Programación Entera: Redes de Flujos [*]

Sergio García Prado
sergio@garciparedes.me

5 de marzo de 2018

Resumen

Los problemas de abastecimiento de producto son un problema constante para nuestra sociedad. Es por ello que la modelización adecuada de los mismos para su estudio y resolución de manera eficiente debe ser estudiada en detalle. Por tanto, en este trabajo se analiza la modelización del problema multiproducto de suministrar existencias a destinos con determinadas necesidades a partir de orígenes con determinadas demandas pasando por puntos intermedios con restricciones de capacidad también determinadas. Dicha problemática se estudia a mediante la modelización de *Programación Lineal*, *Transporte en 2 Etapas* y *Flujo de Redes*.

1. Introducción

En este documento se va a estudiar el planteamiento del problema del transporte de mercancías desde un conjunto de orígenes hacia un conjunto de destinos, los cuales deben visitar un almacen intermedio. En este caso, el contexto del problema es el siguiente: *La empresa transeuro quiere encontrar la solución óptima para transportar sus mercancías de dos tipos* $\{p_1, p_2\}$ *desde los puntos de origen* $I = \{i_1, i_2\}$ *hacia los puntos de destino* $J = \{j_1, j_2, j_3\}$ *teniendo que atravesar obligatoriamente uno de los puntos intemedio* $W = \{w_1, w_2\}$*. En el problema, el conjunto* I *se refiere a puertos de origen,el conjunto* W *a almacenes intermedio y el conjunto* J *a ciudades de destino. Cada puerto de origen tiene una determinada cota de producción por producto (que denotaremos por* s_{ik}*), y cada destino una determinada demanda por producto (que denotaremos por* d_{jk}*). Además, cada almacen impone una restricción de espacio total (que denotamos por* u_w*). En cuanto a los costes de transporte, este tan solo depende de la ruta y se mantiene constante para el producto, por lo que lo denotaremos por* c_{ij}. (Los valores concretos utilizados pueden ser consultados en el Apéndice B).

Puesto que el objetivo de este trabajo reside en la comparación de las distintas modelizaciones posibles para resolver el problema, esta notación variará en ciertas ocasiones para adaptarse a las restricciones específicas de cada problema. Para tratar de facilitar al máximo la comparación entre modelos por parte del usuario, se ha decidido utilizar tan solo una representación en el conjunto de datos (sin que este tenga que tener en cuenta el balanceo del problema cuando el modelo lo requiere). A costa de esto, se ha añadido cierta complejidad en la lectura de datos, que requiere de la adaptación concreta hacia la entrada necesaria para cada modelo.

El conjunto de datos puede visualizarse mediante su representación en forma de grafo en la figura 1, en la cual se incluyen los vértices o puntos que representan, orígenes, almacenes y destinos, así como la oferta y demanda de cada uno de ellos, junto con las restricciones de capacidad y los costes de cada arista.

El resto del documento se organiza de la siguiente manera: se describen los tres modelos estudiados (*Programación Lineal* en la sección 2, *Transporte en 2 Etapas* en la sección 3 y *Flujo de Redes* en la sección 4) en las cuales se hace especial incapié en las dificultades de implementación así como el coste computacional y las restricciones propias del modelo en caso de ser necesarias. Para finalizar, en la sección 5 se concluye el trabajo con una comparación global entre las tres estrategias de modelización.

2. Planteamiento: Programación Lineal

El modelo de programación lineal es el más sencillo de entender a nivel conceptual de entre los que se estudian en este trabajo. Este modelo se muestra en la ecuación (1). Se basa en un conjunto de variables indexadas en 4 índices (referidos a origen, destino, almacén y producto), indicando cada variable la cantidad de producto que se envía por cada camino posible. Para ello, tan solo es necesario la cantidad máxima de producto en el origen (mediante la primera restricción), el cubrimiento de unidades de producto en los destinos (mediante la segunda restricción) y las limitaciones de producto total por almacén (mediante la tercera restricción). Es trivial entender la última restricción de no negatividad.

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

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

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

Nótese que en este caso no es necesario balancear el problema, puesto que no estamos imponiendo ninguna restricción de igualdad que "obligue" a transportar todo el producto. Nos podemos permitir dicha ventaja porque en este modelo no existe la posibilidad de que quede mercancia por los puntos intermedios del camino (algo que no siempre sucederá como veremos en el modelo de *Flujo de Redes* de la sección 4).

En cuanto al número de variables de decisión necesarias para resolver el problema, en este caso necesitamos $n(I)\cdot n(J)\cdot n(W)\cdot n(P) = 2\cdot 3\cdot 2\cdot 2 = 24$. A nivel de restricciones, tenemos $n(I)\cdot n(P) + n(W)\cdot n(P) + n(J)\cdot n(P) + \cdot n(W) + n(I)\cdot n(J)\cdot n(W)\cdot n(P) = 2\cdot 2 + 2\cdot 2 + 3\cdot 2 + 2\cdot 3\cdot 2\cdot 2 = 38$. En la implementación realizada cuyo código fuente puede ser consultado en el Apéndice A.1ăse ha encontrado la solución óptima en 11 iteracciónes.

La solución obtenida mediante la modelización como un problema de programación lineal se muestra en la figura 2.

3. Planteamiento: Transporte en 2 Etapas

Con el modelo de transporte se posee la ventaja de conocer algoritmos muy eficientes para su resolución. Sin embargo, estos están planteados para problemas "directos", es decir, aquellos en los cuales tan solo se tiene orígenes, destinos y un único producto. Por lo tanto, para modelizar nuestro problema de esta manera debemos realizr una serie de modificaciones. Este modelo se muestra en la ecuación (2).

La primera de ellas se basa en el desdoble del problema en dos etapas, para así obtener dos problemas que cumplirían las especificaciones del modelo de transporte si se desdoblan por productos. Por tanto, utilizaremos dos conjuntos de variables indexadas en tres índices y denotadas como x^1_{iwk} para representar la cantidad de producto k del origen i al almacén w y x^2_{wjk} para representar la cantidad de producto k del almacén w al destino j.

Al igual que en el caso anterior, es necesario imponer la restricción máxima de producción sobre los orígenes (mediante la primera restricción), así como la "conexión" de los dos conjuntos de variables en los almacenes (mediante la segunda restricción). Lo siguiente es imponer las restricciones de demanda sobre los origines (tal y como se puede ver en la tercera restricción). Por último, es necesario añadir la restricción de capacidad, que en este caso se podría realizar sobre cualquiera de los dos conjuntos de variables (por estar en los almacenes), pero parece más razonable incluirla sobre el primer conjunto (tal y como se ha hecho en la cuarta restricción). Es trivial entender las dos últimas restricciones de no negatividad.

En este caso tampoco es necesario balancear el problema siguiendo el mismo planteamiento que en el caso anterior, ya que por la relación de transitividad entre las dos etapas del problema de transporte, este hará los excedentes de producto queden en el origen.

En cuanto al número de variables de decisión necesarias para el planteaminento del modelo, en este caso han utilizado $n(I)\cdot n(W)\cdot n(P) + n(W)\cdot n(J)\cdot n(P) = 2\cdot 2\cdot 2 + 2\cdot 3\cdot 2 = 8 + 12 = 20$ junto con $n(I)\cdot n(P) + n(W)\cdot n(P) + n(J)\cdot n(P) + n(W) + n(I)\cdot n(W)\cdot n(P) + n(W)\cdot n(J)\cdot n(P) = 2\cdot 2 + 2\cdot 2 + 3\cdot 2 + 2 + 2\cdot 2\cdot 2 + 2\cdot 3\cdot 2 = 36$. En la implementación cuyo código fuente puede ser consultado en el Apéndice A.2 realizada se ha encontrado la solución óptima en 14 iteracciónes.

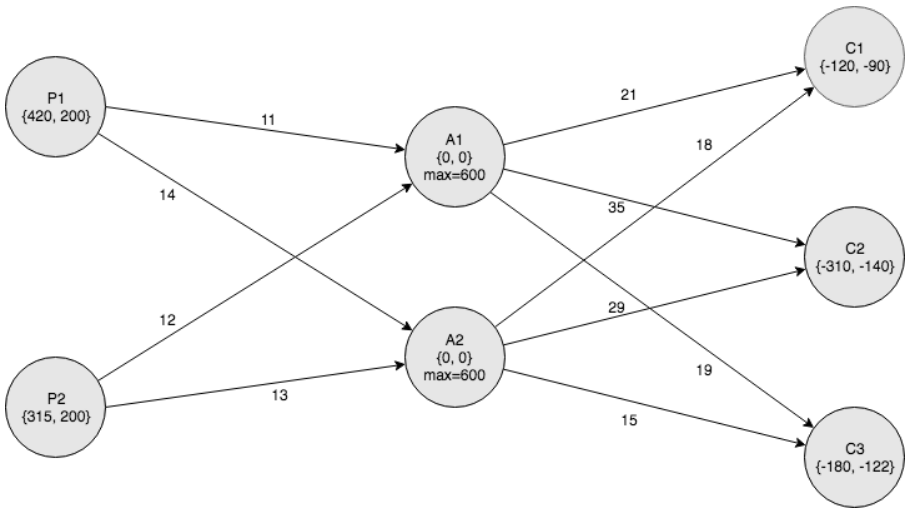

Figura 1: Representación gráfica del Conjunto de Datos

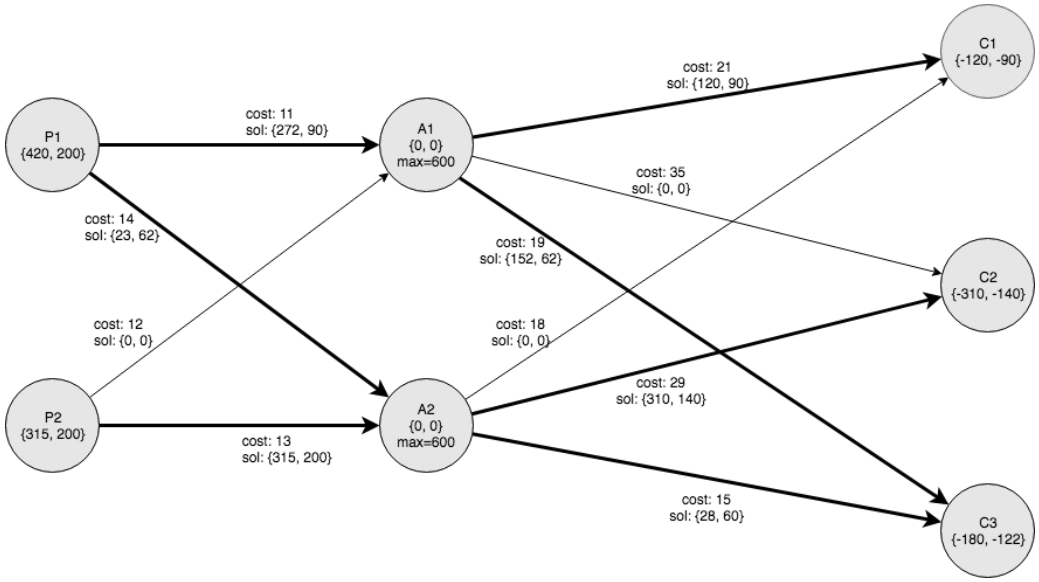

Figura 2: Representación gráfica de la solución obtenida mediante la modelización como problema de *Programación Lineal*

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

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

La solución obtenida mediante la modelización como un problema de transporte se muestra en la figura 3.

4. Planteamiento: Flujo de Redes

El último modelo que se analizará en este trabajo es el basado en flujo de redes. Este se diferencia de los anteriores porque no impone ninguna restricción con respecto a orígenes, destinos o puntos intermedios, sino que todos son equivalentes. Sin embargo, para que esta modelización sea apropiada el problema debe ser balanceado, ya que lo que se busca es el punto de equilibrio entre las necesidades de todos los vértices. Este modelo se muestra en la ecuación (3).

Por tanto, en este caso es necesario balancear el problema (que en el caso de la implementación actual se hace automáticamente). Sin embargo, lo haríamos de manera manual sería incluir un nuevo conjunto de aristas de coste cero conectadas a los vértices productores con demanda equivalente al opuesto del excedente. Es decir, $d_{81} = -125, d_{82} = -48$.

En cuanto a las restricciones necesarias para el modelo, puesto que lo que se busca es la estabilidad es necesario codificar la capacidad y demanda en un único vector al que denominaremos d siendo d_i la demanda (con signo negativo para los vértices consumidores), la oferta (con signo positivo para los vértices productores) y 0 en caso de que sean vértices de "paso". Lo siguiente es igualar la entrada menos la salida a este valor (tal y como se hace en la primera restricción). En cuanto a las capacidades, basta con restringir el valor de entrada (o salida) a la capacidad del máxima del vértice (como se indica en la segunda restricción). La última restricción de positividad es trivial.

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

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

En cuanto al número de variables de decisión necesarias para el planteaminento del modelo, en este caso han utilizado $(n(I)+n(I)+n(W)+1)\cdot n(P) = 8\cdot 2 = 16$ junto con $(n(I)+n(I)+n(W)+1)\cdot n(P) + (n(I)+n(I)+n(W)+1) + (n(I)+n(I)+n(W)+1)\cdot n(P) = 16+8+16 = 40$. En la implementación cuyo código fuente puede ser consultado en el Apéndice A.3 realizada se ha encontrado la solución óptima en 10 iteracciónes.

La solución obtenida mediante la modelización como un problema de flujo de redes se muestra en la figura 4.

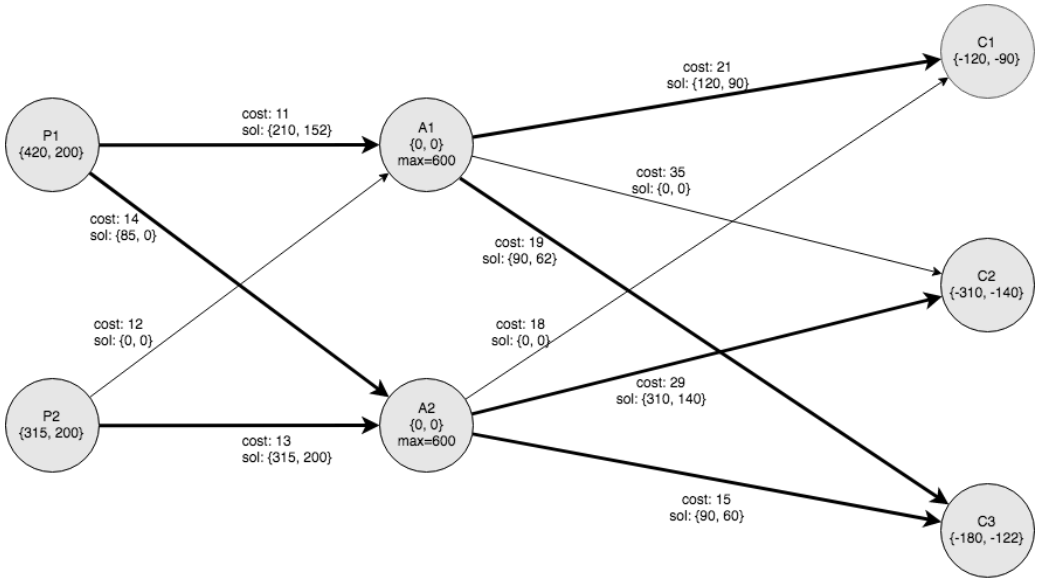

Figura 3: Representación gráfica de la solución obtenida mediante la modelización como problema de *Transporte en 2 Etapas*

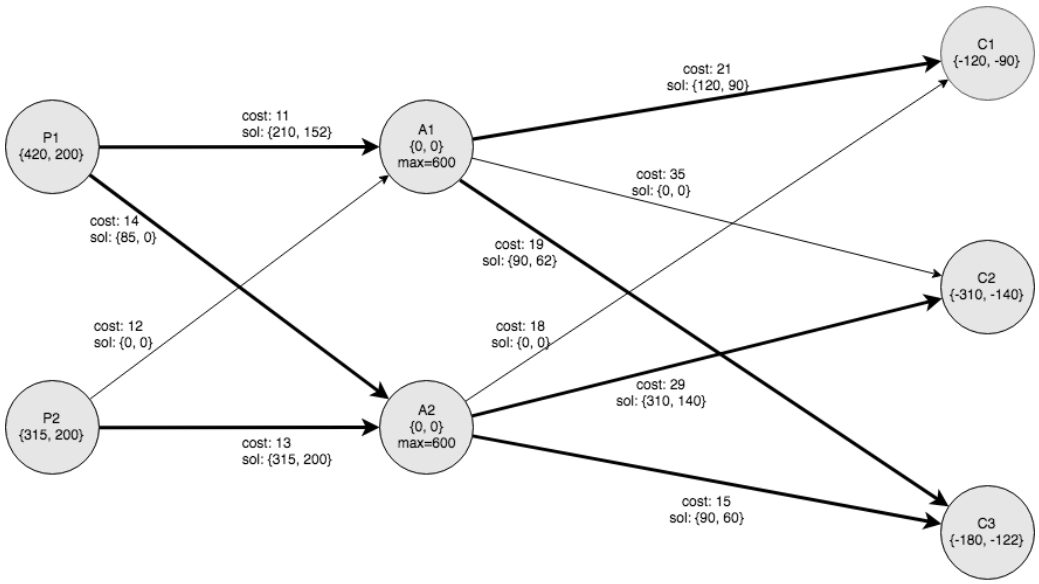

Figura 4: Representación gráfica de la solución obtenida mediante la modelización como problema de *Flujo de Redes*

5. Conclusiones

A partir de las tres modelizaciones se obtiene el resultado óptimo para la planificación del problema (coste de 34465). Sin embargo, esta solución no es única, ya tal y como se ha visto en los resultados, a pesar de que estos no son iguales para los tres modelos, el valor óptimo si que lo es. A nivel de complejidad computacional, el la modelización de flujos de redes parece el más eficiente de los tres pagando el precio de la necesidad de balanceo previo.

Sería interesante comparar dichas soluciones sobre un conjunto de datos de tamaño mayor para comprobar la eficiencia de cada modelización, así como su comparación con soluciones basadas en heurísticas.

Apéndice A Código Fuente

A.1 Programación Lineal

```
model "linear-programming-transeuro"

    !-------------------------------------------
    ! Linear Programming Model - Transeuro
    ! Sergio García Prado - garciparedes.me
    ! March 2018
    !-------------------------------------------

    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, w))

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
```

A.2 Transporte en 2 Etapas

```
model "2-steps-transportation-transeuro"


    !-----------------------------------------
    ! Two Steps Transportation Model - Transeuro
    ! Sergio García Prado - garciparedes.me
    ! March 2018
    !-----------------------------------------

    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
```

A.3 Flujo de Redes

```
model "network-flow-transeuro"

    !-------------------------------------------
    ! Network Flow Model - Transeuro
    ! Sergio García Prado - garciparedes.me
    ! March 2018
    !-------------------------------------------

    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 B 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.