29 de Noviembre,
Concepción

Caso de estudio

Optimización de rutas de abastecimiento en *SurVending*

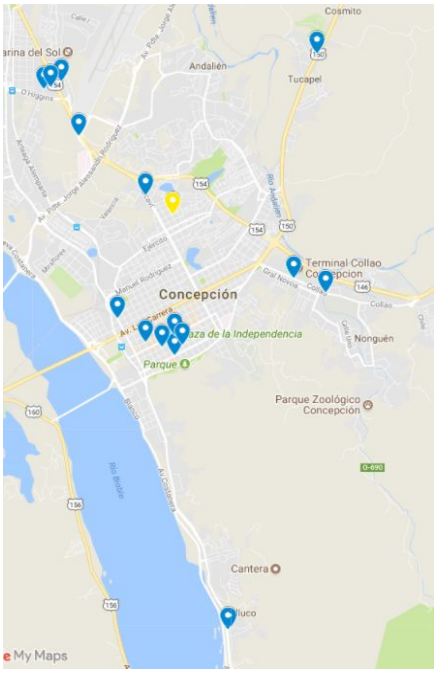
Felipe Condori, Isaías Huerta, Diego Varas
Optimización I
Universidad de Concepción

Introducción

El problema de enrutamiento de vehículos [1] (**VRP**) es un problema de optimización combinatoria y de programación de entero 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 éste 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 regional 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 como 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 nodo de inicio hay m vehículos idénticos de igual capacidad Q.

Cada nodo consumidor i ∈ 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.

Algoritmos 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 algunas 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.
m: cantidad de vehículos, todos con capacidad Q y ubicados en el nodo 0.
cij: Tiempo necesario para ir del nodo i al nodo j.
$y_{ih} = \{1,\ si\ el\ nodo\ i\ pertenece\ a\ la\ ruta\ h.\ 0,\ si\ no.\qquad h\ =\ 1,...,m$

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:

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

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

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):

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

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

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

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

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:

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

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

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

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

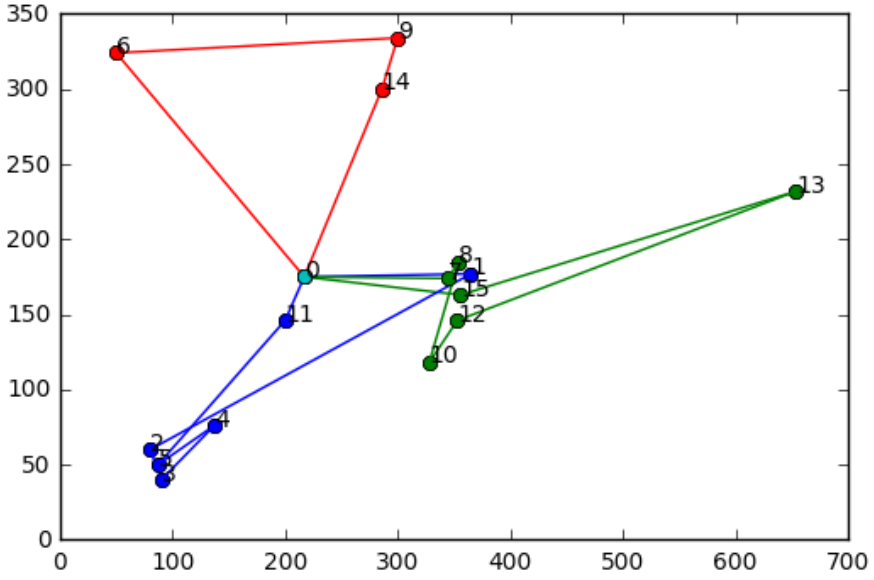
$$x_{ij}^{h} \in \{0,1\} \qquad i,j = 0,...,n$$

Resultados

Dados los datos de las instancias [Ver Anexo 1] los resultados obtenidos son los siguientes:

Valor objetivo : 165.0 minutos. Tiempo de ejecución: 0.1524 segundos.

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



(Gráfico Referencial)

Análisis de resultados

En el gráfico mostrado anteriormente 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 en cada vehículo, no simultáneo, de las rutas realizadas.

Implementación

El modelo se programó en Python 3.6 con ayuda de SageMath para la ejecución del Mixed-integer linear programming (MILP) con solver de GLPK. La implementación del problema fue ejecutada en una máquina Intel i3-5005U 2Ghz con sistema operativo Ubuntu 16.04.3 LTS.

Pseudocódigo

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

#1--cada nodo, aparte del almacen, 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)
#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
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]#todas las ramas X_ij en la ruta 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 almacen 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)
#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  p.solve() #imprime óptimo
```

Anexo 1: Instancias del caso

Considerando :
n igual a 16, donde 15 son consumidores y uno es almacén y m igual a 3 vehículos con capacidad de carga Q=100kg

i	**Nombre**	**Demanda di (kg)**
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 Automotora	5
11	Sergio escobar Automotora	5
12	AFP Modelo	5
13	Strip Villuco	5
14	Deligourmet	5
15	Caja Compensación la Araucana	5

Ci,j= (definida en función del tiempo: minutos)

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

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] Available at: http://survending.cl/

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