

# 2013

## Traveling salesman problem



Lucia Moyeda Cornejo  
Francisco Lumbreras  
Gabriela Martinez Aldape  
FIME  
03/06/2013

## DESCRIPCION DEL PROBLEMA

El problema consiste en trabajar usando listas restringidas de candidatos las cuales pueden ser por:

- LRC.- cardinalidad donde  $\alpha[0,1]$
- LRC.- calidad.

LRC por cardinalidad  $\alpha [0,1] \leftarrow$  valor límite.

El valor límite es igual a:

$$\text{Valor límite} = \text{mínimo} + \alpha (\text{máximo} - \text{mínimo})$$

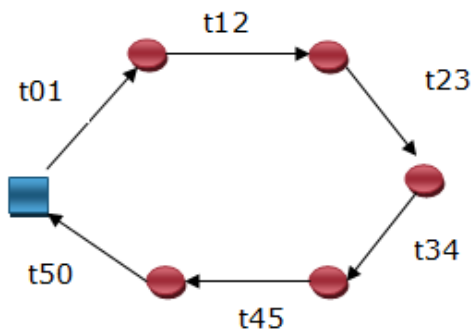
Donde  $\alpha$  definirá que tan grande estará la lista.

Si  $\alpha=0 \rightarrow$  El problema será de tipo Greedy.

Si  $\alpha=1 \rightarrow$  El problema será totalmente aleatorio.

También hablaremos de los TSP (ruteo)

Los cuales nos hablaran de ahorros en costo. Por ejemplo:



Ahorro en costo

$$\text{Min} = t01 + t12 + t23 + t34 + t45 + t50$$

Pero así como existe el TSP de ahorro en costo también existe en TSP de ahorro en tiempo (TSP CON LATENCIA) el cual para el grafico anterior seria este:

- 1 t01
- 2 t01 + t12
- 3 t01 + t12 + t23
- 4 t01 + t12 + t23 + t34
- 5 t01 + t12 + t23 + t34 + t45

Donde el mínimo será:

$$\text{Min} = 5(t_{01}) + 4(t_{12}) + 3(t_{23}) + 2(t_{34}) + t_{45}$$

Como se plantea anteriormente este proyecto consistirá en un problema GRASP en el cual intervendrán lo que es un TSP con LATENCIA para poder tomar lo que son los ahorros en tiempo.

## REVISION LITERARIA

- TSP con latencia

La latencia es un término que se utiliza para expresar el tiempo de espera en un sistema. En problemas de distribución representa el tiempo que espera un cliente para ser visitado y en problemas de programación de la producción, el tiempo de espera de una pieza para ser procesada. En problemas de redes, la latencia está asociada a la métrica utilizada para evaluar la longitud un camino que une un nodo inicial con un nodo dado de la red. El problema de Latencia mínima puede considerarse como una extensión del Problema del Agente Viajero (TSP), donde el objetivo es encontrar un camino Hamiltoniano que comience en el nodo inicial y minimice la suma de las latencias de todos los nodos.

A pesar de las similitudes que tiene con el TSP, es un problema que ha sido menos estudiado y es más difícil de resolver o aproximar desde un punto de vista computacional. Este problema tiene aplicaciones en el aprovisionamiento de ayuda humanitaria zonas de desastres, en la transportación de productos perecederos, en el procedimiento de búsqueda de información en redes informáticas, en la recepción de señales en redes inalámbricas de telecomunicaciones, en transporte de personal y en muchas otras áreas.

- Clarke-Wright

Este es otro tipo de problema el cual es muy parecido a los TSP CON LATENCIA el cual consiste en un algoritmo que también es conocido como método de los "ahorros" y se trata de un heurístico específico para resolver el problema de rutas de vehículos (VRP). Puede resolver el Capacitated VRP donde el número de recursos a utilizar es libre.

El método comienza con rutas que sólo están formadas por el depósito y un solo nodo. A cada paso del algoritmo se unen dos rutas si se genera un ahorro en tiempo/distancia.

## METODO DE SOLUCIÓN

GRASP

(Greedy Randomized Adaptive Search Procedure)

Es un algoritmo meta heurístico comúnmente aplicado a problemas de optimización combinatoria. Consiste en iteraciones hechas para crear una solución en un algoritmo probabilista y mejoras subsecuentes a través de una búsqueda local.

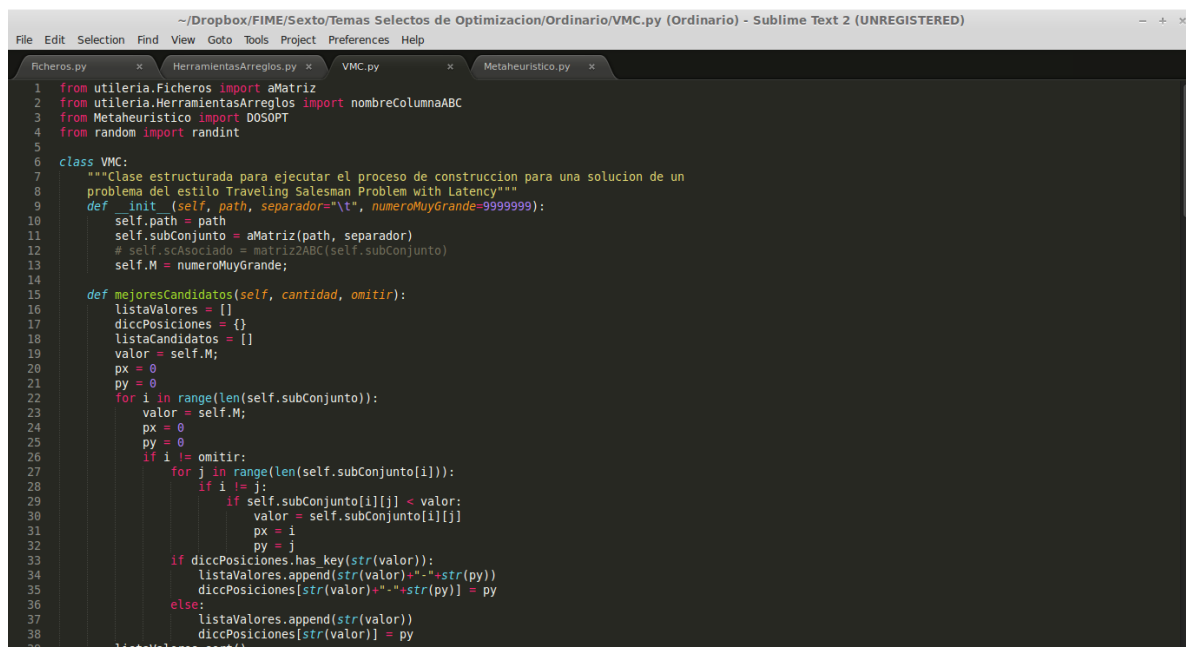
Las soluciones de algoritmo probabilista son generadas añadiendo elementos a la solución del problema desde una lista de elementos establecidos por una función probabilista.

Un algoritmo probabilista (o probabilístico) es un algoritmo que basa su resultado en la toma de algunas decisiones al azar, de tal forma que, en promedio, obtiene una buena solución al problema planteado para cualquier distribución de los datos de entrada. Es decir, al contrario que un algoritmo determinista, a partir de unos mismos datos se pueden obtener distintas soluciones y, en algunos casos, soluciones erróneas.

Existen varios tipos de algoritmos probabilísticos dependiendo de su funcionamiento, pudiéndose distinguir:

- Algoritmos numéricos, que proporcionan una solución aproximada del problema.
- Algoritmos de Montecarlo, que pueden dar la respuesta correcta o respuesta erróneas (con probabilidad baja).
- Algoritmos de Las Vegas, que nunca dan una respuesta incorrecta: o bien no encuentran la respuesta correcta e informan del fallo.

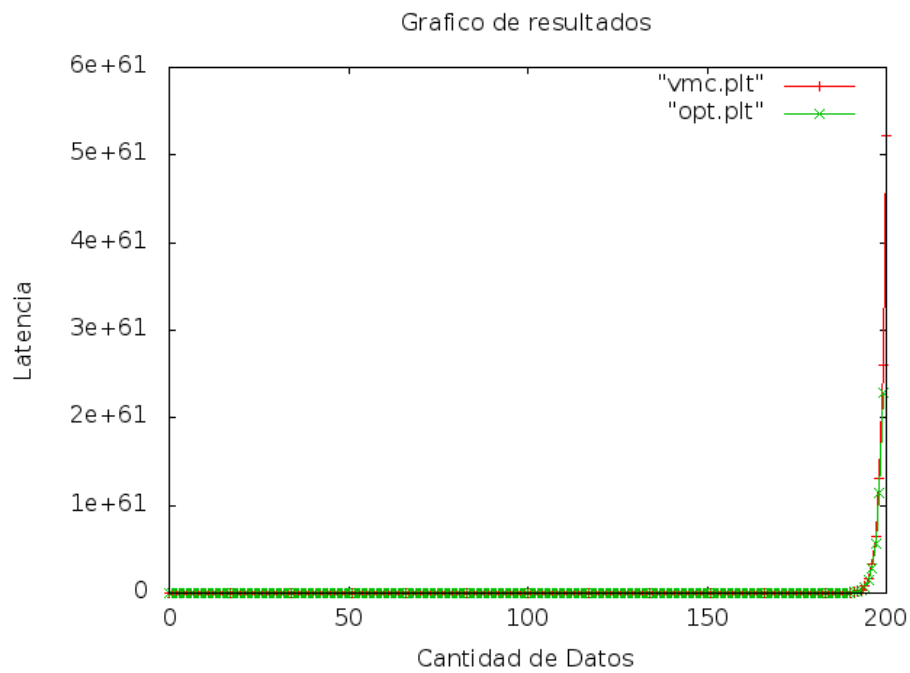
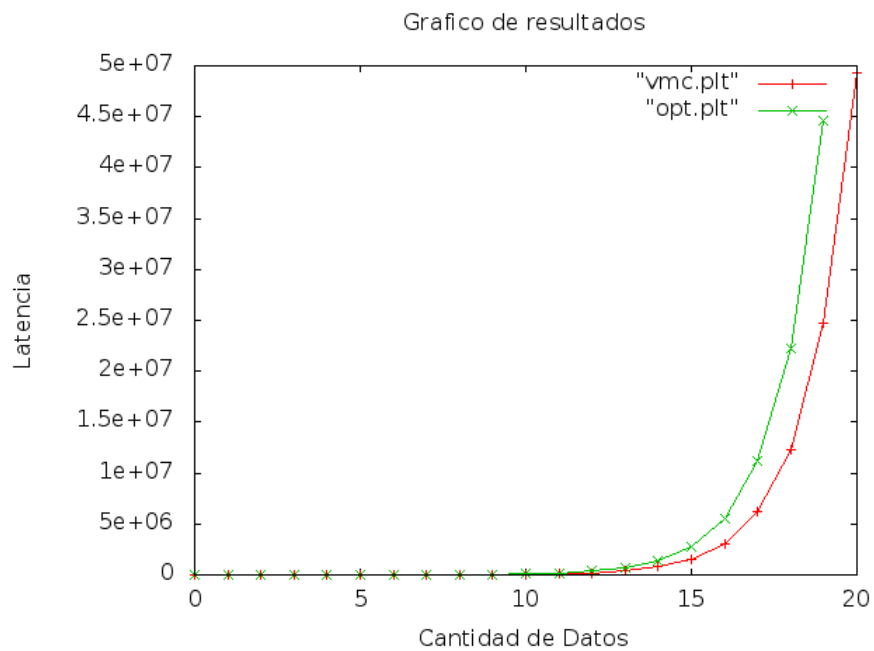
## EXPERIMENTACIÓN COMPUTACIONAL



```
~/Dropbox/FIME/Sexto/Temas Selectos de Optimizacion/Ordinario/VMC.py (Ordinario) - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Richeros.py x HerramientasArreglos.py x VMC.py x Metaheuristico.py x
1 from utileria.Ficheros import aMatriz
2 from utileria.HerramientasArreglos import nombreColumnaABC
3 from Metaheuristico import DOSOPT
4 from random import randint
5
6 class VMC:
7     """Clase estructurada para ejecutar el proceso de construccion para una solucion de un
8     problema del estilo Traveling Salesman Problem with Latency"""
9     def __init__(self, path, separador="\t", numeroMuyGrande=9999999):
10         self.path = path
11         self.subConjunto = aMatriz(path, separador)
12         # self.scAsociado = matriz2ABC(self.subConjunto)
13         self.M = numeroMuyGrande;
14
15     def mejoresCandidatos(self, cantidad, omitir):
16         listaValores = []
17         diccPosiciones = {}
18         listaCandidatos = []
19         valor = self.M;
20         px = 0
21         py = 0
22         for i in range(len(self.subConjunto)):
23             valor = self.M;
24             px = 0
25             py = 0
26             if i != omitir:
27                 for j in range(len(self.subConjunto[i])):
28                     if i != j:
29                         if self.subConjunto[i][j] < valor:
30                             valor = self.subConjunto[i][j]
31                             px = i
32                             py = j
33                 if diccPosiciones.has_key(str(valor)):
34                     listaValores.append(str(valor)+"-"+str(py))
35                     diccPosiciones[str(valor)+"-"+str(py)] = py
36                 else:
37                     listaValores.append(str(valor))
38                     diccPosiciones[str(valor)] = py
39         listaValores.sort()
```

```
~/Dropbox/FIME/Sexto/Temas Selectos de Optimizacion/Ordinario/Metaheuristico.py (Ordinario) - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Richeros.py x HerramientasArreglos.py x VMC.py x Metaheuristico.py x
1 from utileria.Ficheros import aMatriz
2 from utileria.HerramientasArreglos import nombreColumnaABC
3 from random import randint
4
5 class DOSOPT:
6     """Algoritmo de mejora 2 opt modificado para problemas con objetivo de latencia"""
7     def __init__(self, pathSolucion, matrizPesos):
8         self.soluciones = self.obtenListaSolucion(pathSolucion)
9         self.matrizPesos = matrizPesos
10
11     def obtenListaSolucion(self, path):
12         fichero = open(path)
13         lista = []
14         for linea in fichero:
15             lista.append(int(linea.strip()))
16         fichero.close()
17         return lista
18
19     def opt(self, antecesor, mov, sucesor, enlace, index):
20         wOriginal = self.matrizPesos[antecesor][mov]+self.matrizPesos[mov][sucesor]
21         wNueva = self.matrizPesos[antecesor][sucesor]+self.matrizPesos[sucesor][mov]
22         if wNueva < wOriginal:
23             self.soluciones[index] = sucesor
24             self.soluciones[index+1] = mov
25             # print "Entro: ",wNueva,"porque es menor que el peso anterior",wOriginal
26
27     def optMejorado(self, antecesor, mov, sucesor, enlace, index):
28         Olatencia = 0
29         for x in range(1,len(self.soluciones)):
30             Olatencia += self.matrizPesos[x-1][x]
31         self.soluciones[index-1] = antecesor
32         self.soluciones[index] = enlace
33         self.soluciones[index+1] = mov
34         self.soluciones[index+2] = sucesor
35         Nlatencia = 0
36         for x in range(1,len(self.soluciones)):
37             Nlatencia += self.matrizPesos[x-1][x]
38         if Nlatencia > Olatencia:
```

```
~/Dropbox/FIME/Sexto/Temas Selectos de Optimizacion/Ordinario/Metaheuristico.py (Ordinario) - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Richeros.py x HerramientasArreglos.py x VMC.py x Metaheuristico.py x
37         Nlatencia += self.matrizPesos[x-1][x]
38         if Nlatencia > Olatencia:
39             self.soluciones[index-1] = antecesor
40             self.soluciones[index] = mov
41             self.soluciones[index+1] = sucesor
42             self.soluciones[index+2] = enlace
43         Olatencia = 0
44
45     def mejora(self):
46         for i in range(2,len(self.soluciones)-2):
47             self.opt(self.soluciones[i-1], self.soluciones[i], self.soluciones[i+1], self.soluciones[i+2], i)
48             # self.imprime()
49             self.guardaResultado()
50
51     def imprime(self):
52         latencia = 0
53         for x in range(len(self.soluciones)-1):
54             latencia += self.matrizPesos[self.soluciones[x]][self.soluciones[x+1]]
55         print "Ruta: ",nombreColumnaABC(x), "=>", nombreColumnaABC(x+1), "posicion", self.soluciones[x], "latencia", latencia, "\n"
56
57     def guardaResultado(self):
58         path = raw_input("Nombre de archivo >")
59         archivo = open(path, "w")
60         plot = open(path+".plt", "w")
61         latencia = 0;
62         for i in range(1,len(self.soluciones)):
63             latencia += self.matrizPesos[self.soluciones[i-1]][self.soluciones[i]]
64         val = "Ruta: "+str(nombreColumnaABC(self.soluciones[i-1]))+"=>"+str(nombreColumnaABC(self.soluciones[i]))+" posicion "+str(self.soluciones[i])+" con u"
65         plot.write(str(latencia)+"\n")
66         archivo.write(str(val))
67         archivo.close()
68         plot.close()
69
```





## CONCLUSIONES

Al disminuir la latencia se hace más eficiente la entrega a los clientes, lo que optimiza el servicio. Esto es lo que tratamos de conseguir con nuestro método de solución.

Al resolver nuestro problema nos dimos cuenta que para avanzar alrededor del conocimiento actual sobre los avances en el TSP hay tendencia a generar híbridos de los métodos existentes. Esto aborda tanto los cruces entre metaheurísticas como la combinación de métodos de búsqueda global con la local.

Este heurismo se materializó, desde el punto de vista algorítmico, en que en algún momento de la práctica del vecino más cercano, el siguiente desplazamiento no se realizará a la ciudad inmediatamente cercana, disponible, sino que se renunciará a ella para trasladarse hacia la segunda más cercana disponible y, a partir de este cambio, se continuará con la tradicional regla vecino más cercano. Esta estrategia propuesta, atendiendo a una de las tendencias arrojadas por la revisión de literatura, llevó a complementarlo con una búsqueda local: el opt.

## **BIBLIOGRAFÍA**

[http://old.dii.uchile.cl/~gduran/docs/tesis/tesis\\_andres.pdf](http://old.dii.uchile.cl/~gduran/docs/tesis/tesis_andres.pdf)

<http://www.uv.es/~rmarti/paper/docs/heur1.pdf>

<http://arxiv.org/pdf/math/9409223.pdf>

<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.151.7058>

<http://yalma.fime.uanl.mx/~pisis/seminar/abstracts/2012a-bello.htm>

<http://www.iiia.csic.es/udt/en/ia/clarke-wright>