28 de Noviembre,
Concepción

Estudio de Caso

Optimización de rutas de abastecimiento en "SurVending"

Felipe Condori, Jeremías Torres, Diego Varas
Optimización I
Universidad de Concepción

Resumen

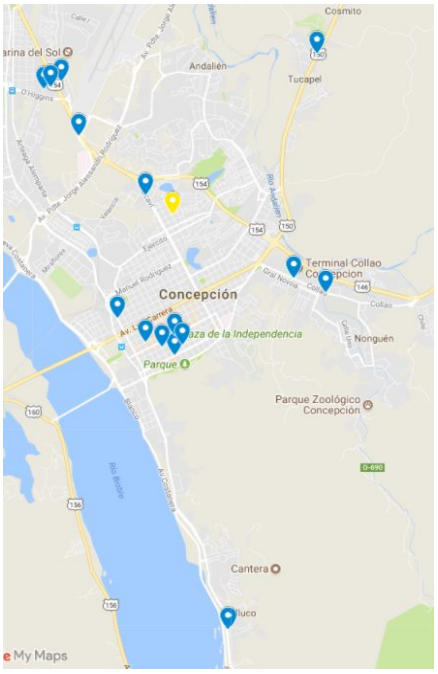
Objetivo: Minimizar los tiempos de reposición de productos en máquinas expendedoras pertenecientes a la empresa "SurVending" el cual utilizaba 3 autos para recorrer las distintas empresas donde se presta el servicio, para resolver esto, se han utilizado técnicas de optimización. Método: Se utilizó el modelo de Problema de Enrutamiento de Vehículo con Capacidad (CVRP), el cual es una variante del problema de optimización combinatoria de Enrutamiento de Vehículos (VRP). Resultados: Se logró establecer una ruta para cada vehículo reponedor y disminuir el tiempo de reposición, el cual pasó de 4 horas a 2 horas y 45 minutos.

Palabras Claves: Vehicle routing problem - VRP - Capacitated Vehicle Routing Problem - CVRP - MILP -Mixed Integer Programming - TSP - Travelling Salesman Problem.

Introducción

El ***problema de enrutamiento de vehículos*** [1] (**VRP**) es un problema de optimización combinatoria y de programación entera que resuelve el conjunto óptimo de rutas para una flota de vehículos que debe satisfacer las demandas de un conjunto dado de clientes. Es una generalización del conocido ***Problema del vendedor viajero*** (TSP). La primera definición aparece en una artículo de George Dantzig y John Ramser en 1959, en donde plantea una aproximación algorítmica y fue aplicado para entregas de gasolina. Determinar la solución óptima es un problema NP-hard de optimización combinatoria. Las implementaciones más utilizadas para resolver el problema se basan en heurísticas debido a que para grandes instancias del problema, que como sucede en ejemplos reales, producen buenos resultados.
La tarea de VRP es determinar un conjunto de rutas óptimas desde un nodo central(almacén) a una serie de nodos (consumidores). Se aplica en transporte, distribución y logística. Para resolver el problema presentado en este documento se utiliza la variante ***Problema de Enrutamiento de Vehículo con Capacidad*** (CVRP), en la cual los vehículos tienen una capacidad de carga definida.

Descripción del problema



Survending [2] es una empresa local dedicada al rubro del Vending, esto es, venta de alimentos a través de máquinas expendedoras. Las máquinas deben ser abastecidas dos veces al día (turno diurno y nocturno), y se encuentran ubicadas en distintos sectores de Concepción, como se puede apreciar en la ilustración, en donde la marca amarilla corresponde al nodo almacén ó nodo 0, que es donde se encuentra la bodega. Desde aquí salen diariamente 3 vehículos a abastecer las máquinas ubicadas en empresas o universidades marcadas en azul. Cabe destacar que en cada punto azul pueden haber 1 o más máquinas.
El objetivo es encontrar las rutas óptimas para los 3 automóviles que salen en cada turno, de manera que minimice los tiempos y costos de viaje.

Definición de VRP

Un VRP clásico [3] se define en un grafo dirigido $G(V,E)$ que consiste en un conjunto de nodos $V = \{0, 1, ..., n\}$ y el conjunto de aristas $E = \{(i,j) \mid i,j \in V, i \neq j\}$. El nodo almacén es el nodo inicial 0, en el cual existen m vehículos idénticos de igual capacidad Q.
Cada nodo consumidor i $\in$ V - {0} está asociado a la necesidad qi bienes no negativos.
Cada arista que conecta un par de nodos está asociado a un costo cij. Todos los costos se pueden expresar utilizando la matriz de peso C.
El problema consiste en determinar un conjunto de rutas para las cuales es válido:

- El inicio y el final de cada ruta es el nodo 0 (almacén),
- Cada nodo consumidor es visitado sólo una vez,
- Minimiza el coste total de la aprobación de todas las rutas.

Algoritmo para resolver VRP

Un problema *de programación lineal* (LP) es un problema de optimización de la forma:

$$max\{c^T x \mid Ax \leq b,\ x \geq 0\}$$

donde dada la matriz $A \in \Re^{m,n}$, el vector de recursos escasos b $\in \Re^m$ y vector c $\in \Re^n$, si alguna o todas las variables dentro del vector x $\in \Re^n$ desconocido, son enteros, el problema se llama Mixed-integer linear programming (MILP) [4].

Modelo VRP

Para detallar el modelo se definen las variables:

n : *cantidad de nodos, considerando el almacén y los consumidores.*

di : *demanda de productos del nodo i.*

mi : *cantidad de vehículos, todos con capacidad Q y ubicados en el nodo* 0.

cij : *Tiempo necesario para ir del nodo i al nodo j.*

$y_{ih} = \{1,$ *si el nodo i pertenece a la ruta h.* $0,$ *si no.* $\quad h = 1, ..., m$

Las restricciones que se aplican a estas variables son:

$$\sum_{h=1}^{m} y_{ih} = 1, \qquad i = 1, ..., n \qquad (1)$$

Igualdad (1) establece que cada consumidor debe pertenecer exactamente a una ruta. La cantidad de vehículos usados puede imponerse estableciendo un límite al número de veces que se asigna el nodo 0 a la ruta:

$$\sum_{h=1}^{m} y_{0h} = m \qquad (2)$$

La limitación de la capacidad del vehículo (3) también se puede expresar al imponer que la suma de las solicitudes de los consumidores asignadas al mismo vehículo no exceda la capacidad máxima Q de los automóviles:

$$\sum_{i=0}^{n} d_{ih} y_{ih} = Q, \qquad h = 1, ..., m \quad (3)$$

$x_{ij}^{h} = \{1,$ *si el vehículo h va desde el nodo i al nodo j.* 0 *si no.* $\quad i,j = 0, ..., n, \qquad h = 1, ..., m$

Para que la ruta sea aceptada como solución, es necesario cumplir con las restricciones para el arco de entrada (4), arco de salida (5) y eliminación de ciclos (6):

$$\sum_{j=0, j\neq i}^{n} x_{ij}^{h} = y_{ih}, \qquad i = 0, ..., n \; h = 1, ..., m \quad (4)$$

$$\sum_{j=0, j\neq i}^{n} x_{ji}^{h} = y_{ih}, \qquad i = 0, ..., n \; h = 1, ..., m \quad (5)$$

$$\sum_{i,j \in S} x_{ij}^{h} \leq |S| - 1, \qquad S \subseteq \{1, ..., n\}, \; |S| \geq 2, \; h = 1, ..., m \; (6)$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij}^{h} = \sum_{j=1}^{n} y_{i}^{h} - 1, \qquad h = 1, ..., m \; (6)$$

Es posible eliminar una parte de las variables usando las restricciones del tipo de igualdad dada por (4) y (5). La restricción (4) define que en la ruta desde el nodo observado sólo un arco de salida puede ir a uno de los otros nodos. Análogamente para (5). La restricción (6) es equivalente a la restricción (6 '), que es más conveniente para su programación. Ésta restricción elimina los ciclos, es decir, para n nodos que pertenecen a la misma ruta h se permite que existan n-1 aristas entre estos n nodos.

Una formulación completa del problema de VRP sería:

$$min\ z = \sum_{h=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}^{h}$$

$$\sum_{j=0,j\neq i}^{n} x_{ij}^{h} = \sum_{j=0,j\neq i}^{n} x_{ji}^{h} = y_{ih} \qquad y = 1,...,n,\ h = 1,...,m$$

$$\sum_{h=1}^{m}\sum_{j=0}^{n} x_{ij}^{h} = 1 \qquad i = 1,...,n$$

$$\sum_{i=1}^{n}\sum_{j=0}^{n} d_i x_{ij}^{h} \leq Q \qquad h = 1,...,m$$

$$x_{ij}^{h} \in \{0,1\} \qquad i,j = 0,...,n$$

Pseudocódigo

```
1 p = MixedIntegerLinearProgram(maximization=False, solver = "GLPK")
2 w = p.new_variable(integer=True, nonnegative=True)
3
4 #1--cada nodo, aparte del almacén, pertenece exactamente a una ruta
5 for i in range (1,n):
6     constraint = 0
7     for h in range (0,m):
8           p.set_max(w[i,h], 1)
9           constraint = constraint + w[i,h]
10    p.add_constraint(constraint == 1)
11 #2--El depósito pertenece a una única ruta
12 for h in range(0,m):
13    p.add_constraint(w[0,h] == 1)
14 #3--limitación de la capacidad del vehículo: la suma de las solicitudes
de los clientes asignadas al mismo vehículo no supera la capacidad máxima
15 for h in range(0,m):
16    constraint = 0
17    for i in range(0,n):
18          constraint = constraint + demands[i]*w[i,h]
19    p.add_constraint(constraint <= Q)
20 #8--si el nodo pertenece a la ruta, la suma de las ramas de salida de
ese nodo según todos los demás es igual a 1
21 x = p.new_variable(integer=True, nonnegative=True)
22 for i in range(0,n):
23    for h in range(0,m):
24          constraint1 = 0
25          constraint2 = 0
26          for j in range(0,n):
27                if i!=j:
28                      p.set_max(x[i,j,h], 1)
29                      constraint1 += x[i, j, h]
30                      constraint2 += x[j, i, h]
31          p.add_constraint(constraint1 == w[i,h])
32          p.add_constraint(constraint2 == w[i,h])
33 #6--eliminación de bucles
34 for h in range(0,m):
```

```
35    constraintx = 0
36    constrainty = 0
37    for i in range(1,n-1):
38          constrainty += w[i,h]
39          for j in range(i+1,n):
40               constraintx += x[i,j,h]#todas las ramas X_ij en la ruta h
41    constrainty += w[n-1, h] #debe agregarse al último nodo
42    p.add_constraint(constraintx == constrainty - 1)
43 #9--desde un nodo que no es un almacén puede ir solo una arista a otro
nodo
44 for i in range (1,n):
45    constraint = 0
46    for h in range (0,m):
47          for j in range (0,n):
48                if i!=j:
49                      constraint += x[i,j,h]
50    p.add_constraint(constraint == 1)
51 #10--la capacidad actual del vehículo es >= a partir de la suma de las
solicitudes de nodos que aún no se han visitado
52 for h in range(0,m):
53    constraint = 0
54    for i in range(1,n):
55          for j in range(0,n):
56                if i!=j:
57                      constraint += demands[i]*x[j,i,h]
58    p.add_constraint(constraint <= Q)
59 #7--criterio: la ruta mínima recorrida en todas las rutas
60 J = 0;
61 for h in range(0, m):
62    for i in range(0,n):
63          for j in range(0,n):
64                J = J + costs[i][j]*x[i,j,h]
65 p.set_objective(J)
66 print  p.solve() #imprime óptimo
```

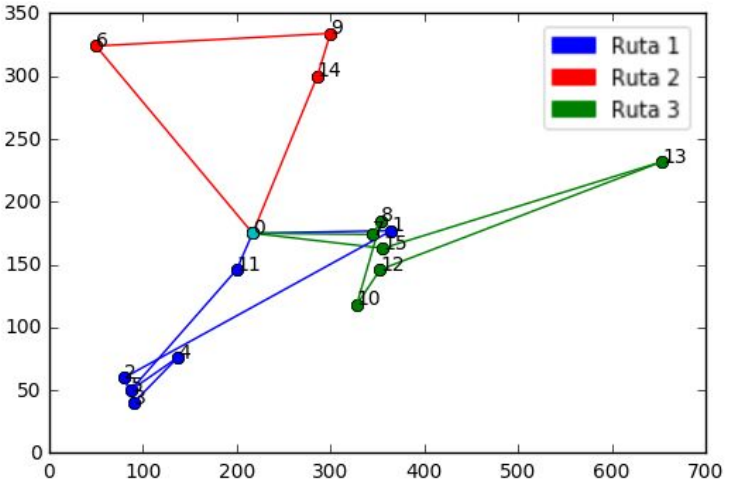
Resultados

Dados los datos de las instancias [Ver Anexo: Instancias del Caso] los resultados obtenidos son los siguientes:

- Valor objetivo : 165.0 minutos.
- Tiempo de ejecución: 0.23 segundos.

Se generaron 816 variables y 138 restricciones lo que da un tamaño de problema de 112608.

Las 3 rutas generadas se muestran en el siguiente gráfico con azul, rojo y verde:



(Figura 1)

Análisis de resultados

El modelo se programó en Python 2.7 con ayuda de SageMath 8.3 para la ejecución del Mixed-integer linear programming (MILP) con solver de GLPK. La implementación del problema fue ejecutada en una máquina con procesador Intel Core i3-5005U de 2.0Ghz con sistema operativo Windows 10 versión 1809.

En la *Figura 1* se describen 3 rutas, una para cada vehículo, las cuales son las óptimas para para poder abarcar todas entregas, con su correspondiente demanda, en el menor tiempo posible:

- La ruta de color rojo representa al vehículo que recorre las empresas Digosa, Constructora bio casa y Deligourmet.
- La ruta de color azul comprenden las empresas Sergio Escobar automotriz, Econorent, Suractivo, Linde, Inacap y Udla
- La ruta de color verde comprende las empresas de Optimisa, Ainahue Club de Campo, Salazar Israel automotriz, AFP Modelo, Strip Villuco y Caja de Compensación la araucana.

Todas en el orden mencionado. El valor objetivo obtenido se interpreta como la sumatoria del tiempo utilizado por cada vehículo en cada una de las rutas realizadas.
Estos resultados mejoran considerablemente el tiempo en que se realizaba el abastecimiento, el cual supera las 4 horas, por lo que siguiendo las rutas entregadas anteriormente, dicho abastecimiento se puede lograr en 165 min que es casi 3 horas.

Conclusión

En principio, al tratarse de un problema de optimización combinatoria (NP-Hard) e incluso una instancia real con puntos y variables de decisión que involucran el gran Concepción podría parecer una labor compleja, sin embargo a través de la identificación de conceptos asociados a la programación matemática, como la definición de una instancia particular del problema y el uso de herramientas como GLPK, LINGO, CPLEX, o similares, es posible determinar mejores condiciones que puedan influir en la renta de una empresa como Survending o cualquier otra de rubro o servicio similar. Demostrandonos así la gran utilidad que la programación matemática tiene para ofrecernos en problemas tan comunes como la repartición de determinado commodity.

Survending corresponde a una PYME con una flota de tan sólo 3 vehículos repartidores, la búsqueda y optimización de la ruta óptima para repartir indica (a partir de lo mencionado por su socios) habría una mejora del uso de la flota de 2 horas, esto implicaría la utilización de la flota en nuevas actividades o incluso el mismo personal con este ahorro podría desempeñar alguna otra labor de importancia.

Referencias

1.- Problema de enrutamiento de vehículos

En el texto: (Es.wikipedia.org, 2017)

Bibliografía: Es.wikipedia.org. (2017). *Problema de enrutamiento de vehículos*. [online] Available at: http://es.wikipedia.org/wiki/Problema_de_enrutamiento_de_veh%C3%ADculos

2.- SurVending / Máquinas Vending

En el texto: (Survending.cl, 2017)
Bibliografía: Survending.cl. (2017). *Sur Vending / Máquinas Vending Snacks Bebidas*. [online]

3.- Sljivo, Amina. (2015). Vehicle Routing Problem solution using Mixed Integer Linear Programming. . 10.13140/RG.2.1.2126.5128.

4.- Mixed Integer Linear Proogramming — Sage Reference Manual V8.0: Numerical Optimization.

En el texto: ("Mixed Integer Linear Programming — Sage Reference Manual v8.0: Numerical Optimization", 2017)

Bibliografía: *Mixed Integer Linear Programming — Sage Reference Manual v8.0: Numerical Optimization*. (2017). *Doc.sagemath.org*. Retrieved 29 November 2017, from http://doc.sagemath.org/html/en/reference/numerical/sage/numerical/mip.html

Anexo

Instancias del caso

Considerando n:cantidad de nodos, m: cantidad de vehículos y Q:carga de cada vehículo, donde n es igual a 16, de los cuales 15 son consumidores y uno es almacén, m es igual a 3 y Q=100 objetos.

i	Nombre	Demanda di (objetos)
0	Bodega	0
1	UDLA	20
2	Inacap	20
3	Linde	5
4	Suractivo	10
5	Econorent	5
6	Digosa	5
7	Optimisa	5
8	Ainahue Club de Campo	5
9	Constructora Bio Casa	5
10	Salazar Israel Automotriz	5
11	Sergio escobar Automotriz	5
12	AFP Modelo	5
13	Strip Villuco	5
14	Deligourmet	5
15	Caja Compensación la Araucana	5

Matriz de costos (C)

Matriz generada por las instancias del caso.
Ci,j = Tiempo estimado que toma ir de i a j (minutos) i,j = 0,...,15

-	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-	15	27	12	9	8	14	11	11	11	9	4	12	21	12	14
1	14	-	14	16	17	15	20	7	1	15	8	14	4	14	16	2
2	24	15	-	6	4	2	16	15	15	11	9	6	16	20	14	15
3	13	14	10	-	7	11	19	16	16	17	13	9	15	19	18	14
4	8	13	5	12	-	6	14	14	17	14	11	5	13	19	12	14
5	8	14	6	7	3	-	15	15	16	12	9	5	16	19	14	15
6	10	16	10	15	13	11	-	14	19	13	18	11	15	25	11	16
7	12	5	12	14	15	13	18	-	4	12	9	13	3	15	13	3
8	13	5	17	16	16	16	16	4	-	13	9	13	5	14	14	3
9	12	13	14	17	13	13	11	11	13	-	14	12	14	24	3	14
10	8	5	14	13	12	13	13	4	7	12	-	8	6	14	14	7
11	4	13	5	10	8	6	14	12	12	12	10	-	13	20	11	14
12	12	6	12	15	14	13	19	5	6	12	5	13	-	16	13	3
13	21	10	17	17	14	16	24	11	12	19	11	14	11	-	20	11
14	10	13	10	15	10	11	11	13	12	2	11	13	12	23	-	1
15	11	5	12	14	13	13	18	2	5	11	4	13	2	15	12	-

Código Fuente

Código escrito en lenguaje Python 2.7 ejecutable con el complemento SageMath 8.3. Como opción online para la ejecución del código es https://sagecell.sagemath.org/ .

```
import numpy
import math
import matplotlib.pyplot as plt
import matplotlib.pyplot as pltt
import matplotlib.patches as mpatches
import time
from sage.all import *

# INICIALIZACIÓN DEL PROBLEMA
# n: cantidad de nodos, incluido el depósito
# m - número de vehículos (= número de rutas)
# Q - capacidad del vehículo
n = 16
m = 3
Q = 100
nodes = numpy.arange(0,n,1)
nodos=[0,217,175,
       1,365,177,
       2,80,60,
       3,90,40,
       4,137,76,
       5,87,50,
       6,50,324,
       7,345,174,
       8,354,185,
       9,300,334,
       10,328,118,
       11,200,146,
       12,353,146,
       13,654,232,
       14,286,300,
       15,356,163]
requisitos=[0,0,
            1,20,
            2,20,
            3,5,
```

```
            4,10,
            5,5,
            6,5,
            7,5,
            8,5,
            9,5,
            10,5,
            11,5,
            12,5,
            13,5,
            14,5,
            15,5]
empresa = ['0 Bodega','1 UDLA','2 Inacap','3 Linde','4 Suractivo','5
Econorent','6 Digosa','7 Optimisa','8 Ainahue Club de Campo','9
Constructora Bio Casa',
            '10 Salazar Israel Automotriz','11 Sergio escobar
Automotriz','12 AFP Modelo','13 Strip Villuco','14 Deligourmet','15
Caja Compensación la Araucana']
c = numpy.zeros(shape=(n,n))
c =
[[0,15,27,12,9,8,14,11,11,11,9,4,12,21,12,14],[14,0,14,16,17,15,20,7,
1,15,8,14,4,14,16,2],[24,15,0,6,4,2,16,15,15,11,9,6,16,20,14,15],[13,
14,10,0,7,11,19,16,16,17,13,9,15,19,18,14],[8,13,5,12,0,6,14,14,17,14
,11,5,13,19,12,14],[8,14,6,7,3,0,15,15,16,12,9,5,16,19,14,15],[10,16,
10,15,13,11,0,14,19,13,18,11,15,25,11,16],[12,5,12,14,15,13,18,0,4,12
,9,13,3,15,13,3],[13,5,17,16,16,16,16,4,0,13,9,13,5,14,14,3],[12,13,1
4,17,13,13,11,11,13,0,14,12,14,24,3,14],[8,5,14,13,12,13,13,4,7,12,0,
8,6,14,14,7],[4,13,5,10,8,6,14,12,12,12,10,0,13,20,11,14],[12,6,12,15
,14,13,19,5,6,12,5,13,0,16,13,3],[21,10,17,17,14,16,24,11,12,19,11,14
,11,0,20,11],[10,13,10,15,10,11,11,13,12,2,11,13,12,23,0,13],[11,5,12
,14,13,13,18,2,5,11,4,13,2,15,12,0]]

#=====================================================================
xcoord = []
ycoord = []
demands = []
for i in numpy.arange(1,len(nodos),3):
      xcoord.append(nodos[i])
      ycoord.append(nodos[i+1])
for i in numpy.arange(1,len(requisitos),2):
      demands.append(requisitos[i])
print 'Demandas:\n'
```

```
for i in range(n):
    print str(empresa[i])+'='+str(demands[i])+' '
costs = c
print '\nMatriz de Costos:'

for i in range(len(costs)):
    print costs[i]

p = MixedIntegerLinearProgram(maximization=False, solver = "GLPK")
w = p.new_variable(integer=True, nonnegative=True)
start = time.time()

#1--cada nodo, aparte del almacén, pertenece exactamente a una ruta
for i in range (1,n):
      constraint = 0
      for h in range (0,m):
            p.set_max(w[i,h], 1)
            constraint = constraint + w[i,h]
      p.add_constraint(constraint == 1)
#2--El depósito pertenece a una única ruta
for h in range(0,m):
      p.add_constraint(w[0,h] == 1)
#3--limitación de la capacidad del vehículo: la suma de las
solicitudes de los clientes asignadas al mismo vehículo no supera la
capacidad máxima
for h in range(0,m):
      constraint = 0
      for i in range(0,n):
            constraint = constraint + demands[i]*w[i,h]
      p.add_constraint(constraint <= Q-26)
#8--si el nodo pertenece a la ruta, la suma de las ramas de salida de
ese nodo según todos los demás es igual a 1
x = p.new_variable(integer=True, nonnegative=True)
for i in range(0,n):
      for h in range(0,m):
            constraint1 = 0
            constraint2 = 0
            for j in range(0,n):
                  if i!=j:
                        p.set_max(x[i,j,h], 1)
                        constraint1 += x[i, j, h]
                        constraint2 += x[j, i, h]
```

```
            p.add_constraint(constraint1 == w[i,h])
            p.add_constraint(constraint2 == w[i,h])
#6--eliminación de bucles: br_grana = br_cvorova-1
for h in range(0,m):
      constraintx = 0
      constrainty = 0
      for i in range(1,n-1):
            constrainty += w[i,h]
            for j in range(i+1,n):
                  constraintx += x[i,j,h]
      constrainty += w[n-1, h] #debe agregarse al último nodo
      p.add_constraint(constraintx == constrainty - 1)
#9--desde un nodo que no es un almacén puede ir solo una arista a
otro nodo
for i in range (1,n):
      constraint = 0
      for h in range (0,m):
            for j in range (0,n):
                  if i!=j:
                        constraint += x[i,j,h]
      p.add_constraint(constraint == 1)
#10--la capacidad actual del vehículo es >= a partir de la suma de
las solicitudes de nodos que aún no se han visitado
for h in range(0,m):
      constraint = 0
      for i in range(1,n):
            for j in range(0,n):
                  if i!=j:
                        constraint += demands[i]*x[j,i,h]
      p.add_constraint(constraint <= Q-26)
#7--criterio: la ruta mínima recorrida en todas las rutas
J = 0;
for h in range(0, m):
      for i in range(0,n):
            for j in range(0,n):
                  J = J + costs[i][j]*x[i,j,h]
p.set_objective(J)

print '\nValor Función Objetivo:', p.solve(),'min'
print '\nNumero de Variables:',p.number_of_variables()
print '\nNumero de Restricciones:',p.number_of_constraints()
print '\nTamaño del
```

```
problema:',int(p.number_of_variables())*int(p.number_of_constraints()
)

end = time.time()
fromn = []
ton = []
route = []

for i,v in p.get_values(x).iteritems():
      if v>0:
            fromn.append(i[0])
            ton.append(i[1])
            route.append(i[2])

print '\nTiempo de ejecución del programa = ',
round(end-start,2),'seg'
print '\nGrafico de Rutas'

for i in range (0, len(fromn)):
      if route[i] == 0:
            pltt.plot(xcoord[fromn[i]], ycoord[fromn[i]], 'ro')
            pltt.plot(xcoord[ton[i]], ycoord[ton[i]], 'ro')
            pltt.plot([xcoord[fromn[i]],
xcoord[ton[i]]],[ycoord[fromn[i]], ycoord[ton[i]]], 'r-')
      elif route[i]==1:
            pltt.plot(xcoord[fromn[i]], ycoord[fromn[i]], 'go')
            pltt.plot(xcoord[ton[i]], ycoord[ton[i]], 'go')
            pltt.plot([xcoord[fromn[i]],
xcoord[ton[i]]],[ycoord[fromn[i]], ycoord[ton[i]]], 'g-')
      elif route[i]==2:
            pltt.plot(xcoord[fromn[i]], ycoord[fromn[i]], 'bo')
            pltt.plot(xcoord[ton[i]], ycoord[ton[i]], 'bo')
            pltt.plot([xcoord[fromn[i]],
xcoord[ton[i]]],[ycoord[fromn[i]], ycoord[ton[i]]], 'b-')
      elif route[i]==3:
            pltt.plot(xcoord[fromn[i]], ycoord[fromn[i]], 'mo')
            pltt.plot(xcoord[ton[i]], ycoord[ton[i]], 'mo')
            pltt.plot([xcoord[fromn[i]],
xcoord[ton[i]]],[ycoord[fromn[i]], ycoord[ton[i]]], 'm-')
      elif route[i]==4:
            pltt.plot(xcoord[fromn[i]], ycoord[fromn[i]], 'ko')
            pltt.plot(xcoord[ton[i]], ycoord[ton[i]], 'ko')
```

```
           pltt.plot([xcoord[fromn[i]],
xcoord[ton[i]]],[ycoord[fromn[i]], ycoord[ton[i]]], 'k-')
pltt.plot(ycoord[0], xcoord[0], 'co')

labels = ['{0}'.format(i) for i in nodes]

for label, x, y in zip(labels, ycoord, xcoord):
     pltt.annotate(label, xy = (y,x))

red_patch = mpatches.Patch(color='red', label='Ruta 2')
blue_patch = mpatches.Patch(color='blue', label='Ruta 1')
green_patch = mpatches.Patch(color='green', label='Ruta 3')
pltt.legend(handles=[blue_patch,red_patch,green_patch])

pltt.show()
```