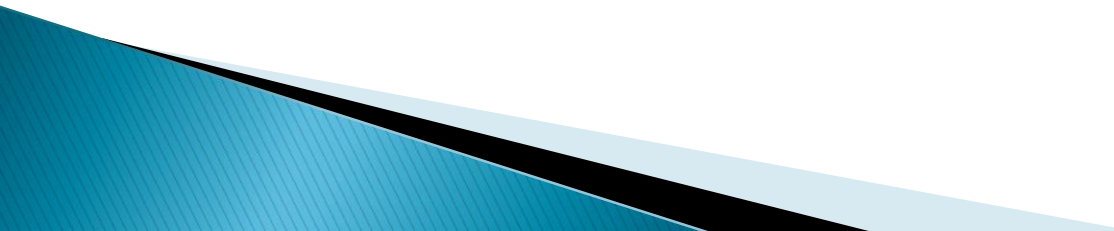


Práctica 4: Ramificación y poda (Branch and Bound)

Problema del viajante del comercio.

1.Recordatorio del problema viajante de comercio.


- ▶ Dado un conjunto de ciudades y una matriz con las distancias entre todas ellas:
 - Un viajante debe recorrer todas las ciudades exactamente una vez, regresando al punto de partida, de forma tal que la distancia recorrida sea mínima.
 - Más formalmente, dado un grafo G , conexo y ponderado, se trata de hallar el ciclo hamiltoniano de mínimo peso de ese grafo.
- 

2. Problema a realizar

- ▶ Buscar un algoritmo de ramificación o poda (Branch&Bound).



3. Introducción a Branch&Bound

- ▶ El esquema de Ramificación y Poda es el algoritmo de búsqueda más eficiente en espacios de búsqueda que sean árboles.
 - ▶ En cada momento hay una solución en curso.
 - ▶ La ramificación es óptima si el coste es menor que la solución en curso, en este caso esta última pasaría a ser la solución óptima en curso, en caso contrario sería podada.
 - ▶ El algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no están siendo óptimas, para que se <<pode>> esa rama del árbol y así no se malgastan recursos y procesos en casos que se alejan de la solución óptima.
 - ▶ Así se recorre todas las posibles soluciones hasta dar con la óptima
- 

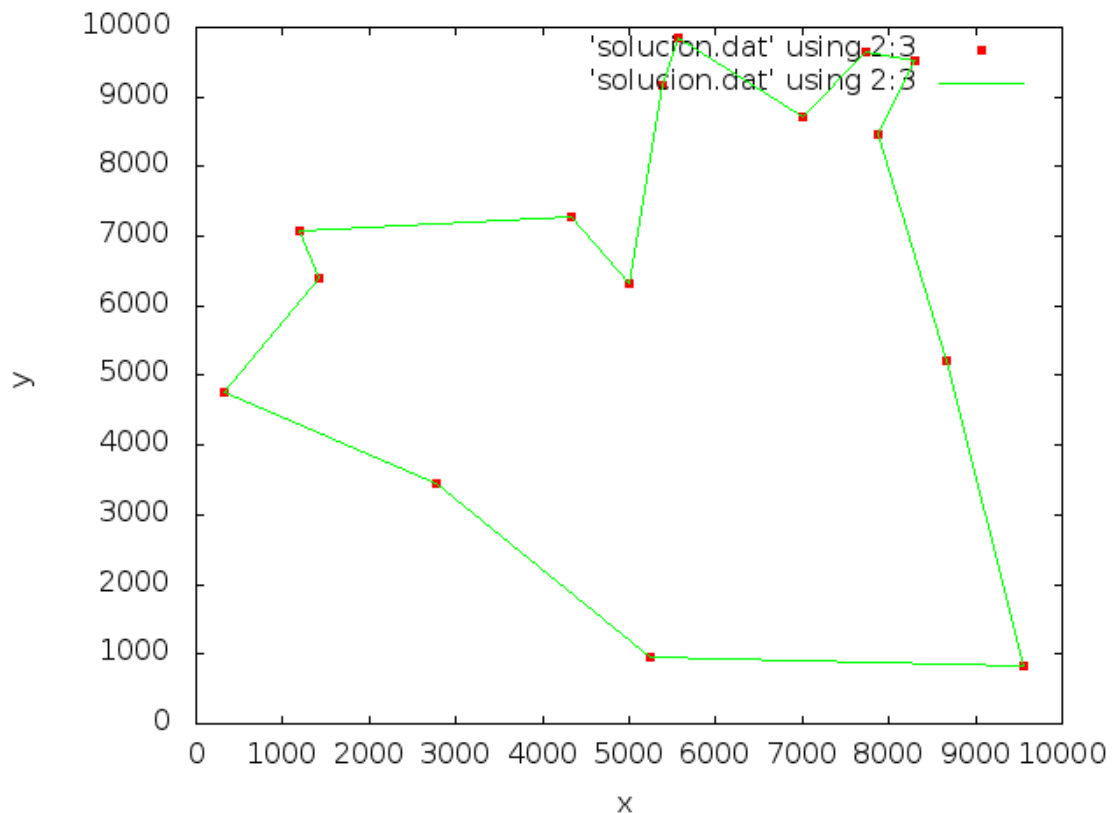
4. Solución al problema del viajante del comercio con Branch&Bound.

- ▶ Para la solución de esta práctica creamos una estructura llamada “nodo” la cual representará a cada ciudad, por lo que habrá tantas como número de ciudades les hayamos pasado a través del fichero, y los crea.
- ▶ Para realizar la poda, guardamos en todo momento en un vector de nodos el costo de la mejor solución obtenida hasta ahora (que se utiliza como cota superior global: la solución óptima debe tener un coste menor o igual a esa). Esa variable se inicializa con un algoritmo greedy, de la práctica anterior, contendrá la primera rama que explora.
- ▶ Con esto seguimos hacia delante, y calculamos la distancia absoluta del siguiente nivel, si ésta es menor que la cota anterior, expande este nodo y lo añade a nuestro vector de solución.
- ▶ En caso contrario, poda esa rama y se continúa con la siguiente.
- ▶ Podemos controlar los niveles en los que hemos estado mediante un campo “bandera” que nos indica si ese nivel ha sido visitado ya.

5. Ejemplos

- Para su representación hemos usado gnuplot. En este caso el número de ciudades será 15:

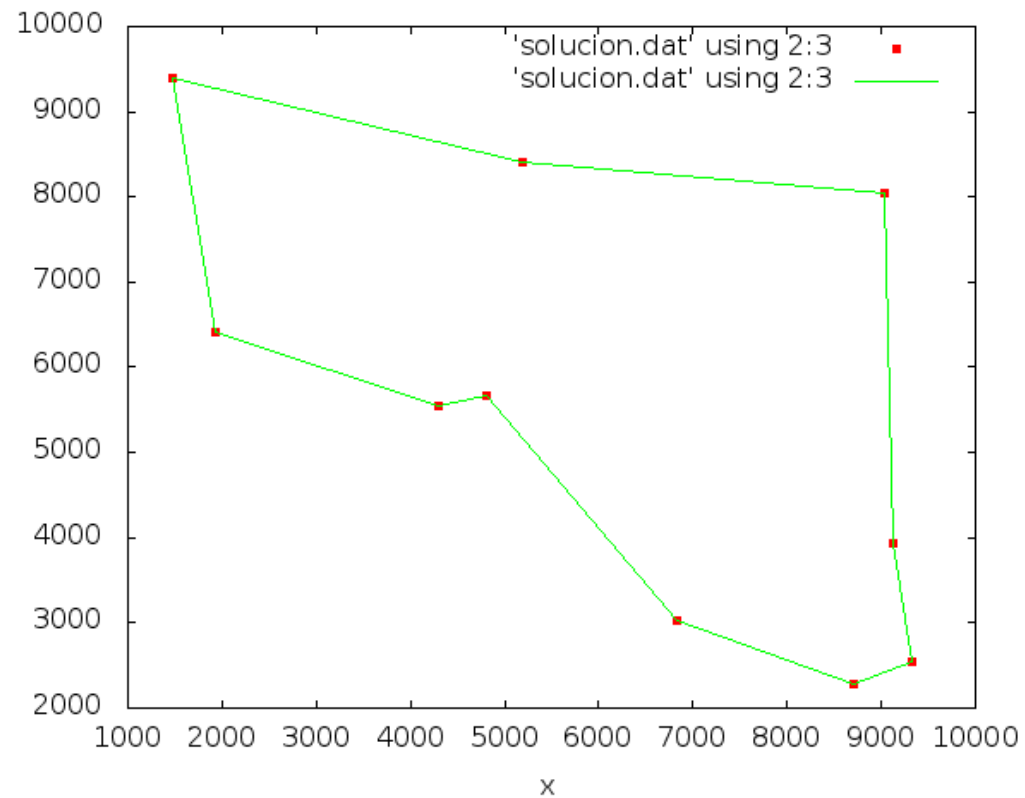
Ciudades	Valor eje X	Valor eje Y
1	8671	5217
8	7871	8460
13	8299	9533
4	7738	9656
5	7007	8718
2	5562	9844
7	5389	9174
12	5007	6325
3	4333	7286
9	1195	7085
6	1408	6409
14	324	4767
0	2782	3443



Ejemplo con número de ciudades igual a 10:

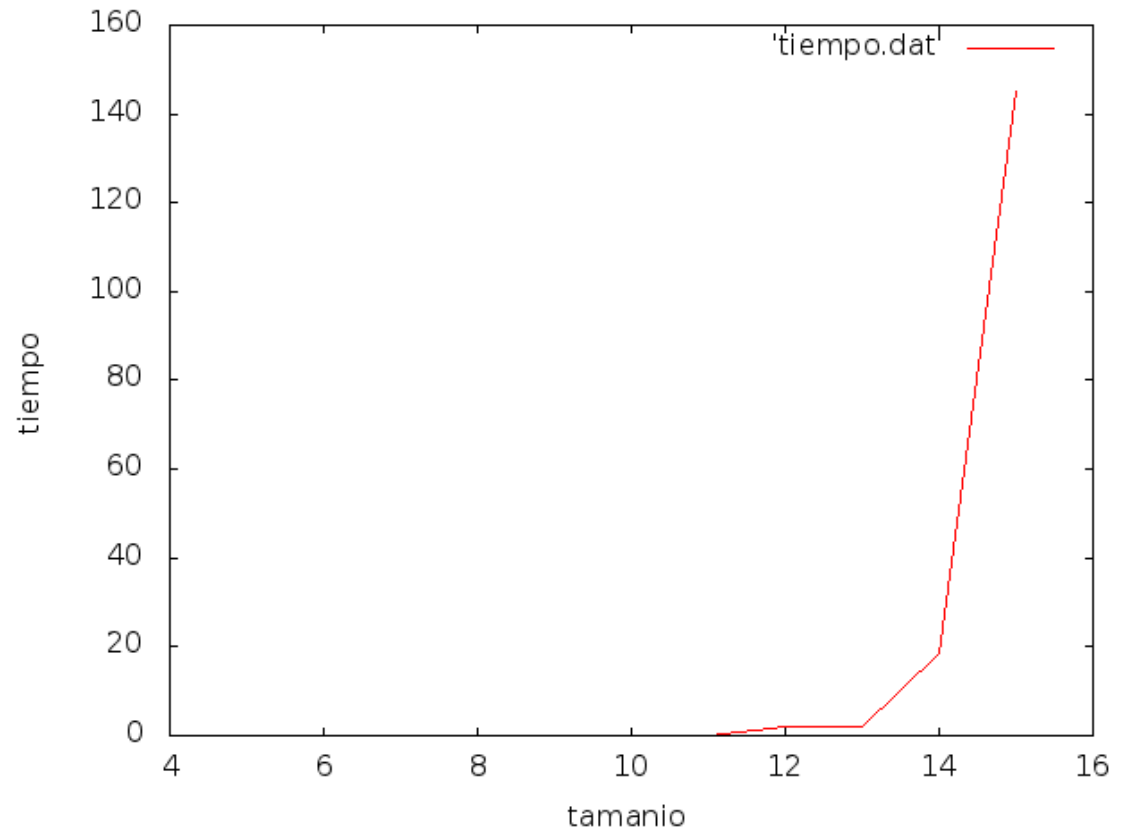
Ciudades	Valor eje X	Valor eje Y
3	1480	9402
5	1936	6415
4	4302	5554
8	4813	5660
7	6831	3021
9	8702	2278
6	9320	3932
2	9124	8043
0	9039	9402

>



6. Cálculo de la eficiencia empírica.

Tamaño	Tiempo
8	0.00281443
9	0.0159347
10	0.0306175
11	0.143857
12	1.93821
13	2.00506
14	18.651
15	144.896



7. Conclusión

- ▶ Como podemos comprobar cuando aumenta el tamaño el tiempo va creciendo, sobre todo al pasar el umbral de la cantidad de 14 ciudades, el tiempo crece de forma exponencial y rápidamente.