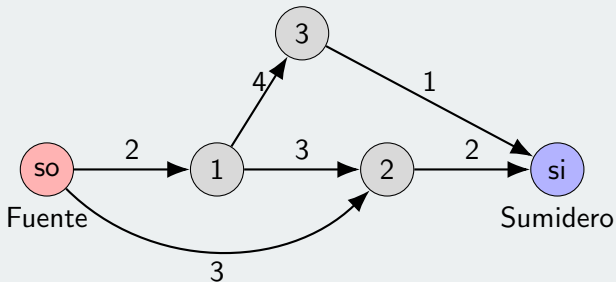
Aplicaciones:

≡ Redes de agua/gas

📶 Tráfico de red

🚗 Flujo vehicular

Ejemplo: Red de Tuberías



Formulación Flujo Máximo

Variables: x_i = flujo por arco i

Función objetivo:

Maximizar: Flujo total de salida desde fuente

Restricciones:

- **Capacidad:** $x_i \leq \text{capacidad}_i$ para cada arco
- **Conservación:** Entrada = Salida en nodos intermedios
- **Balance:** Salida fuente = Entrada sumidero

Implementación Flujo Máximo

```
1 import picos
2
3 # Crear el problema
4 P = picos.Problem()
5
6 # Definir variables
7 x = picos.RealVariable('x', 6)
8
9 # Funcion objetivo
10 P.set_objective('max', x[0] + x[1])
11
12 # Restricciones de conservacion
13 P.add_constraint(x[0] + x[1] == x[4] + x[5]) # salida = llegada
14 P.add_constraint(x[0] + x[1] == x[2] + x[3]) # nodo 1
15 P.add_constraint(x[1] + x[3] == x[5]) # nodo 2
16 P.add_constraint(x[2] == x[4]) # nodo 3
17
18 # Restricciones de capacidad
19 P.add_constraint(x[0] <= 2) # so-1
20 P.add_constraint(x[1] <= 3) # so-2
21 P.add_constraint(x[2] <= 3) # 1-2
22 P.add_constraint(x[3] <= 4) # 1-3
23 P.add_constraint(x[4] <= 1) # 3-si
24 P.add_constraint(x[5] <= 2) # 2-si
25
26 P.solve(solver='glpk') # Solucion: 2 unidades
```


Comparación de Problemas de Redes

Problema	Objetivo	Restricciones	Aplicación
Transporte	Min costo	Oferta/Demanda	Logística
Transshipment	Min costo	Balance + O/D	Distribución
Flujo Máximo	Max flujo	Capacidad	Redes

Elementos comunes:

- Variables de flujo
- Restricciones de balance/conservación
- Representación gráfica
- Algoritmos especializados

Para practicar:

- Resolver problemas de transporte balanceado y desbalanceado
- Experimentar con diferentes configuraciones de red
- Analizar cuellos de botella en flujo máximo
- Implementar soluciones en PICOS

Siguiente clase:

- Programación Entera

¿Dudas?
¿Consultas?



Universidad de
SanAndrés