Programación Entera: Redes de Flujos *

Sergio García Prado

sergio@garciparedes.me

4 de marzo de 2018

1. Introducción

[TODO]

2. Planteamiento: Programación Lineal

[TODO]

$$
\begin{aligned}
\text{Minimizar} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{w=1}^{l}\sum_{k=1}^{p} x_{ijwk} \cdot c_{ijw} & \\
\text{sujeto a} \quad & \sum_{j=1}^{m}\sum_{w=1}^{l} x_{ijwk} \leq c_{ik}, & \forall i \in \{1,...,n\}, \forall k \in \{1,...,p\} \\
& \sum_{i=1}^{n}\sum_{w=1}^{l} x_{ijwk} \leq d_{jk}, & \forall j \in \{1,...,m\}, \forall k \in \{1,...,p\} \\
& \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{p} x_{ijwk} \leq u_{w}, & \forall w \in \{1,...,l\} \\
& x_{ijwk} \geq 0, & \forall i \in \{1,...,n\}, \forall j \in \{1,...,m\}, \forall w \in \{1,...,l\}, \forall k \in \{1,...,p\}
\end{aligned}
\tag{1}
$$

Ecuación 1: Formulación como *Problema de Programación Lineal.*

3. Planteamiento: Transporte en 2 Etapas

[TODO]

---

* URL: `https://github.com/garciparedes/network-flow-transeuro`

$$
\begin{aligned}
\text{Minimizar} \quad & \sum_{i=1}^{n}\sum_{w=1}^{l}\sum_{k=1}^{p} x^1_{iwk}\cdot c^1_{iw} + \sum_{w=1}^{l}\sum_{j=1}^{m}\sum_{k=1}^{p} x^2_{wjk}\cdot c^2_{wj} \\
\text{sujeto a} \quad & \sum_{w=1}^{l} x^1_{iwk} \le c_{ik}, && \forall i \in \{1,...,n\}, \forall k \in \{1,...,p\} \\
& \sum_{i=1}^{n} x^1_{iwk} = \sum_{j=1}^{m} x^2_{wjk}, && \forall w \in \{1,...,l\}, \forall k \in \{1,...,p\} \\
& \sum_{w=1}^{l} x^2_{wjk} \ge d_{jw}, && \forall j \in \{1,...,m\}, \forall k \in \{1,...,p\} \\
& \sum_{i=1}^{n}\sum_{k=1}^{p} x^1_{iwk} \le u_w, && \forall w \in \{1,...,l\} \\
& x^1_{iwk} \ge 0, && \forall i \in \{1,...,n\}, \forall w \in \{1,...,l\}, \forall k \in \{1,...,p\} \\
& x^2_{wjk} \ge 0, && \forall w \in \{1,...,l\}, \forall j \in \{1,...,m\} \forall k \in \{1,...,p\}
\end{aligned}
\tag{2}
$$

Ecuación 2: Formulación como *Problema de Transporte en 2 Etapas.*

4. Planteamiento: Flujo de Redes

[TODO]

$$
\begin{aligned}
\text{Minimizar} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{p} x^1_{ijk}\cdot c_{ij} \\
\text{sujeto a} \quad & \sum_{j=1}^{n} x^1_{ijk} - \sum_{j=1}^{n} x^1_{jik} = d_{ik}, \forall i \in \{1,...,n\}, \forall k \in \{1,...,p\} \\
& \sum_{j=1}^{n}\sum_{k=1}^{p} x_{ijk} \le u_i, \qquad \forall i \in \{1,...,l\} \cap u_i > 0 \\
& x_{ijk} \ge 0, \qquad \forall i \in \{1,...,n\}, \forall j \in \{1,...,n\} \forall k \in \{1,...,p\}
\end{aligned}
\tag{3}
$$

Ecuación 3: Formulación como *Problema Flujo de Redes.*

5. Conclusiones

[TODO]

Apéndice A Código Fuente: Programación Lineal

```
model "linear-programming-transeuro"
    uses "mmxprs";

    declarations
        n: integer
        p: integer
    end-declarations

    initializations from "data.dat"
        n p
    end-initializations
```

```
declarations
    vertices = 1..n
    products = 1..p

    demand: array(vertices, products) of real
    cost: array(vertices, vertices) of real
    capacity: array(vertices) of real
end-declarations

initializations from "data.dat"
    demand
    cost
    capacity
end-initializations

n_origins := sum(v in vertices | sum(k in products) demand(v, k) > 0) 1
n_destinations := sum(v in vertices | sum(k in products) demand(v, k) < 0 ) 1
n_warehouses := sum(v in vertices | sum(k in products) demand(v, k) = 0 ) 1

declarations
 origins = 1..n_origins
 destinations = 1..n_destinations
 warehouses = 1..n_warehouses

 c: array(origins, products) of real
 d: array(destinations, products) of real
 cap: array(warehouses) of real

 cst: array(origins, destinations, warehouses) of real

    x: array(vertices, vertices, vertices, products) of mpvar
end-declarations


forall(k in products) do
 i_origins := 1
 i_destinations := 1
 i_warehouses := 1
 forall(v in vertices) do
     if(demand(v, k) > 0) then
         c(i_origins, k) := demand(v, k)
         i_origins := i_origins + 1;
     elif(demand(v, k) < 0) then
         d(i_destinations, k) := - demand(v, k)
         i_destinations := i_destinations + 1
     end-if
 end-do
end-do


k1:=1
i_origins := 1
forall(i in vertices) do
 j_destinations := 1
```

```
 origins_change := FALSE
    forall(j in vertices) do
     w_warehouses := 1
     destinations_change := FALSE
     forall(w in vertices) do
         if(demand(i, k1) > 0 and demand(j, k1) < 0 and demand(w, k1) = 0) then
             cst(i_origins, j_destinations, w_warehouses) := cost(i, w) + cost(w, j)
             origins_change := TRUE
             destinations_change := TRUE
             w_warehouses := w_warehouses + 1
         end-if
     end-do
     if (destinations_change = TRUE) then
            j_destinations := j_destinations + 1
        end-if
    end-do
    if (origins_change = TRUE) then
        i_origins := i_origins + 1
    end-if
end-do

i_warehouses := 1
forall(v in vertices | capacity(v) > 0) do
    cap(i_warehouses) := capacity(v)
    i_warehouses := i_warehouses + 1
end-do

!------------------------------------------------------------------------------------------
! Model
!------------------------------------------------------------------------------------------


forall(i in origins, k in products) do
    res_in(i, k) := sum(j in destinations, w in warehouses) x(i, j, w, k) <= c(i, k)
end-do

forall(w in warehouses) do
    res_max(w) := sum(i in origins, j in destinations, k in products) x(i, j, w, k) <= cap(w)
end-do

forall(j in destinations, k in products) do
    res_out(j, k) := sum(i in origins, w in warehouses) x(i, j, w, k) >= d(j, k)
end-do

objetive := sum(i in origins, j in destinations, w in warehouses, k in products) x(i, j, w, k)*(cst(i, j

minimize(objetive)

!------------------------------------------------------------------------------------------


forall(k in products) do
    writeln
    writeln("k = ", k)
    forall(i in origins) do
```

```
            writeln
            writeln("\ti = ", i)
            write("\t\t\t")
            forall(j in destinations) do
               write(j, "\t")
            end-do
            forall(w in warehouses) do
                writeln
                write("\t\t",w, "\t")
                forall(j in destinations) do
                    write(getsol(x(i, j, w, k)), "\t")
                end-do
            end-do
            writeln
        end-do
        writeln
    end-do
    writeln


    writeln("objetive = ", getobjval)

end-model
```

Apéndice B Código Fuente: Transporte en 2 Etapas

```
model "2-steps-transportation-transeuro"
    uses "mmxprs";

    declarations
        n: integer
        p: integer
    end-declarations

    initializations from "data.dat"
        n p
    end-initializations

    declarations
        vertices = 1..n
        products = 1..p

        demand: array(vertices, products) of real
        cost: array(vertices, vertices) of real
        capacity: array(vertices) of real
    end-declarations

    initializations from "data.dat"
        demand
        cost
        capacity
    end-initializations

    n_origins := sum(v in vertices | sum(k in products) demand(v, k) > 0) 1
    n_destinations := sum(v in vertices | sum(k in products) demand(v, k) < 0 ) 1
```

```
n_warehouses := sum(v in vertices | sum(k in products) demand(v, k) = 0 ) 1

declarations
 origins = 1..n_origins
 destinations = 1..n_destinations
 warehouses = 1..n_warehouses

 c: array(origins, products) of real
 d: array(destinations, products) of real
 cap: array(warehouses) of real

 cost_1: array(origins, warehouses) of real
 cost_2: array(warehouses, destinations) of real

 x_1: array(origins, warehouses, products) of mpvar
 x_2: array(warehouses, destinations, products) of mpvar
end-declarations

forall(k in products) do
 i_origins := 1
 i_destinations := 1
 i_warehouses := 1
 forall(v in vertices) do
     if(demand(v, k) > 0) then
         c(i_origins, k) := demand(v, k)
         i_origins := i_origins + 1;
     elif(demand(v, k) < 0) then
         d(i_destinations, k) := - demand(v, k)
         i_destinations := i_destinations + 1
     end-if
 end-do
end-do


k1:=1
i_origins := 1
i_destinations := 1
forall(i in vertices) do
 j_destinations := 1
 j_warehouses := 1
 origins_change := FALSE
 warehouses_change := FALSE
    forall(j in vertices) do
     if(demand(i, k1) > 0 and demand(j, k1) = 0) then
         cost_1(i_origins, j_warehouses) := cost(i, j)
         j_warehouses := j_warehouses + 1
         origins_change := TRUE
      elif(demand(i, k1) = 0 and demand(j, k1) < 0) then
         cost_2(i_warehouses, j_destinations) := cost(i, j)
            j_destinations := j_destinations + 1
            warehouse_change := TRUE
     end-if
    end-do
    if (origins_change = TRUE) then
        i_origins := i_origins + 1
```

```
    end-if
    if (warehouse_change = TRUE) then
        i_warehouses := i_warehouses + 1
    end-if
end-do

i_warehouses := 1
forall(v in vertices | capacity(v) > 0) do
    cap(i_warehouses) := capacity(v)
    i_warehouses := i_warehouses + 1
end-do


!-------------------------------------------------------------------------------------------
! Model
!-------------------------------------------------------------------------------------------

forall(i in origins, k in products) do
 res_in(i, k) := (sum(w in warehouses) x_1(i, w, k)) <= c(i, k)
end-do


forall(w in warehouses, k in products) do
    res_union(w, k) := sum(i in origins) x_1(i, w, k) = sum(j in destinations) x_2(w, j, k)
end-do

forall(w in warehouses) do
    res_max(w) := sum(i in origins, k in products) x_1(i, w, k) <= cap(w)
end-do

forall(j in destinations, k in products) do
 res_out(j, k) := sum(w in warehouses) x_2(w, j, k) >= d(j, k)
end-do


objetive := sum(i in origins, w in warehouses, k in products) x_1(i, w, k) * cost_1(i, w) +
                sum(w in warehouses, j in destinations, k in products) x_2(w, j, k) * cost_2(w, j)

minimize(objetive)

!-------------------------------------------------------------------------------------------


forall(k in products) do
    writeln
    writeln("k = ", k)
    write("\t")
    forall(j in warehouses) do
       write(j, "\t")
    end-do
    forall(i in origins) do
        writeln
        write(i, "\t")
        forall(j in warehouses) do
            write(getsol(x_1(i, j, k)), "\t")
```

```
            end-do
        end-do
        writeln
        writeln
        write("\t")
        forall(j in destinations) do
           write(j, "\t")
        end-do
        forall(i in warehouses) do
            writeln
            write(i, "\t")
            forall(j in destinations) do
                write(getsol(x_2(i, j, k)), "\t")
            end-do
        end-do
        writeln
    end-do
    writeln

    writeln("objetive = ", getobjval)

end-model
```

Apéndice C Código Fuente: Flujo de Redes

```
model "network-flow-transeuro"
    uses "mmxprs";

    declarations
        n: integer
        p: integer
    end-declarations

    initializations from "data.dat"
        n p
    end-initializations

    declarations
        vertices = 1..(n + 1)
        products = 1..p
        demand: array(vertices, products) of real
        cost: array(vertices, vertices) of real
        capacity: array(vertices) of real
        x: dynamic array(vertices, vertices, products) of mpvar
    end-declarations

    initializations from "data.dat"
        demand cost capacity
    end-initializations

    forall(i in vertices, j in vertices, k in products | cost(i, j) <> 0 and i <= n and j <= n) do
        create(x(i, j, k))
    end-do

    forall(k in products) do
```

```
        d_k := sum(i in vertices) demand(i, k)
        if (d_k > 0) then
            forall(i in vertices | demand(i, k) > 0) do
                create(x(i, n + 1, k))
            end-do
            demand(n + 1, k) := - d_k
        end-if
    end-do

    !---------------------------------------------------------------------------------------
    ! Model
    !---------------------------------------------------------------------------------------

    forall(i in vertices, k in products) do
        res_network(i, k) := sum(j in vertices) x(i, j, k) - sum(j in vertices) x(j, i, k) = demand(i, k)
    end-do

    forall(i in vertices | capacity(i) > 0) do
        res_capacity(i) := (sum(j in vertices, k in products) x(i, j, k)) <= capacity(i)
    end-do

    objetive := sum(i in vertices, j in vertices, k in products) x(i, j, k) * cost(i, j)

    minimize(objetive)

    !---------------------------------------------------------------------------------------

    forall(k in products) do
        writeln
        writeln("k = ", k)
        write("\t")
        forall(j in vertices) do
           write(j, "\t")
        end-do
        forall(i in vertices) do
            writeln
            write(i, "\t")
            forall(j in vertices) do
                write(getsol(x(i, j, k)), "\t")
            end-do
        end-do
        writeln
    end-do

    writeln
    writeln("objetive = ", getobjval)

end-model
```

Apéndice D Datos

```
n: 7
p: 2

demand:[(1 1) 420  (1 2) 200
```

```
        (2 1) 315  (2 2) 200
        (5 1) -120 (5 2) -90
        (6 1) -310 (6 2) -140
        (7 1) -180 (7 2) -122]

cost:[(1 3) 11 (1 4) 14
      (2 3) 12 (2 4) 13
      (3 5) 21 (3 6) 35 (3 7) 19
      (4 5) 18 (4 6) 29 (4 7) 15]


capacity:[(3) 600
          (4) 600]
```

Referencias

[FIC] FICO Xpress. Xpress-Mosel. `http://www.maths.ed.ac.uk/hall/Xpress/FICO_Docs/mosel/mosel_lang/dhtml/moselref.html/`.

[GP18] Sergio García Prado. Network Flow Transeuro, 2018. `https://github.com/garciparedes/network-flow-transeuro`.

[SA18] Jesús Sáez Aguado. Programación Entera, 2017/18. Facultad de Ciencias: Departamento de Estadística e Investigación Operativa.