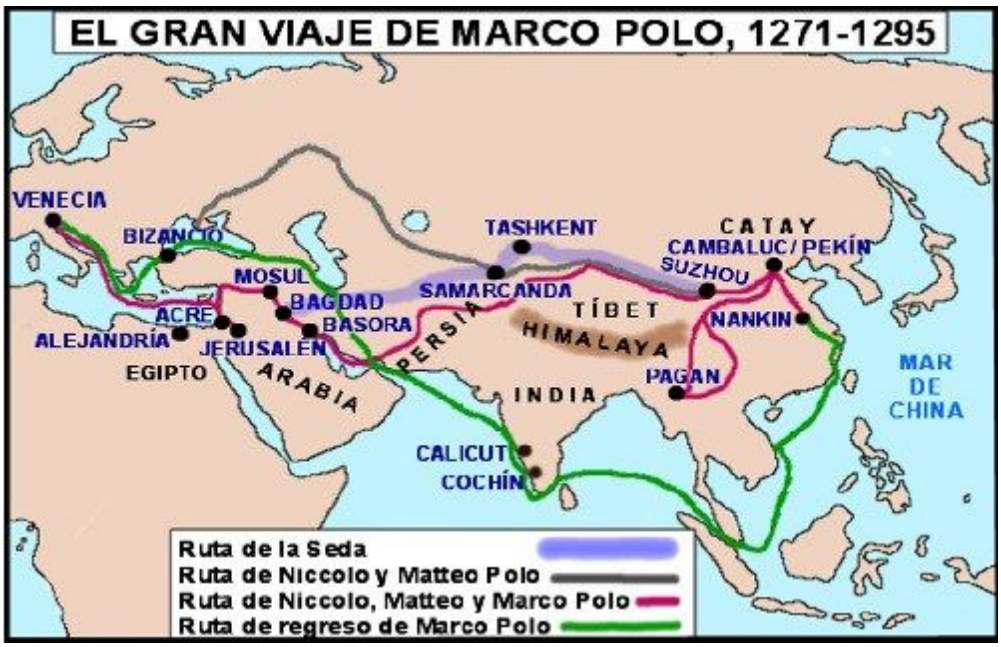
Práctica 4, parte 2: El viajante de comercio

Branch&Bound

Realizado por:
Sergio Cervilla Ortega
Daniel Díaz Pareja
Marina Hurtado Rosales
Adrián Morente Gabaldón
Joaquín Rodríguez Claverías



Índice

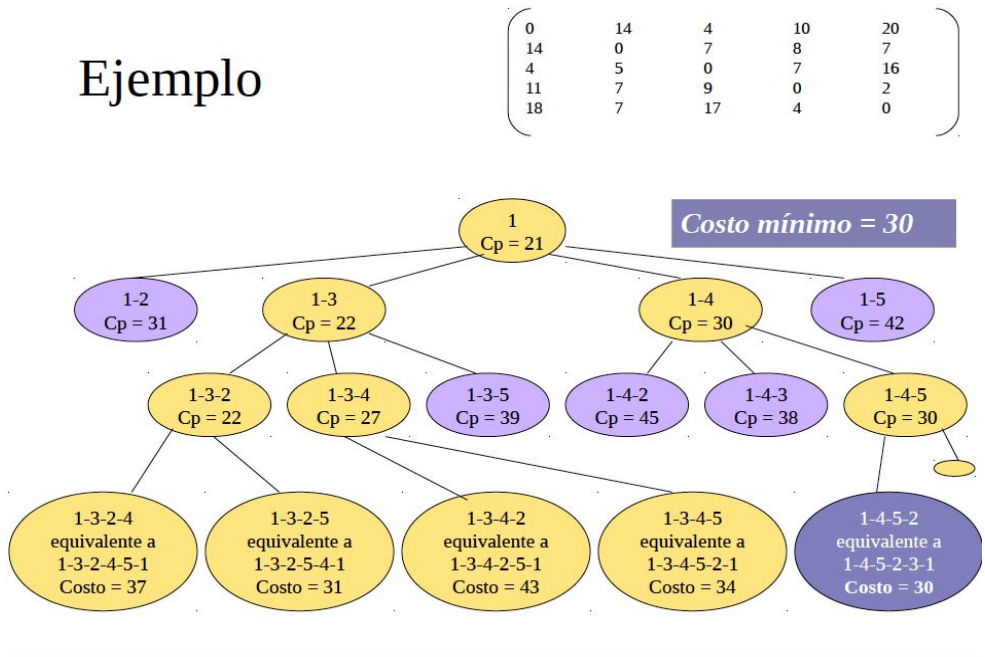
1. Definición del problema

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.

Para emplear un algoritmo de ramificación y poda como el que nos disponemos a usar, precisamos de una cota local inferior que nos guíe a la hora de tomar y/o desechar soluciones. Dicha cota consistirá en un valor menor o igual que el coste real de la mejor solución que podamos haber encontrado previamente; la cual se puede obtener a partir de la solución parcial en que nos encontremos.

La dinámica del algoritmo está basada en las transparencias de teoría (diapositiva 31 hasta 44) facilitando así la comprensión.



2. Partes del problema.

- **Solución parcial**: Será un vector de enteros donde se almacenarán las ciudades en el orden en que se van visitando, es decir, una permutación de n elementos.
- **Función de poda**: Si la cota local CL de una solución parcial es mayor a la cota global CG (que contiene el costo de la mejor solución obtenida hasta ahora), se poda el nodo que expande dicha solución parcial.

  La cota local se obtiene sumando las aristas de menor costo de las ciudades restantes a la solución parcial que llevamos. La cota global inicial en cambio, se calcula utilizando un algoritmo de tipo greedy con la heurística de la ciudad más cercana (expuesta y explicada en la práctica 3 parte 2).
- **Restricciones explícitas**: X[i] toma valores en {1, 2, 3, ..., n} donde n es el número de ciudades
- **Restricciones implícita**s: Una misma ciudad no puede aparecer dos veces en el recorrido.

3. Algoritmo B&B

El algoritmo que vamos a utilizar requiere un análisis específico de varias partes del código, que se adjuntará junto con éste documento.

Representamos los datos mediante un árbol de búsqueda, en el que cada nodo es una ciudad, cada arista es la distancia que hay entre las ciudades que une, y donde cada nodo se expande para posibilitar la exploración de los demás nodos.

Para saber qué nodo vamos a expandir se emplea el criterio del "más prometedor". En este caso consideraremos como nodo más prometedor aquel que presente el menor valor de cota local.

Para encontrar una solución creamos una clase con ese mismo nombre, la clase *Solución*, que contendrá un vector de ciudades ordenadas según el procedimiento branch & bound.

Para calcular una solución usamos un algoritmo branch&bound. ¿En qué consiste este algoritmo?

En primer lugar necesitaremos una matriz con las distancias que unen las ciudades, la cual se calcula con la función *calcular_matriz*; un vector donde se almacenan los caminos más cortos entre ciudades, el cual se obtiene con la función *calcularMenorArista*; y el número de ciudades que se van a visitar. También son necesarios un vector donde

almacenaremos las ciudades restantes, y una cola con prioridad, donde se guardaran los nodos ya generados (nodos vivos).

Partimos de una solución inicial, calculada mediante la función *recorridoGreedy*. En esta función, lo primero que hacemos es introducir el primer nodo en la solución, se compara con todas las restantes, y nos quedamos con la de menor distancia. Esta heurística es una de las que teníamos en la segunda parte de la práctica 3, heurística de la ciudad más cercana. . Gracias a este movimiento obtenemos la cota global, que es la distancia de la secuencia obtenida mediante un algoritmo greedy.

A continuación, la parte más importante, comenzamos un bucle que se mantiene mientras que queden nodos vivos y la cota local sea menor que la cota global. Actualizamos el nodo a expandir con el primer nodo que haya en la cola, a su vez lo eliminamos de la cola de nodos ya generados y actualizamos las ciudades restantes.

Seguidamente, mientras queden ciudades, añadimos una posible ciudad a la solución parcial y comprobamos si con eso obtenemos una solución. Si es así, comprobamos si la distancia que tenemos acumulada es mejor que la cota global. La distancia actual se obtiene con el método *Evalua* (que nos proporciona la distancia total entre la primera y la última ciudad). En caso de que la distancia actual sea menor que la cota global, actualizamos la cota global y la mejor solución. La cota global se actualiza a la nueva distancia proporcionada por *Evalua* y la mejor solución al nuevo recorrido. En caso que no sea una mejor solución, es decir si la cota local es menor que la cota global, la ignoramos y expandimos el siguiente nodo.

Si no hemos obtenido una posible solución, comprobamos si la cota local es menor que la cota global. En caso afirmativo, continuamos explorado. En caso negativo, podamos.

Por último solo nos queda descartar la última ciudad que se acaba de añadir y contar con la última ciudad que se añadió.

Una consideración importante en el código es que en ejecución se contarán las podas, los nodos expandidos y el máximo de nodos vivos, para llevar un mayor control de la información que maneja el algoritmo.

4. Resultados

Creemos necesario indicar que los ficheros de datos sobre los que se ha trabajado han sido creado por nosotros a partir de los originales aportados con la práctica, y es que el número inicial de ciudades incluidas en dichos archivos, saturaban la memoria de nuestros computadores a la hora de poner en práctica nuestro algoritmo. Así nuestros ficheros serán aquellos sobre los que hemos ejecutado y obtenido resultados (se mantiene el formato de cada fichero según los aportados inicialmente).

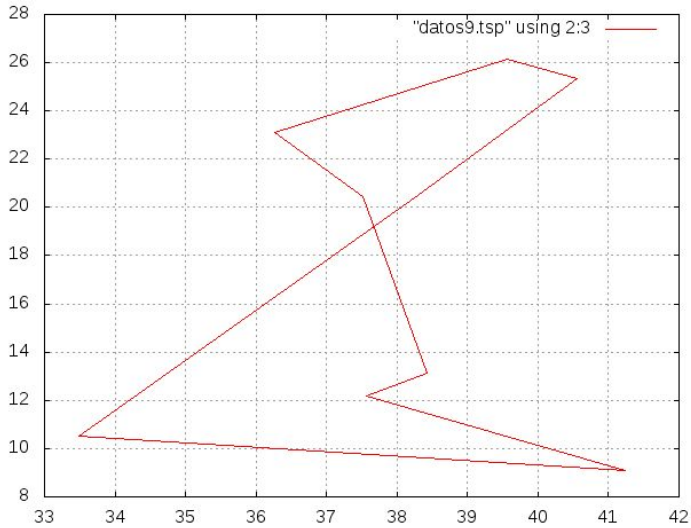
A continuación se pueden ver los ejemplos de rutas de ciudades:

	ciudad9.tsp		ciudad13.tsp		ciudad14.tsp	
1	38.24	20.42	38.24	20.42	38.24	20.42
2	39.57	26.15	39.57	26.15	39.57	26.15
3	40.56	25.32	40.56	25.32	40.56	25.32
4	36.26	23.12	36.26	23.12	36.26	23.12
5	33.48	10.54	33.48	10.54	33.48	10.54
6	37.56	12.19	37.56	12.19	37.56	12.19
7	38.42	13.11	38.42	13.11	38.42	13.11
8	37.52	20.44	37.52	20.44	37.52	20.44
9	41.23	9.10	41.23	9.10	41.23	9.10
10			41.17	13.05	41.17	13.05
11			36.08	-5.21	36.08	-5.21
12			38.47	15.13	38.47	15.13
13			38.15	15.35	38.15	15.35
14					37.51	15.17

Hemos ejecutado nuestro algoritmo B&B para estos tres conjuntos de datos, con los resultados siguientes:

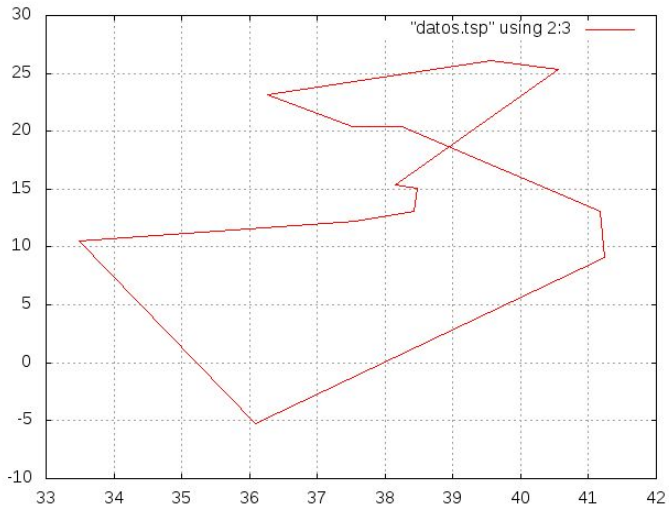
Para ciudad9.tsp:

```
$ g++ B\&B_FINAL.cpp -o main -O2
$ ./main datos_9.tsp
EL numero de podas es : 1523
El numero de nodos expandidos es: 6952
El maximo de nodos vivos es: 2037
1 38.24 20.42
3 40.56 25.32
2 39.57 26.15
4 36.26 23.12
8 37.52 20.44
7 38.42 13.11
6 37.56 12.19
9 41.23 9.1
5 33.48 10.54
1 38.24 20.42
Distancia: 41
```



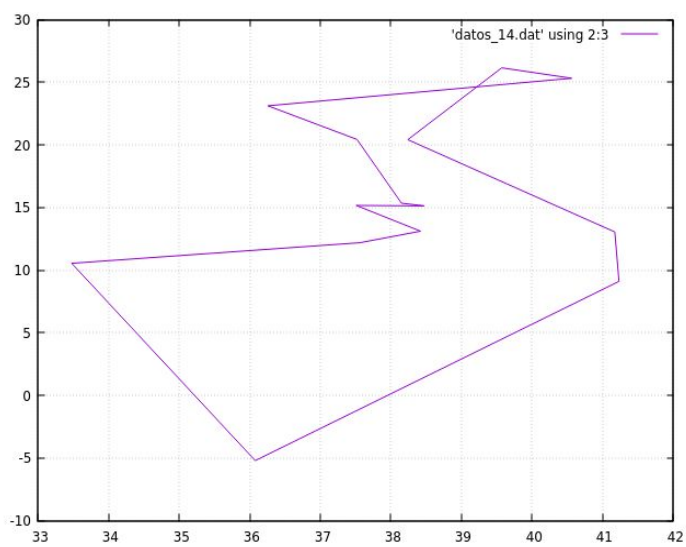
Para ciudad13.tsp:

```
$ ./main datos_13.tsp
EL numero de podas es : 323464
El numero de nodos expandidos es: 3968080
El maximo de nodos vivos es: 1292532
1 38.24 20.42
8 37.52 20.44
4 36.26 23.12
2 39.57 26.15
3 40.56 25.32
13 38.15 15.35
12 38.47 15.13
7 38.42 13.11
6 37.56 12.19
5 33.48 10.54
11 36.08 -5.21
9 41.23 9.1
10 41.17 13.05
1 38.24 20.42
Distancia: 64
```



Para ciudad14.tsp:

```
$ ./main datos_14.tsp
EL numero de podas es : 1720983
El numero de nodos expandidos es: 23940784
El maximo de nodos vivos es: 7271935
1 38.24 20.42
2 39.57 26.15
3 40.56 25.32
4 36.26 23.12
8 37.52 20.44
13 38.15 15.35
12 38.47 15.13
14 37.51 15.17
7 38.42 13.11
6 37.56 12.19
5 33.48 10.54
11 36.08 -5.21
9 41.23 9.1
10 41.17 13.05
1 38.24 20.42
Distancia: 64
```



Nótese que al principio de cada ejecución aparece el número de podas, los nodos expandidos y el máximo de nodos vivos. La secuencia de ciudades visitadas se aprecia leyendo la primera cifra de cada fila. Aparece el nodo de inicio dos veces para completar el ciclo.