Memoria del Trabajo "Problema del Transporte"

Matemática Aplicada a los Sistemas de Información – MASI 20/21

Universidad de Sevilla

Autores:

- Alejandro Fernández Trigo
- Juan Diego Villalobos Quirós

Índice de contenidos:

Descripción General

Los problemas de programación lineal (también llamados PPLs) son un tipo de problema matemático que busca optimizar (maximizar / minimizar) una determinada función lineal.

El problema que nos ocupa, el llamado “Problema del Transporte” (muy común en el ámbito matemático y sobre todo en el campo de la economía), es un caso específico de PPL que pretende minimizar los costes de ofrecer una determinada demanda a una serie de entidades (clientes, empresas, personas, etc.) dados una serie de ofertantes (empresas, almacenes, productores, etc.). Para ello, los problemas se modelan teniendo en cuenta:

- Variables de decisión
- Restricciones
  - Demandantes
  - Ofertantes
- Función objetivo

La finalidad de este trabajo será la de resolver un problema planteado del tipo anteriormente descrito, para lo cual partimos de un enunciado que modelaremos, describiremos y daremos una ‘solución inicial factible' mediante tres métodos distintos, a saber:

- North-west method o método esquina noreste.
- Minimal-cost method o least-cost method o método del mínimo coste.
- Vogel’s method o método de Vogel.

A continuación, estudiaremos si la solución inicial es degenerada o no, en cuyo caso, de ser no-degenerada procederemos a mejorarla mediante el método de mejora llamado “stepping stones”.

Por último, hallaremos una solución factible mediante un software matemático cómo SAGE Math v.9+ y, adicionalmente, añadiremos algunos algoritmos en Python para los métodos anteriormente citados.

Concluiremos el presente trabajo con algunas anotaciones al respecto así cómo añadiendo su adecuada documentación de referencia para el desarrollo de este.

Métodos de Resolución

Método de North-West Corner o Método de la esquina noreste

Método de programación lineal para encontrar una solución inicial factible a partir de un modelo ya balanceado; es uno de los métodos más sencillos para encontrar soluciones iniciales, debido a que ignora la magnitud relativa de los costes. Es una de las técnicas más rápidas, pero no garantiza una solución.

Pasos:

1. Seleccionar la esquina noreste, esto es, la esquina superior izquierda para comenzar.
2. Asignar la mayor capacidad posible a la celda seleccionada ajustando las cantidades de oferta/demanda restando la cantidad asignada.
3. Cuando la oferta/demanda = 0, cancelamos la fila/columna que ha quedado exhausta.
4. Cuando sólo queda una fila/columna, acabamos. De lo contrario, pasamos a la celda derecha (si hemos cancelado una columna) o a la celda de abajo (si hemos cancelado una fila).

Método de Least-Cost o Método del Mínimo Coste

Método de programación lineal para encontrar una solución inicial factible a partir de un modelo ya balanceado, parecido al método de la esquina noreste, pero que devuelve mejores resultados. Trata de asignar la mayor cantidad de unidades posibles (bajo restricciones) a la celda menos costosa.

Pasos:

1. De la matriz de variables de decisión elegir la celda menos costosa y a dicha celda, asignar la mayor capacidad posible, restringido por la oferta/demanda. Se ajusta la oferta/demanda de la fila restando la cantidad asignada.
2. Se elimina la fila/columna cuya oferta/demanda sea 0 tras el primer paso.
3. Si existen dos posibilidades, la primera en la que quede una sola fila o columna, se cancela esa. Si llegamos a este caso, hemos terminado. Si queda más de una fila/columna, volvemos al primer paso.

Método de Vogel o Vogel's Method

Método de programación lineal heurístico para encontrar una solución inicial factible a partir de un modelo ya balanceado. Es el método que mas se aleja de los dos anteriormente vistos. Precisa de más iteraciones con lo que es más lento pero sus resultados iniciales suelen ser mejores. Usa una serie de penalizaciones para calcular el mínimo coste.

Pasos:

1. A partir de la matriz de variables de decisión calcular el menor coste por fila/columna. Aquella fila/columna con mayor penalización será la de inicio. (Si hay valores que coinciden queda a discreción).
2. Partiendo de la fila/columna escogida, escoger la celda de menor coste y asignarle la mayor capacidad posible (restringido por la oferta/demanda). Podemos eliminar la fila/columna una vez exhausta la oferta/demanda.
3. Si solo queda una fila, paramos. En caso contrario volver al paso primero.

Resolución del Problema

6. A company has four warehouses and supplies four customers. The distances involved are small, and the company charges its customers in terms of the charges per unit involved in loading at the warehouses and unloading at the destinations according to the following table. Formulate the problem, and use the transportation algorithm to find an optimal solution. Is it unique? (*Hint*: Construct an initial basic feasible solution by the three methods.)

Warehouse	Unit loading charge	Available suppliers	Customer	Unit unloading charge	Requirements
1	1	20	1	1	15
2	2	20	2	2	35
3	2	30	3	3	15
4	4	10	4	4	10

El problema planteado consiste en minimizar el coste de transportar mercancías desde almacenes (warehouses) a clientes (customers). Se especifica una demanda (requirements) que hay que satisfacer y una cantidad disponible para cada suministrador. Cada ruta tiene un coste asociado, un coste de carga y descarga.

Planteamos el problema cómo un Problema de Programación Lineal:

INICIAL	**Cliente 1**	**Cliente 2**	**Cliente 3**	**Cliente 4**	Proveedores disponibles	
Almacén 1	*1+1*	*2+1*	*3+1*	*4+1*	<= 20	
Almacén 2	*1+2*	*2+2*	*3+2*	*4+2*	<= 20	
Almacén 3	*1+2*	*2+2*	*3+2*	*4+2*	<= 30	
Almacén 4	*1+3*	*2+3*	*3+3*	*4+3*	<= 10	
Requisitos	>= 15	>= 35	>= 15	>= 10	75 != 80 así que añado 5	

¿Está balanceado? El problema debe estar balanceado para aplicar cualquiera de los tres algoritmos anteriormente vistos. Cómo la suma de requisitos no es igual a la suma de proveedores, esto es, (15+35+15+10) = 75 != 80 = (20+20+30+10), sumamos 5, lo que implica añadir una “columna tonta” o “dummy column”. Esto balancea el problema para empezar.

El problema expresado como PPL balanceado quedaría:

PASO 0	**Cliente 1**	**Cliente 2**	**Cliente 3**	**Cliente 4**	Columna inservible	Proveedores disponibles
Almacén 1	*2*	*3*	*4*	*5*	0	<= 20
Almacén 2	*3*	*4*	*5*	*6*	0	<= 20
Almacén 3	*3*	*4*	*5*	*6*	0	<= 30
Almacén 4	*4*	*5*	*6*	*7*	0	<= 10
Requisitos	>= 15	>= 35	>= 15	>= 10	5	

, cuyos elementos fundamentales son:

- Variables de decisión.
- Restricciones.
- Función objetivo.

Variables de decisión

Las variables de decisión son aquellos términos que controlan el problema modelado en última instancia. Están controladas por las restricciones del problema. En nuestro caso particular, cómo TODAS son variables enteras, nuestro problema de programación lineal es un caso particular de problema de programación lineal entera.

Restricciones

Dos tipos:

- Restricciones de demanda → requisitos.
- Restricciones de oferta → proveedores.

Las restricciones representan las limitaciones de nuestro problema.

Función objetivo

Función lineal de varias variables sujeta a las restricciones del problema que representa la solución o valor objetivo. Esta función puede ser o bien de maximizar, o bien de minimizar (nuestro caso).

Resolución

Vamos a dar una solución inicial factible, para lo cual partimos del PPL que hemos balanceado antes.

Método North-West o esquina noreste

PASO 0							Instrucciones
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	
Almacén 1	2	3	4	5	0	<= 20	Comparar Almacen1 con Cliente1
Almacén 2	3	4	5	6	0	<= 20	El resultado actualiza la capacidad de almacén 1
Almacén 3	3	4	5	6	0	<= 30	El mínimo se carga en la posición consultada
Almacén 4	4	5	6	7	0	<= 10	min(20,15) = 15 --> 20 - 15 = 5
Requisitos	>= 15	>= 35	>= 15	>= 10	5		

PASO 1							Instrucciones
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	
Almacén 1	2 (15)	3	4	5	0	<= 5	La demanda de 1 queda satisfecha y se compara Almacen1 con Cliente2
Almacén 2	3	4	5	6	0	<= 20	El resultado actualiza la capacidad de almacén 2
Almacén 3	3	4	5	6	0	<= 30	El mínimo se carga en la posición consultada
Almacén 4	4	5	6	7	0	<= 10	min(5,35) = 5 --> 35 - 5 = 30
Requisitos	>= 0	>= 35	>= 15	>= 10	5		

PASO 0							Instrucciones
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	
Almacén 1	2	3	4	5	0	<= 20	Comparar Almacén1 con Cliente1
Almacén 2	3	4	5	6	0	<= 20	El resultado actualiza la capacidad de almacén 1
Almacén 3	3	4	5	6	0	<= 30	El mínimo se carga en la posición consultada
Almacén 4	4	5	6	7	0	<= 10	min(20,15) = 15 --> 20 - 15 = 5
Requisitos	>= 15	>= 35	>= 15	>= 10	5		

PASO 1							Instrucciones
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	
Almacén 1	2 (15)	3	4	5	0	<= 5	La demanda de 1 queda satisfecha y se compara Almacén1 con Cliente2
Almacén 2	3	4	5	6	0	<= 20	El resultado actualiza la capacidad de almacén 2
Almacén 3	3	4	5	6	0	<= 30	El mínimo se carga en la posición consultada
Almacén 4	4	5	6	7	0	<= 10	min(5,35) = 5 --> 35 - 5 = 30
Requisitos	>= 0	>= 35	>= 15	>= 10	5		

Seguimos iterando en base al mismo principio con lo que obtenemos las siguientes tablas:

PASO 2						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	2 (15)	3 (5)	4	5	0	<= 0
Almacén 2	3	4	5	6	0	<= 20
Almacén 3	3	4	5	6	0	<= 30
Almacén 4	4	5	6	7	0	<= 10
Requisitos	>= 0	>= 30	>= 15	>= 10	5	

PASO 3						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	5	0	0	0	<= 0
Almacén 2	0	20	0	0	0	<= 0
Almacén 3	0	4	5	6	0	<= 30
Almacén 4	0	5	6	7	0	<= 10
Requisitos	>= 0	>= 10	>= 15	>= 10	5	

PASO 4						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	5	0	0	0	<= 0
Almacén 2	0	20	0	0	0	<= 0
Almacén 3	0	10	5	6	0	<= 20
Almacén 4	0	0	6	7	0	<= 10
Requisitos	>= 0	>= 0	>= 15	>= 10	5	

PASO 5						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	5	0	0	0	<= 0
Almacén 2	0	20	0	0	0	<= 0
Almacén 3	0	10	15	6	0	<= 5
Almacén 4	0	0	0	7	0	<= 10
Requisitos	>= 0	>= 0	>= 0	>= 10	5	

PASO 6						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	5	0	0	0	<= 0
Almacén 2	0	20	0	0	0	<= 0
Almacén 3	0	10	15	5	0	<= 0
Almacén 4	0	0	0	7	0	<= 10
Requisitos	>= 0	>= 0	>= 0	>= 5	5	

PASO 7						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	5	0	0	0	<= 0
Almacén 2	0	20	0	0	0	<= 0
Almacén 3	0	10	15	5	0	<= 0
Almacén 4	0	0	0	5	0	<= 5
Requisitos	>= 0	>= 0	>= 0	>= 0	5	

Esto nos lleva a la solución inicial factible o viable que queda tal que así:

	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	2 (15)	3 (5)	4	5	0	<= 20
Almacén 2	3	4 (20)	5	6	0	<= 20
Almacén 3	3	4 (10)	5 (15)	6 (5)	0	<= 30
Almacén 4	4	5	6	7 (5)	0 (5)	<= 10
Requisitos	>= 15	>= 35	>= 15	>= 10	5	
Coste mínimo transporte: 2*15 + 3*5 + 4*20 + 4*10 + 5*15 + 6*5 + 7*5 + 0*5 = 305						

305 es el coste del problema, esto representa el valor objetivo. Luego veremos cómo obtener este valor mediante un software matemático apropiado como SAGE Math.

¿Es esta solución adecuada para continuar? Depende de si es degenerada o no-degenerada. Para saberlo tenemos que localizar las celdas “allocated” o “localizadas”. Si se cumple que nº de “allocated cells” = m + (n – 1), entonces tendríamos una solución no-degenerada (donde m es el nº de filas y n el nº de columnas).

Cómo 8 = 4 + (5 – 1) se verifica, estamos ante una solución inicial factible NO-DEGENERADA, lo que la hace apropiada para dar una mejora mediante el método de stepping-stones.

Veremos antes los otros dos métodos, pero adelantaremos ya, que devuelven soluciones degeneradas así que nos centraremos en la solución obtenida por North-West para mejorarla. Si todos los métodos fueran degenerados tendríamos delante un caso que precisa de un paso adicional.

¿Qué es este paso? En caso de una solución degenerada, si no podemos dar una solución no degenerada por ningún otro método, o si todos los demás métodos arrojan un coste mayor, podemos añadir un cero a cualquiera de las celdas vacías, pero deberíamos tener en cuenta dicho valor a la hora de mejorar la solución.

Método least-cost method o del mínimo coste

PASO 0							
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Instrucciones
Almacén 1	2	3	4	5	0	<= 20	min(20,5) = 5 --> 20 – 5 = 15
Almacén 2	3	4	5	6	0	<= 20	
Almacén 3	3	4	5	6	0	<= 30	
Almacén 4	4	5	6	7	0	<= 10	
Requisitos	>= 15	>= 35	>= 15	>= 10	5		

PASO 1						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	2	3	4	5	0 (5)	<= 15
Almacén 2	3	4	5	6	0	<= 20
Almacén 3	3	4	5	6	0	<= 30
Almacén 4	3	5	6	7	0	<= 10
Requisitos	>= 15	>= 35	>= 15	>= 10	0	

Seguimos iterando en base al mismo principio con lo que obtenemos las siguientes tablas:

PASO 2						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	0	0	0	5	<= 0
Almacén 2	0	4	5	6	0	<= 20
Almacén 3	0	4	5	6	0	<= 30
Almacén 4	0	5	6	7	0	<= 10
Requisitos	>= 0	>= 35	>= 15	>= 10	0	

PASO 3						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	0	0	0	5	<= 0
Almacén 2	0	4	5	6	0	<= 20
Almacén 3	0	4	5	6	0	<= 30
Almacén 4	0	5	6	7	0	<= 10
Requisitos	>= 0	>= 35	>= 15	>= 10	0	

PASO 4						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	0	0	0	5	<= 0
Almacén 2	0	4	5	6	0	<= 20
Almacén 3	0	30	0	0	0	<= 0
Almacén 4	0	5	6	7	0	<= 10
Requisitos	>= 0	>= 5	>= 15	>= 10	0	

PASO 5						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	0	0	0	5	<= 0
Almacén 2	0	5	5	6	0	<= 15
Almacén 3	0	30	0	0	0	<= 0
Almacén 4	0	0	6	7	0	<= 10
Requisitos	>= 0	>= 0	>= 15	>= 10	0	

PASO 6						
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	15	0	0	0	5	<= 0
Almacén 2	0	5	15	0	0	<= 0
Almacén 3	0	30	0	0	0	<= 0
Almacén 4	0	0	0	7	0	<= 10
Requisitos	>= 0	>= 0	>= 0	>= 10	0	

Esto nos lleva a la solución inicial factible o viable que queda tal que así:

	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	2 (15)	3	4	5	0 (5)	<= 20
Almacén 2	3	4 (5)	5 (15)	6	0	<= 20
Almacén 3	3	4 (30)	5	6	0	<= 30
Almacén 4	4	5	6	7 (10)	0	<= 10
Requisitos	>= 15	>= 35	>= 15	>= 10	5	
Coste mínimo transporte: 2*15 + 0*5 + 4*5 + 5*15 + 4*30 + 7*10 = 315						

Esta solución, comparada con la anterior, podemos ver que es peor dado que su coste (valor objetivo) es mayor; además, vamos a ver que es degenerada porque el nº de “allocated cells”, 6, es distinto a 4 + (5 – 1). Nos seguimos quedando con la solución devuelta por el método de North-West.

Vogel's Method o Método de Vogel

PASO 0							
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F
Almacén 1	2	3	4	5	0	<= 20	
Almacén 2	3	4	5	6	0	<= 20	
Almacén 3	3	4	5	6	0	<= 30	
Almacén 4	4	5	6	7	0	<= 10	
Requisitos	>= 15	>= 35	>= 15	>= 10	5		
Penalización C							

PASO 1								Instrucciones
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F	
Almacén 1	2	3	4	5	0	<= 20	2 - 0 = 2	
Almacén 2	3	4	5	6	0	<= 20	3 - 0 = 3	
Almacén3	3	4	5	6	0	<= 30	3 - 0 = 3	
Almacén 4	4	5	6	7	0	<= 10	4 - 0 = 4	← Máxima diferencia
Requisitos	>= 15	>= 35	>= 15	>= 10	5			
Penalización C	3 - 2 = 1	4 - 3 = 1	5 - 4 = 1	6 - 5 = 1	0 - 0 = 0		La resta de los menores	

PASO 1								Instrucciones
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F	
Almacén 1	2	3	4	5	0	<= 20	2 - 0 = 2	
Almacén 2	3	4	5	6	0	<= 20	3 - 0 = 3	
Almacén3	3	4	5	6	0	<= 30	3 - 0 = 3	
Almacén 4	4	5	6	7	0	<= 10	4 - 0 = 4	← Máxima diferencia
Requisitos	>= 15	>= 35	>= 15	>= 10	5			min(10,5) = 5 --> 10 - 5 = 5
Penalización C	3 - 2 = 1	4 - 3 = 1	5 - 4 = 1	6 - 5 = 1	0 - 0 = 0		La resta de los menores	

PASO 2								Instrucciones
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F	
Almacén 1	2	3	4	5	0	<= 20	3 - 2 = 1	
Almacén 2	3	4	5	6	0	<= 20	4 - 3 = 1	
Almacén 3	3	4	5	6	0	<= 30	4 - 3 = 1	
Almacén 4	4	5	6	7	0 (5)	<= 5	5 - 4 = 1	
Requisitos	>= 15	>= 35	>= 15	>= 10	0			
Penalización C	3 - 2 = 1	4 - 3 = 1	5 - 4 = 1	6 - 5 = 1	null		La resta de los menores	

Seguimos iterando en base al mismo principio con lo que obtenemos las siguientes tablas:

PASO 3							
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F
Almacén 1	2(15)	3	4	5	0	<= 5	4 - 3 = 1
Almacén 2	3	4	5	6	0	<= 20	5 - 4 = 1
Almacén 3	3	4	5	6	0	<= 30	5 - 4 = 1
Almacén 4	4	5	6	7	0 (5)	<= 5	6 - 5 = 1
Requisitos	>= 0	>= 35	>= 15	>= 10	0		
Penalización C	null	4 - 3 = 1	5 - 4 = 1	6 - 5 = 1	null		La resta de los menores

PASO 4							
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F
Almacén 1	2(15)	3 (5)	4	5	0	<= 0	null
Almacén 2	3	4	5	6	0	<= 20	5 - 4 = 1
Almacén 3	3	4	5	6	0	<= 30	5 - 4 = 1
Almacén 4	4	5	6	7	0 (5)	<= 5	6 - 5 = 1
Requisitos	>= 0	>= 30	>= 15	>= 10	0		
Penalización C	null	4 - 4 = 0	5 - 5 = 0	6 - 6 = 0	null		

PASO 5							
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F
Almacén 1	2(15)	3 (5)	4	5	0	<= 0	null
Almacén 2	3	4	5	6	0	<= 20	6 - 5 = 1
Almacén 3	3	4 (30)	5	6	0	<= 0	null
Almacén 4	4	5	6	7	0 (5)	<= 5	7 - 6 = 1
Requisitos	>= 0	>= 0	>= 15	>= 10	0		
Penalización C	null	null	6 - 5 = 1	7 - 6 = 1	null		

PASO 6							
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F
Almacén 1	2 (15)	3 (5)	4	5	0	<= 0	null
Almacén 2	3	4	5 (15)	6	0	<= 5	6
Almacén 3	3	4 (30)	5	6	0	<= 0	null
Almacén 4	4	5	6	7	0 (5)	<= 5	7
Requisitos	>= 0	>= 0	>= 0	>= 10	0		
Penalización C	null	null	null	7 - 6 = 1	null		

PASO 7							
	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles	Penalización F
Almacén 1	2 (15)	3 (5)	4	5	0	<= 0	null
Almacén 2	3	4	5 (15)	6	0	<= 5	6
Almacén 3	3	4 (30)	5	6	0	<= 0	null
Almacén 4	4	5	6	7 (5)	0 (5)	<= 0	null
Requisitos	>= 0	>= 0	>= 0	>= 5	0		
Penalización C	null	null	null	6	null		

Esto nos lleva a la solución inicial factible o viable que queda tal que así:

	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Columna inservible	Proveedores disponibles
Almacén 1	2 (15)	3 (5)	4	5	0	<= 20
Almacén 2	3	4	5 (15)	6	0	<= 20
Almacén 3	3	4 (30)	5	6	0	<= 30
Almacén 4	4	5	6	7 (5)	0 (5)	<= 10
Requisitos	>= 15	>= 35	>= 15	>= 10	5	
Coste mínimo transporte: 2*15 + 3*5 + 5*15 + 6*5 + 4*30 + 7*5 + 0*5 = 305						

El coste es similar al devuelto por el método de North-West, dado que esta solución es degenerada (6 es distinto a 4 + (5 – 1)) y dado que la solución inicial de North-West era NO-degenerada, nos quedaremos con esta última ya que nos permite aplicar un método de mejora SIN pasos intermedios.

Antes de ello, vamos a calcular precisamente esta solución inicial mediante SAGE Math.

Resolución mediante software: SAGE Math v.9+

SAGE Math es un software matemático construido sobre Python y otras utilidades cómo NumPy, Sympy, etc. y basado en código libre, pensado para la resolución de problemas matemáticos, principalmente aquellos problemas que requieren de métodos iterativos mucho más fáciles de repetir por un computador.

Para resolver este problema se ha optado por seguir algunos apuntes dados por la propia asignatura para el desarrollo de las prácticas, haciendo uso de SAGE Math en su versión 9.2 instalado sobre un computador con Windows 10 64 bits.

Se adjuntan capturas del código que también se incluye en formato de notebook en el anexo de este trabajo.

In [18]:
```
# Definimos un PPL de minimización:
PROBLEMA = MixedIntegerLinearProgram(maximization=False)

# Definimos una serie de variables de decisión: lo hacemos cómo una matriz:
a11, a12, a13, a14 = PROBLEMA['Almacen1-Cliente1'], PROBLEMA['Almacen1-Cliente2'], PROBLEMA['Almacen1-Cliente3'], PROBLEMA['Almac
a21, a22, a23, a24 = PROBLEMA['Almacen2-Cliente1'], PROBLEMA['Almacen2-Cliente2'], PROBLEMA['Almacen2-Cliente3'], PROBLEMA['Almac
a31, a32, a33, a34 = PROBLEMA['Almacen3-Cliente1'], PROBLEMA['Almacen3-Cliente2'], PROBLEMA['Almacen3-Cliente3'], PROBLEMA['Almac
a41, a42, a43, a44 = PROBLEMA['Almacen4-Cliente1'], PROBLEMA['Almacen4-Cliente2'], PROBLEMA['Almacen4-Cliente3'], PROBLEMA['Almac
```

In [19]:
```
# Definimos una función objetivo a partir de los valores de la tabla del problema:
PROBLEMA.set_objective(2*a11 + 3*a12 + 4*a13 + 5*a14 +
                       3*a21 + 4*a22 + 5*a23 + 6*a24 +
                       3*a31 + 4*a32 + 5*a33 + 6*a34 +
                       4*a41 + 5*a42 + 6*a43 + 7*a44
                      )
```

In [20]:
```
# Definimos las restricciones de nuestro problema:

'''
Restricciones de oferta: proveedores
'''
PROBLEMA.add_constraint(a11 + a12 + a13 + a14 <= 20)
PROBLEMA.add_constraint(a21 + a22 + a23 + a24 <= 20)
PROBLEMA.add_constraint(a31 + a32 + a33 + a34 <= 30)
PROBLEMA.add_constraint(a31 + a32 + a33 + a34 <= 30)

'''
Restricciones de demanda: clientes
'''
PROBLEMA.add_constraint(a11 + a21 + a31 + a41 >= 15)
PROBLEMA.add_constraint(a12 + a22 + a32 + a42 >= 35)
PROBLEMA.add_constraint(a13 + a23 + a33 + a43 >= 15)
PROBLEMA.add_constraint(a14 + a24 + a34 + a44 >= 10)
```

In [21]:
```
# Obligar a la no negatividad del problema:
PROBLEMA.add_constraint(a11 >= 0)
PROBLEMA.add_constraint(a12 >= 0)
PROBLEMA.add_constraint(a13 >= 0)
PROBLEMA.add_constraint(a14 >= 0)

PROBLEMA.add_constraint(a21 >= 0)
PROBLEMA.add_constraint(a22 >= 0)
PROBLEMA.add_constraint(a23 >= 0)
PROBLEMA.add_constraint(a24 >= 0)

PROBLEMA.add_constraint(a31 >= 0)
PROBLEMA.add_constraint(a32 >= 0)
PROBLEMA.add_constraint(a33 >= 0)
PROBLEMA.add_constraint(a34 >= 0)

PROBLEMA.add_constraint(a41 >= 0)
PROBLEMA.add_constraint(a42 >= 0)
PROBLEMA.add_constraint(a43 >= 0)
PROBLEMA.add_constraint(a44 >= 0)
```

In [22]:
```
# Obtener la solución factible del PPL: esto es, el valor objetivo:
print("El valor objetivo del PPL resuelto es: ", PROBLEMA.solve())
```

```
El valor objetivo del PPL resuelto es:  305.0
```

Podemos confirmar que el valor objetivo coincide con el que calculamos antes mediante el método de North-West.

In [24]:
```
# El valor coincide con los cálculos efectuados a mano; tras lo cual, obtenemos los valores de las variables de decisión:
print("Coste de Almacén 1 a Cliente 1 es ", PROBLEMA.get_values(a11))
print("Coste de Almacén 1 a Cliente 2 es ", PROBLEMA.get_values(a12))
print("Coste de Almacén 1 a Cliente 3 es ", PROBLEMA.get_values(a13))
print("Coste de Almacén 1 a Cliente 4 es ", PROBLEMA.get_values(a14))

print("Coste de Almacén 2 a Cliente 1 es ", PROBLEMA.get_values(a21))
print("Coste de Almacén 2 a Cliente 2 es ", PROBLEMA.get_values(a22))
print("Coste de Almacén 2 a Cliente 3 es ", PROBLEMA.get_values(a23))
print("Coste de Almacén 2 a Cliente 4 es ", PROBLEMA.get_values(a24))

print("Coste de Almacén 3 a Cliente 1 es ", PROBLEMA.get_values(a31))
print("Coste de Almacén 3 a Cliente 2 es ", PROBLEMA.get_values(a32))
print("Coste de Almacén 3 a Cliente 3 es ", PROBLEMA.get_values(a33))
print("Coste de Almacén 3 a Cliente 4 es ", PROBLEMA.get_values(a34))

print("Coste de Almacén 4 a Cliente 1 es ", PROBLEMA.get_values(a41))
print("Coste de Almacén 4 a Cliente 2 es ", PROBLEMA.get_values(a42))
print("Coste de Almacén 4 a Cliente 3 es ", PROBLEMA.get_values(a43))
print("Coste de Almacén 4 a Cliente 4 es ", PROBLEMA.get_values(a44))
```

```
Coste de Almacén 1 a Cliente 1 es  15.0
Coste de Almacén 1 a Cliente 2 es  5.0
Coste de Almacén 1 a Cliente 3 es  -0.0
Coste de Almacén 1 a Cliente 4 es  -0.0
Coste de Almacén 2 a Cliente 1 es  -0.0
Coste de Almacén 2 a Cliente 2 es  20.0
Coste de Almacén 2 a Cliente 3 es  -0.0
Coste de Almacén 2 a Cliente 4 es  -0.0
Coste de Almacén 3 a Cliente 1 es  -0.0
Coste de Almacén 3 a Cliente 2 es  10.0
Coste de Almacén 3 a Cliente 3 es  15.0
Coste de Almacén 3 a Cliente 4 es  5.0
Coste de Almacén 4 a Cliente 1 es  -0.0
Coste de Almacén 4 a Cliente 2 es  -0.0
Coste de Almacén 4 a Cliente 3 es  -0.0
Coste de Almacén 4 a Cliente 4 es  5.0
```

2 (15)	3 (5)	4 (0)	5 (0)
3 (0)	4 (20)	5 (0)	6 (0)
3 (0)	4 (10)	5 (15)	6 (5)
4 (0)	5 (0)	6 (0)	7 (5)

Podemos confirmar que los valores de las variables de decisión coinciden con los calculados previamente.

Mejora de una solución inicial factible (“stepping-stone method”)

Ahora que ya disponemos de una solución inicial factible devuelta por el método de la esquina noreste o North-West, que además hemos verificado mediante SAGE, sabemos que su valor objetivo es el mejor que podemos obtener tras haber probado los tres algoritmos y, además, sabemos que dicha solución no necesita de movimientos intermedios al ser no-degenerada; vamos ahora a mejorar dicha solución mediante un movimiento de mejora dado por el método de “stepping-stone”.

El algoritmo de “stepping-stone”, también denominado algoritmo o método del paso a paso, trata de calcular cual es la variación del coste al enviar los productos desde un almacén a un cliente siguiendo una ruta no utilizada. Para ello, debemos partir de una solución inicial factible no degenerada, cómo es nuestro caso.

La idea consiste en trazar todos los caminos cerrados posibles partiendo de cualquiera de las celdas vacías de la tabla (celda a la que no se haya asignado nada tras, por ejemplo, aplicar North-West). Para hacer esto, avanzamos desde la celda vacía hasta una llena (asignada) y giramos siguiendo un ángulo recto hasta otra celda llena. Se repite hasta que lleguemos a la celda de inicio.

Una vez planteados todos los caminos posibles, se procede a evaluarlos; esto es, sumar los costes unitarios de las celdas del camino (solo se tienen en cuenta las casillas que se usan para girar en ángulo recto y componer el camino, no las intermedias).

Si la evaluación nos da un valor positivo, este caso supone un incremento del coste y por tanto no nos interesa. Si por el contrario la evaluación es nula, no se produce ni incremento ni decremento del coste, lo cual nos interesa.

n

m

	Cliente 1	**Cliente 2**	**Cliente 3**	**Cliente 4**	Columna inservible	Proveedores disponibles
Almacén 1	2 (15)	3 (5)	4	5	0	20
Almacén 2	3	4 (20)	5	6	0	20
Almacén 3	3	4 (10)	5 (15)	6 (5)	0	30
Almacén 4	4	5	6	7 (5)	0 (5)	10
Requisitos	15	35	15	10	5	

	Cliente 1	**Cliente 2**	**Cliente 3**	**Cliente 4**	Columna inservible	Proveedores disponibles
Almacén 1	2 (15)	3 (5)	4	5	0	20
Almacén 2	3	4 (20)	5	6	0	20
Almacén 3	3	4 (10)	5 (15)	6 (5)	0	30
Almacén 4	4	5	6	7 (5)	0 (5)	10
Requisitos	15	35	15	10	5	

Coste de la ruta = 4 – 3 + 4 – 5 = 0

Iterando en base a este principio hallamos todas las rutas. De estas, nos quedaremos con la mejor ruta.

Celdas no ocupadas	Circuito cerrado	Coste
Almacén 1, Cliente 3	$A_1C_3 \rightarrow A_1C_2 \rightarrow A_3C_2 \rightarrow A_3C_3$	4 – 3 + 4 – 5 = 0
Almacén 1, Cliente 4	$A_1C_4 \rightarrow A_1C_2 \rightarrow A_3C_2 \rightarrow A_3C_4$	5 – 3 + 4 – 6 = 0
Almacén 1, $C_{Inservible}$	$A_1C_{Inservible} \rightarrow A_1C_2 \rightarrow A_3C_2 \rightarrow A_3C_4 \rightarrow A_4C_4 \rightarrow A_4C_{Inservible}$	– 3 + 4 – 6 + 7 =2
Almacén 2, Cliente 1	$A_2C_1 \rightarrow A_2C_2 \rightarrow A_1C_2 \rightarrow A_1C_1$	3 – 4 +. 3 – 2 = 0
Almacén 2, Cliente 3	$A_2C_3 \rightarrow A_2C_2 \rightarrow A_3C_2 \rightarrow A_3C_3$	5 – 4 + 4 – 5 = 0
Almacén 2, Cliente 4	$A_2C_4 \rightarrow A_2C_2 \rightarrow A_3C_2 \rightarrow A_3C_4$	6 – 4 + 4 - 6 = 0
Almacén 2, $C_{Inservible}$	$A_2C_2 \rightarrow A_2C_2 \rightarrow A_3C_2 \rightarrow A_3C_4 \rightarrow A_4C_4 \rightarrow A_4C_{Inservible}$	- 4 + 4 - 6 + 7 = 1
Almacén 3, Cliente 1	$A_3C_1 \rightarrow A_3C_2 \rightarrow A_1C_2 \rightarrow A_1C_1$	3 – 4 + 3 – 2 = 0
Almacén 3, $C_{Inservible}$	$A_3C_{Inservible} \rightarrow A_3C_4 \rightarrow A_4C_4 \rightarrow A_4C_{Inservible}$	- 6 + 7 = 1
Almacén 4, Cliente 1	$A_4C_1 \rightarrow A_4C_4 \rightarrow A_3C_4 \rightarrow A_3C_2 \rightarrow A_1C_2 \rightarrow A_1C_1$	4 – 7 + 6 – 4 + 3 - 2 = 0
Almacén 4, Cliente 2	$A_4C_2 \rightarrow A_4C_4 \rightarrow A_3C_4 \rightarrow A_3C_2$	5 – 7 + 6 – 4 = 0
Almacén 4, Cliente 3	$A_4C_3 \rightarrow A_4C_4 \rightarrow A_3C_4 \rightarrow A_3C_3$	6 – 7 + 6 – 5 = 0

	Cliente 1	**Cliente 2**	**Cliente 3**	**Cliente 4**	Columna inservible	Proveedores disponibles
Almacén 1	2 (15)	3 (5)	4	5	0	20
Almacén 2	3	4 (20)	5	6	0	20
Almacén 3	3	4 (10)	5 (15)	6 (5)	0	30
Almacén 4	4	5	6	7 (5)	0 (5)	10
Requisitos	15	35	15	10	5	

Coste total mínimo del transporte = 2 * 15 + 3 * 5 + 4 * 20 + 4 * 10 + 5 * 15 + 6 * 5 + 7 * 5 + 0 * 5 = 305

Otras vías (Python)

Para complementar el desarrollo de este trabajo también añadimos soluciones al algoritmo del método noreste o North-West desarrollados en Python (v.3+) que se incluyen aquí a modo de capturas pero que también están contenidos en el anexo del trabajo.

```python
des = [[2, 3, 4, 5],
       [3, 4, 5, 6],
       [3, 4, 5, 6],
       [4, 5, 6, 7]]

supp = [20, 20, 30, 10]
deman = [15, 35, 15, 10]
col = [0, 0, 0, 0]
row = [0, 0, 0, 0]
rows = 4
columns = 4
ans = 0
for i in range(0, columns):
    for j in range(0, rows):
        if col[i] == 1 or row[j] == 1:
            continue
        m = min(supp[i], deman[j])
        supp[i] = supp[i] - m
        deman[j] = deman[j] - m
        ans = ans + des[i][j] * m
        if deman[j] == 0:
            row[j] = 1
        if supp[i] == 0:
            col[i] = 1

print(ans)
```

Resultado de la ejecución de un método desarrollado en Python para el cálculo del valor objetivo mediante el método de North-West.

```
PS C:\Users\aleja> & python "e:/SAGEMath/MASI/Algoritmos/North-west corner rule method.py"
305
PS C:\Users\aleja>
```

Conclusiones y comentarios

Los problemas de programación lineal son un conjunto en sí mismo de problemas matemáticos que engloba otros muchos. El caso concreto del problema del transporte no es otra cosa sino un tipo específico de resolución de PPLs cómo podría serlo el método Simplex (abordado en esta asignatura).

En concreto, el método, problema o algoritmo del transporte (y sus métodos para hallar soluciones iniciales y mejorarlas; Nort-West, Least-Cost, Vogel y Stepping-stones) son un tipo de problema muy recurrente en el campo de la economía, así como en algunas áreas de la ingeniería.

Este trabajo ha sido desarrollado íntegramente por Alejandro Fernández Trigo (50% participación, aportando desarrollo de la memoria, partes de la presentación, código SAGE, búsqueda de información) y Juan Diego Villalobos Quirós (50% participación, aportando desarrollo de la presentación, cálculos de tablas, código Python y búsqueda de información).

Se encuentra enmarcado dentro de la asignatura Matemática Aplicada a Sistemas de Información de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla para el curso 2020/2021.

Bibliografía

- ‘Investigación de operaciones’ de Wayne L. Winston. (https://bit.ly/3wKD5N5)
- ‘Formulación de Modelos de Programación Lineal’ de Humberto Chávez Milla. (https://bit.ly/2RKMlC3)
- ‘Nuevos métodos para la obtención de soluciones iniciales en el problema del transporte’ de Francisco López Ruiz. (https://bit.ly/3vA7rl5)
- ‘Implementación del método esquina noroeste’ de Gabriela Katiuska & Chimbo Ponce. (https://bit.ly/34oR4fc)
- ‘The Distribution of a Product from Several Sources to Numerous Localities’ de F. L. Hitchcock para la Jour. Math. and Phys., 1941.
- Fuentes propias dadas por el profesorado de la asignatura.
- Otras fuentes.