Algoritmica
Practica 4. TSP B&B Parte 2
2018-2019

UNIVERSIDAD
DE GRANADA

Félix Ramírez García
Raúl Del Pozo Moreno
Cristian Piles Ruiz
Oleksandr Kudryavtsev
Sixto Coca Cruz

INDICE

1. Introducción

El problema del viajante de comercio ya se ha comentado y utilizado en la práctica sobre algoritmos voraces, donde se estudiaron métodos de este tipo para encontrar soluciones razonables (no óptimas necesariamente) a este problema. Si se desea encontrar una solución óptima es necesario utilizar métodos más potentes (y costosos), como la ramificación y poda, que exploren el espacio de posibles soluciones de forma más exhaustiva.

2. Planteamiento

Para la resolución de este problema se almacenan en memoria durante la ejecución las siguientes variables globales, para poder ser accedidas desde cualquier método:

```
int numeroNodos = 0;     // Almacena el numero de nodos visitados
int numeroPodas = 0;     // Almacena el numero de podas realizadas
int maxCola = 0;         // Almacena el tamaño maximo alcanzada de la cola con prioridad
duration<double> transcurrido;  // Almacena el tiempo transcurrido
double cotaGlobal;              // Almacena el coste hasta el nodo hoja
vector<Ciudad> ciudades_fich;   // Almacena las ciudades con sus coordenadas
int cantidadCiudades = 0;       // Almacena la cantidad de ciudades leidas
```

La estructura tipo Ciudad contiene un parámetro para el índice, y otros dos para los parámetros x e y. Esta estructura se usa para almacenar la lista de ciudades leídas desde el fichero.

```
// Estructura que almacena informacion de una ciudad
struct Ciudad {
    int indice;  // Almacena el indice de la ciudad
    double x;    // Almacena coordenada x
    double y;    // Almacena coordenada y
} ciudad;
```

La estructura tipo NodosVivos contiene un parámetro para controlar el nivel actual del nodo, otro para almacenar la cotaLocal optimista de ese nodo, y dos vectores para almacenar las ciudades visitadas y no visitadas hasta el momento. Esta estructura se usa para representar el conjunto de nodos que genera el programa.

```
// Estructura que almacena informacion de un nodo
struct NodosVivos {
    int nivelNodo;           // Almacena el nivel al que corresponde el nodo
    double cotaLocal;        // Almacena la cota local del nodo
    vector<int> sinVisitar;  // Almacena la lista de ciudades sin visitar
    vector<int> visitadas;   // Almacena la lista de ciudades visitadas
};
```

Al empezar el programa cargamos en un vector global (ciudades_fich) los datos de localización obtenidos desde el fichero y para emplear un algoritmo de ramificación y poda es necesario utilizar una cota inferior: un valor menor o igual que el verdadero coste de la mejor solución (la de menor coste) que se puede obtener a partir de la solución parcial en la que nos encontremos.

Ya que en todo momento conocemos las ciudades que faltan por visitar, una estimación optimista para cada nodo será la suma de la distancia que lleva recorrida más la distancia mínima del circuito multiplicada por el número de ciudades sin visitar. (PDF backparte2.pdf diapositiva 58 [Cota1]).

Para realizar la poda, guardamos en una variable global 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 el costo de la solución obtenida utilizando el algoritmo voraz implementado para la solución de la práctica anterior mediante el Vecino Más Cercano. Si, para una solución parcial, su cota inferior es mayor que el costo de la mejor solución actual, entonces se puede realizar la poda.

Como criterio para seleccionar el siguiente nodo a expandir en la posible solución, se utiliza una cola con prioridad con el criterio LC o “más prometedor”. Así, se ha sobrecargado el operador () de forma que los nodos con una cotaLocal inferior estén en el tope de la cola. (PDF backparte3.pdf diapositiva 7)

```
class ComparadorNodosCola {
    public:
        bool operator()(const NodosVivos nodoA, const NodosVivos nodoB) {
            return nodoA.cotaLocal > nodoB.cotaLocal;
        }
};
```

```
// Declaracion cola con prioridad de nodos vivos
priority_queue<NodosVivos, vector<NodosVivos>, ComparadorNodosCola> colaNodos;
```

Se trata de partir de la primera ciudad, que coincide con la raíz del árbol de exploración, y empezar el proceso de expansión. Para todas las ciudades restantes no visitadas se genera un hijo que incluye el número (nivel) de la ciudad, el recorrido realizado hasta el momento, el conjunto de las ciudades que faltan por seleccionar y la cota local. Si hemos recorrido todas las ciudades, es decir, el nivel en el que estamos corresponde con el número de ciudades que se quieren explorar (nodo hoja), calculamos la distancia total acumulada. Si dicha distancia es mejor que la cota global, actualizamos dicho valor y guardamos la solución.

```
// Si el nodo generado es nodo hoja
if(nodoHijo.nivelNodo == cantidadCiudades-1) {
    // Si la cota mejora, es solucion
    if(costeTotal(nodoHijo.visitadas) <= cotaGlobal){
        cotaGlobal = nodoHijo.cotaLocal;
        optimo = nodoHijo;
        optimo.visitadas.push_back(optimo.visitadas[0]);
    }
}
// Si no es hoja pero mejora la cota
else if(nodoHijo.cotaLocal < cotaGlobal) colaNodos.push(nodoHijo);
// Si no, se poda (no hace nada)
else numeroPodas++;
```

En el caso de queden ciudades por visitar, se comprueba la condición de poda y ramificación. Para este proceso, comparamos la cota local del nodo actual con la cota global, mejor costo encontrado hasta ese momento, y, en el caso de que sea superior realizamos la poda, si no, guardamos el nodo en la cola con prioridad de nodos por explorar. Mientras no se haya alcanzado la solución, vamos sacando el primer nodo de la cola y repetimos el proceso de expansión.

3. Elementos de la solución del problema

Conjunto Candidatos	Todas las ciudades
Conjunto Seleccionados	Las ciudades no visitadas
Restricciones implícitas	No se puede pasar 2 veces por la misma ciudad
Restricciones explícitas	El vector que almacena las ciudades toma valores de 1 a N
Función Solución	Ruta con distancia óptima entre ciudades
Función Factibilidad	La cota local optimista en un nodo hoja es mejor que la cota global.
Función Selección	La cota local optimista en un nodo es mejor que la cota global.
Función Objetivo	Obtener la ruta entre las ciudades óptima

a. Cotas

Cota global: Al iniciar el algoritmo se obtiene con la solución del vecino más cercano usada en la práctica de algoritmos voraces. Una vez inicializada, se va actualizando en cada iteración del algoritmo cuando llega un nodo hoja y el coste total mejora respecto a la cota de la ultima solucion.

Cota local: Para cada nodo se calcula su cota local optimista, que es la suma de la distancia recorrida hasta ese nodo más la multiplicación de la distancia mínima entre todas las ciudades por el número de ciudades que queden por visitar (cota 1 de las diapositivas de backtracking parte2).

b. Criterio de poda

Para podar se utiliza el criterio de que el coste de la solución obtenida en el nodo hoja, sea peor que el coste de la solución obtenida anteriormente (la primera comparación con coste de VMC). En el caso de que el nodo no sea un nodo hoja, comprueba si mejora el coste de ese nodo, en caso de no mejorar, se poda.

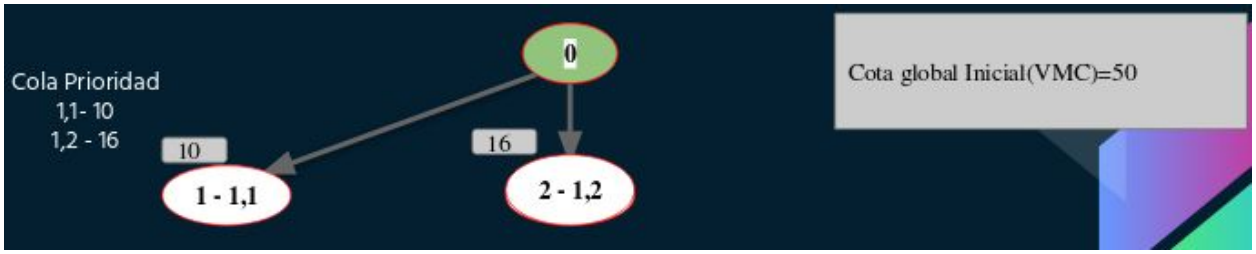
4. Pseudocodigo

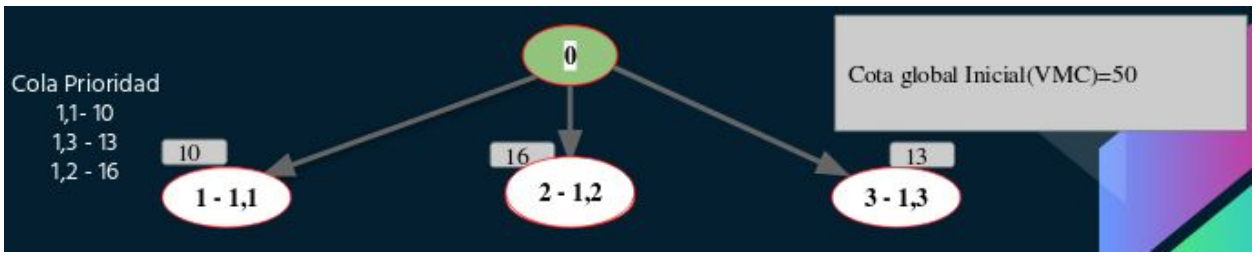
```
Se crea el primer nodo del árbol (raiz).
    Se obtiene la cota local (optimista) del nodo padre.
    Se añade el nodo padre a la cola con prioridad.
    Mientras haya nodos en la cola con prioridad y mejore el coste.

        Se obtiene el nodo del tope de la cola (mejor cotaLocal , más prometedor)
        Se aumenta el contador de nodo expandido
        Se elimina el nodo del tope de la cola

        Para cada ciudad no visitada del nodo

            Se genera un hijo del nodo
            Se obtiene la cota local (optimista) del nodo hijo
            Si el nodo generado es nodo hoja
                Si la cota mejora, es solucion

                    Si es hoja y mejora la cota
                            Se actualiza la solución
                    Si no es hoja y mejora el coste
                            Se añade el nodo a la cola
                    Si no, se poda (no sigue explorando ese nodo)
```

5. Ejemplo de árbol recorrido para 4 ciudades

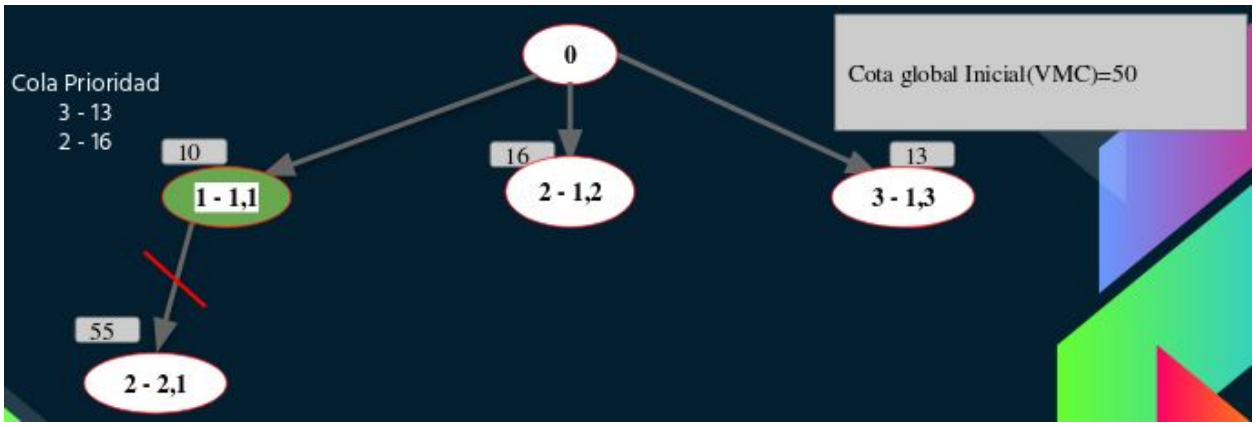
Se genera el nodo padre 0, se crea su primer hijo y se calcula su cota local (10), como 10 <= 50 y no es nodo hoja se añade a la cola con prioridad

Igualmente se genera su segundo y tercer hijo, ya que ninguno excede la cota global

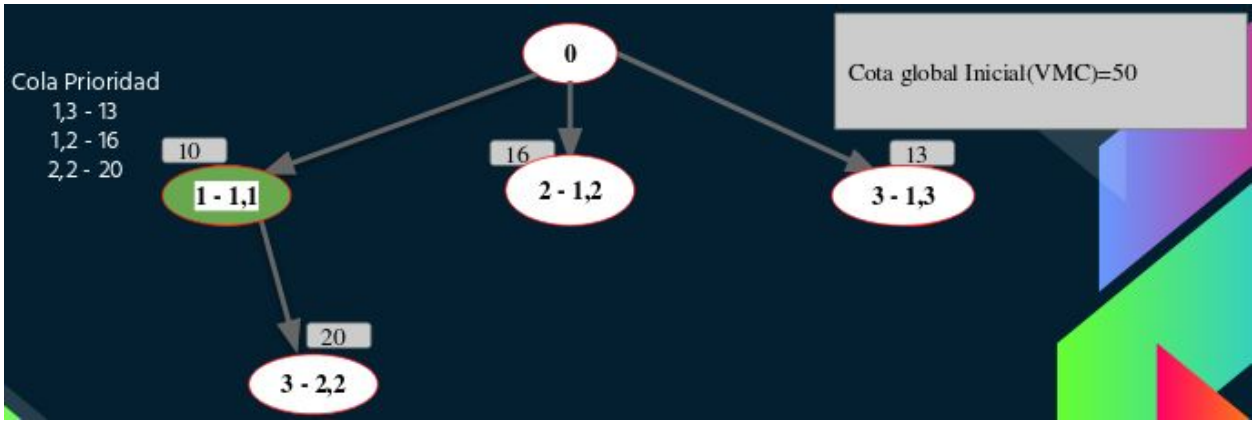




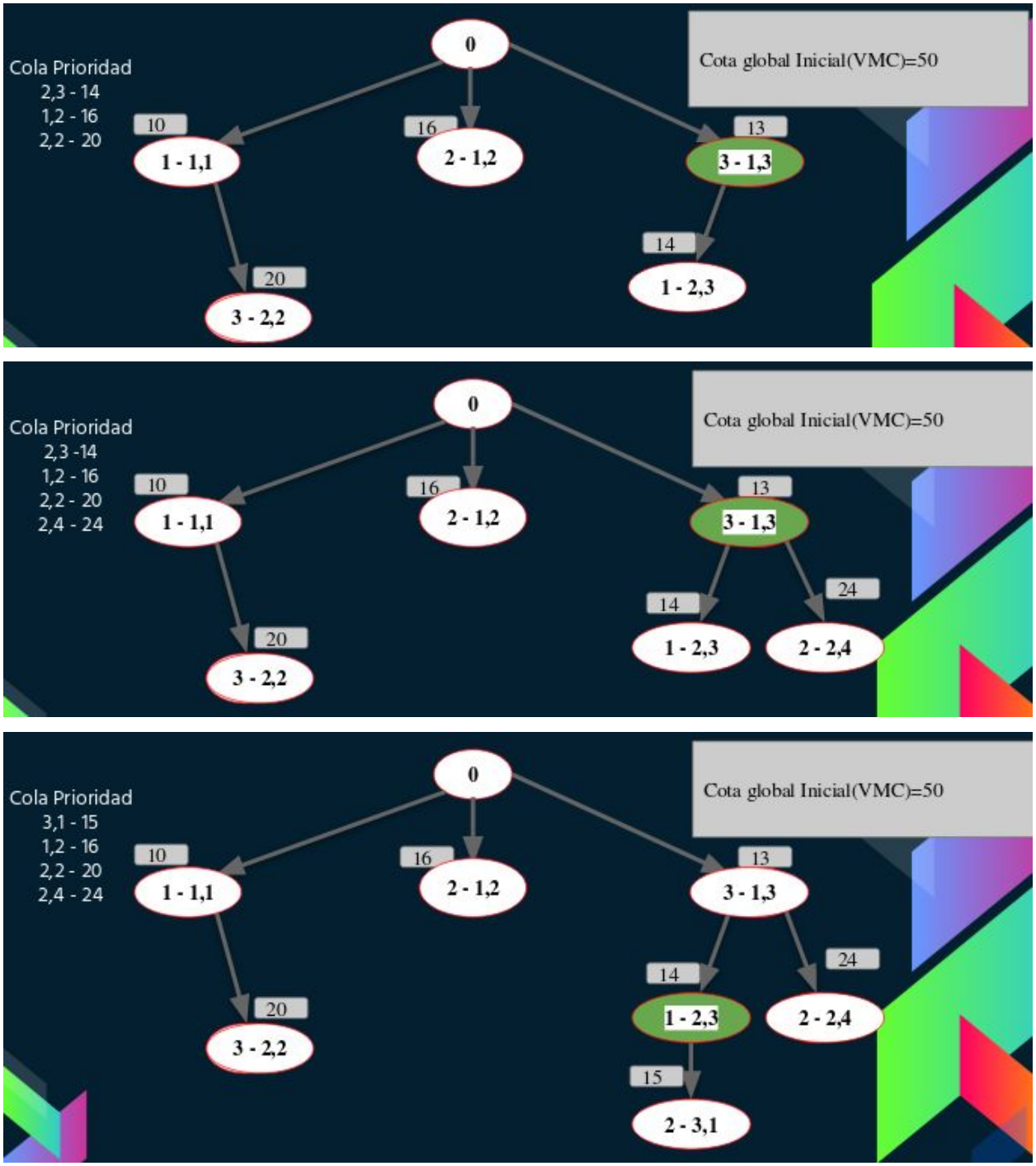
Una vez que ha generado todos los hijos, obtiene el hijo más prometedor según la cola de prioridad (tope), en este caso el nodo {1,1}, el cual, se elimina de la cola al seleccionarse.
Se generan sus 2 hijos (no visitados).
Como la cota del nodo hijo {2,1} excede a la cota global, se poda.



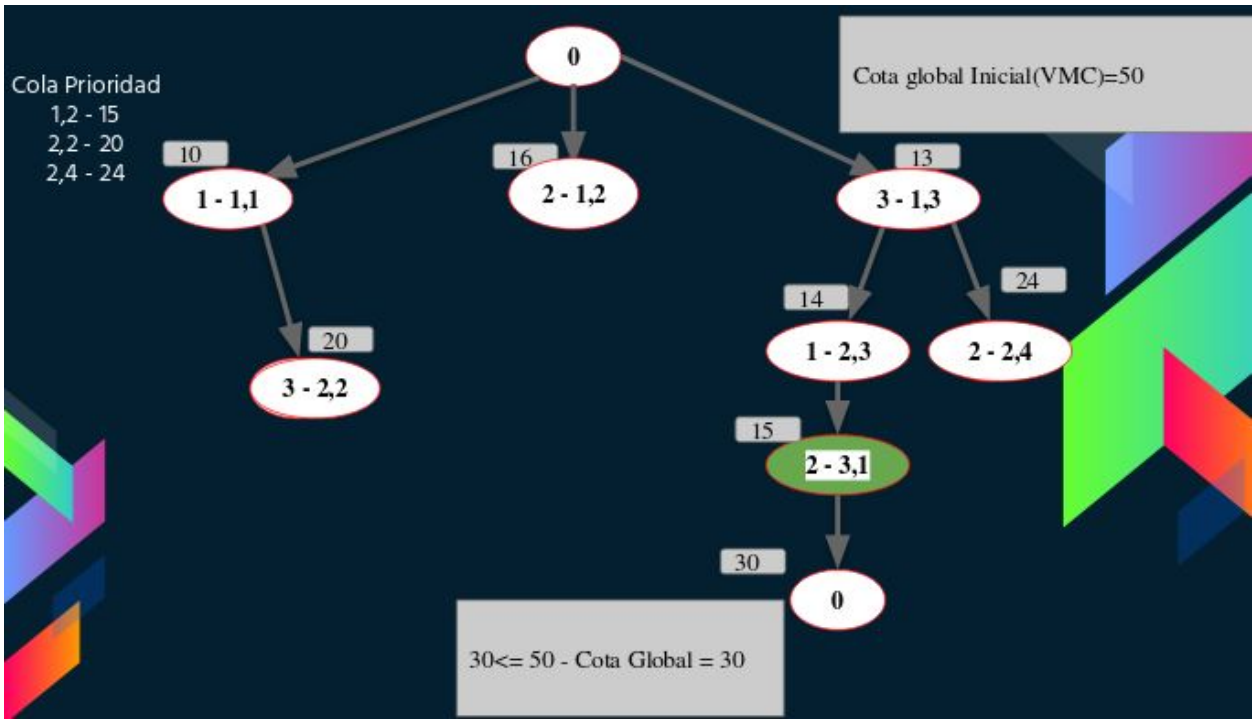
Se genera el segundo hijo, cuya cota no excede la cota global y se añade a la cola con prioridad.



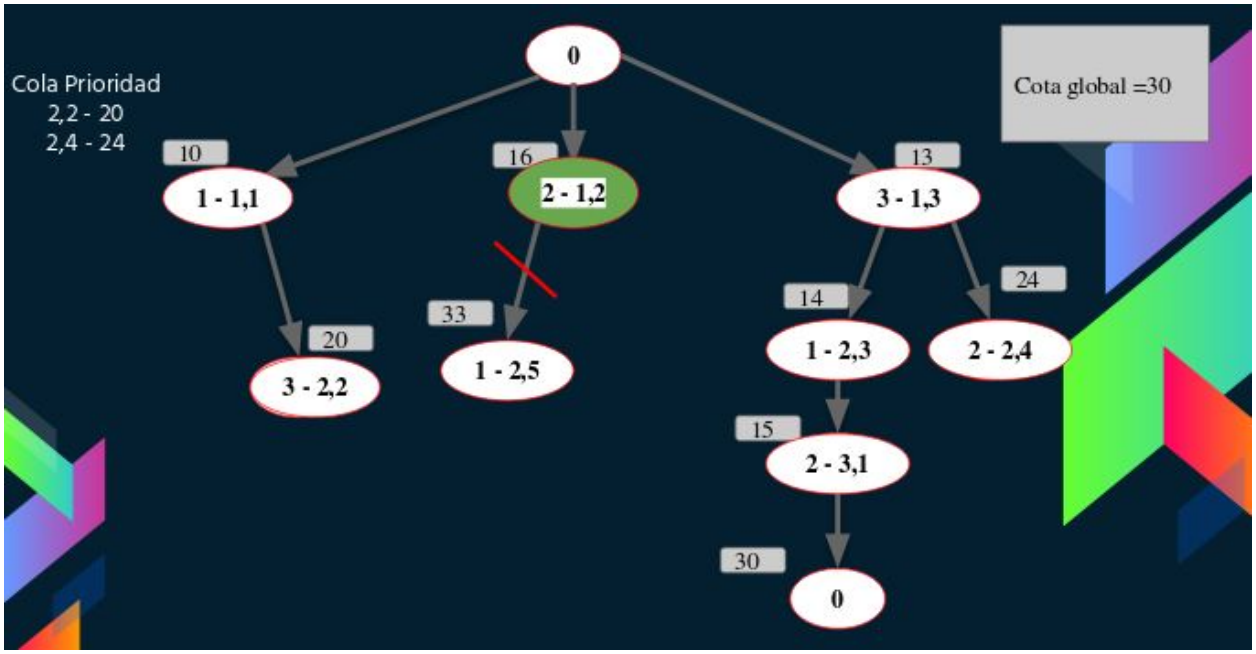
Se selecciona el nodo más prometedor de la cola y se elimina, se genera su primer hijo y se añade a la cola. Esto se repite hasta alcanzar el nodo hoja de esta rama.

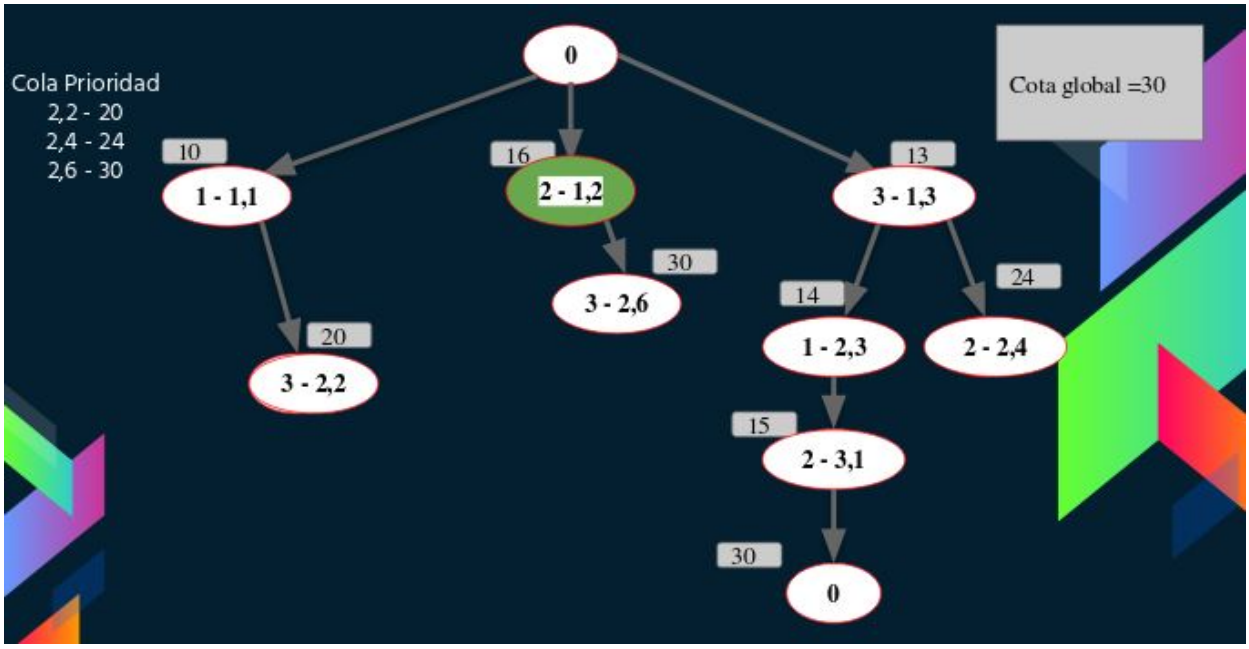


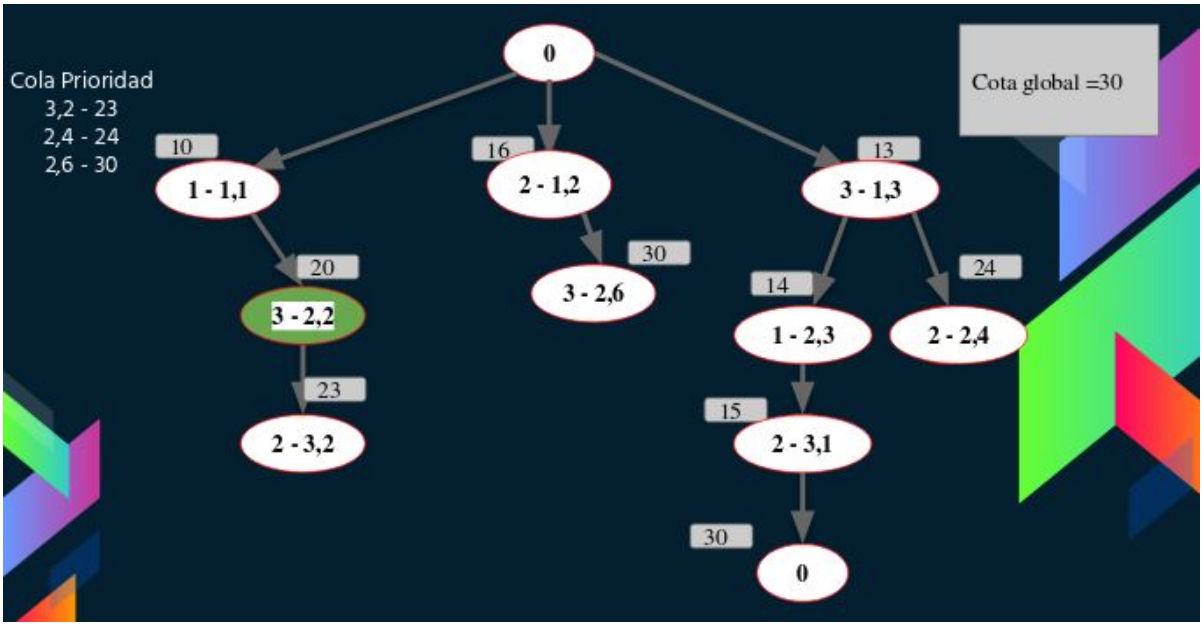
Una vez alcanzado el nodo hoja, se comprueba si el coste total de la rama (volviendo al origen) mejora la cota global actual, como la mejora, esta rama pasa a ser la solución actual y se actualiza la cota global con el coste de dicha solución.



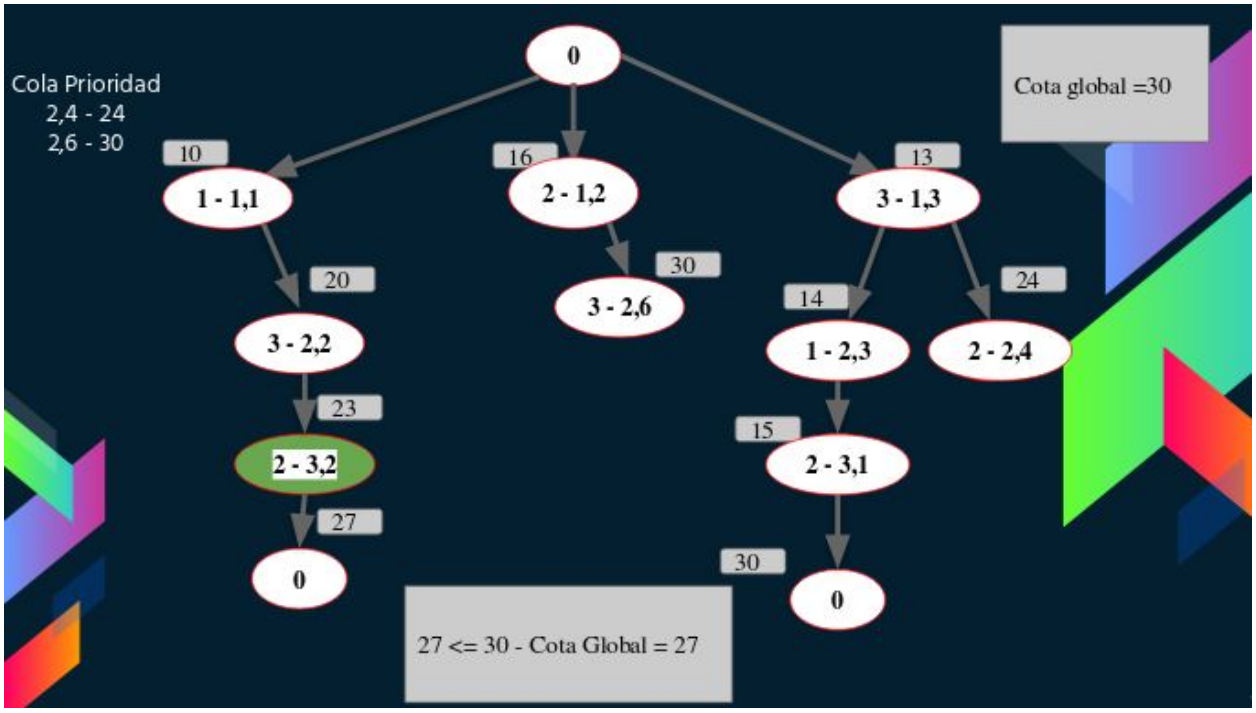
Se sigue con el siguiente nodo más prometedor de la cola. Este proceso se repite hasta que no queden más nodos en la cola.

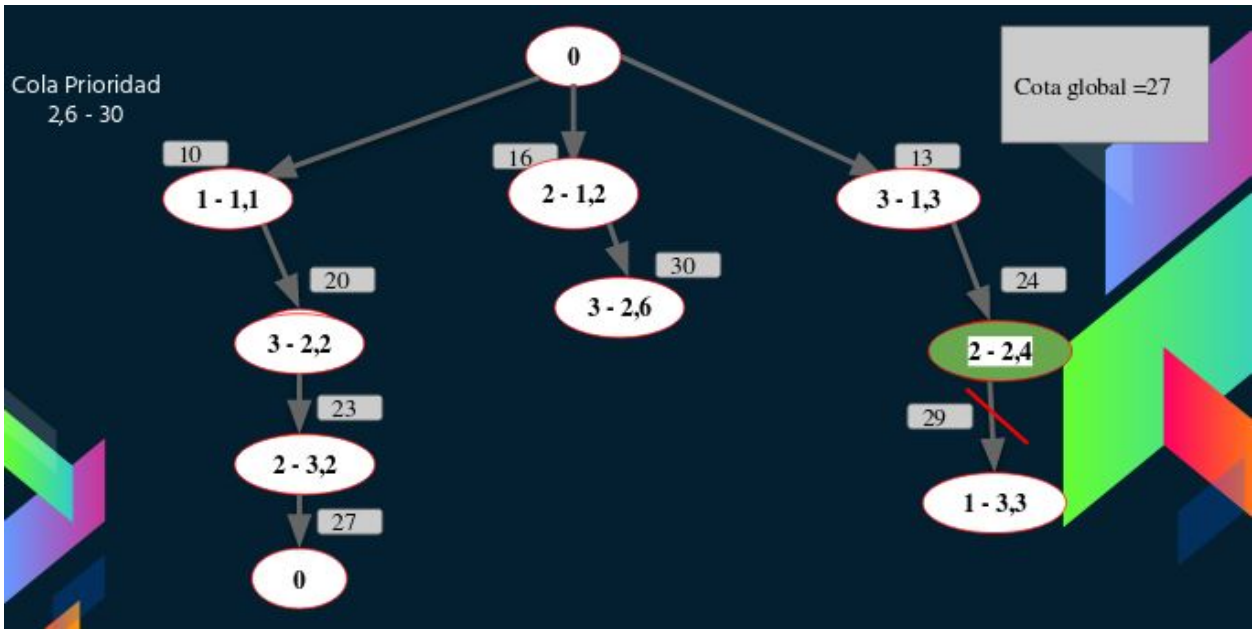




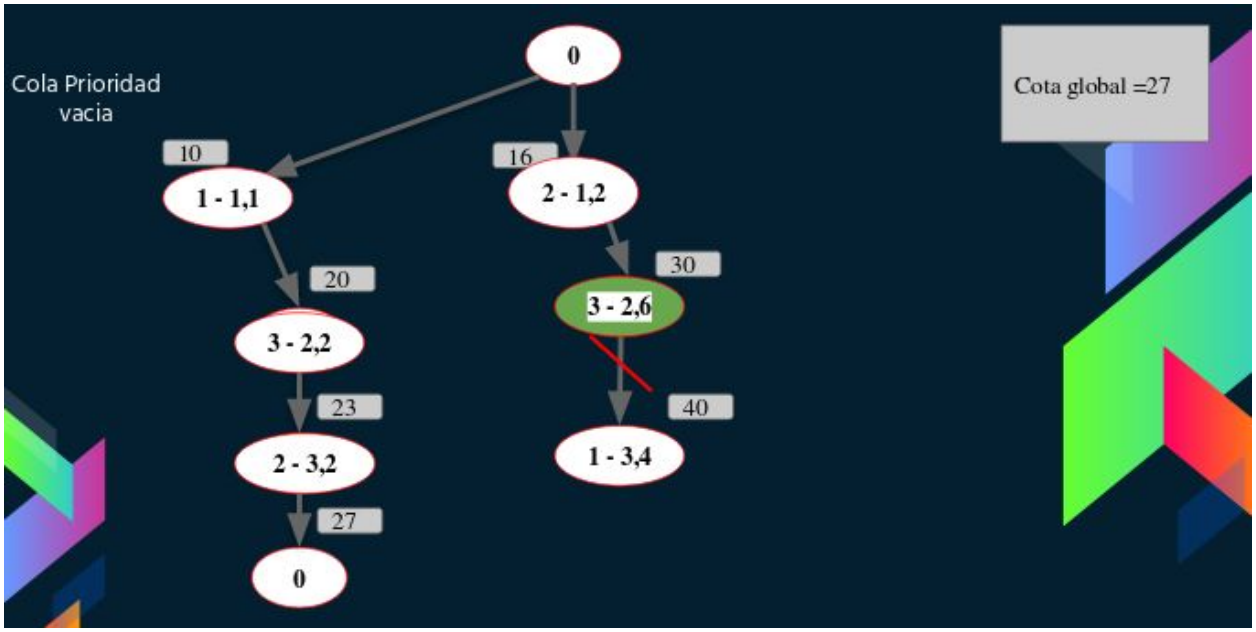

Cola Prioridad
3,2 - 23
2,4 - 24
2,6 - 30
Cota global =30
0
10
1 - 1,1
16
2 - 1,2
13
3 - 1,3
20
3 - 2,2
30
3 - 2,6
14
1 - 2,3
24
2 - 2,4
23
2 - 3,2
15
2 - 3,1
30
0

Aquí se ha alcanzado otro nodo hoja, cuyo coste mejora al actual, actualizando la solución y la cota global.

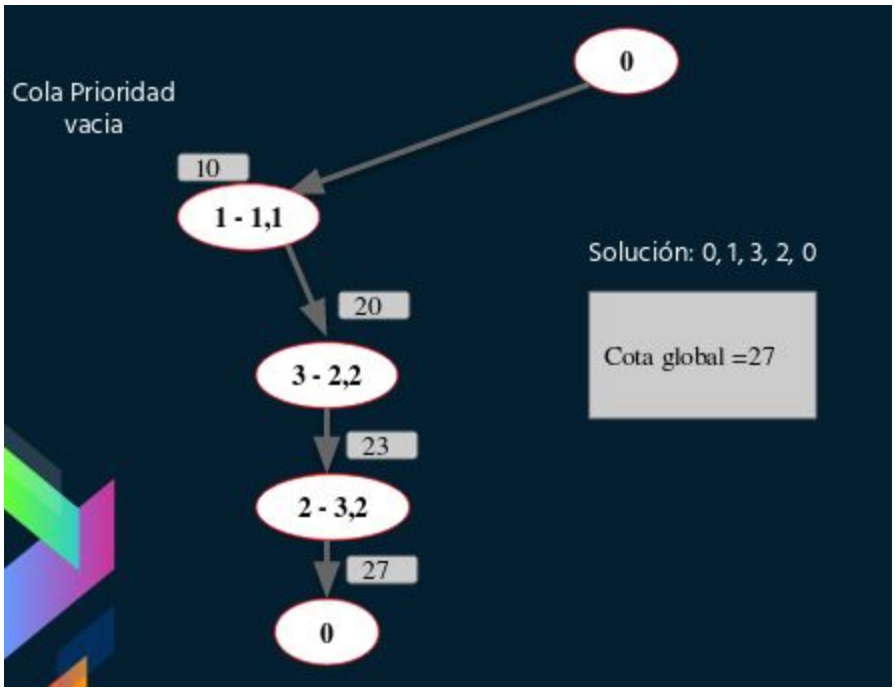




Finalmente la cola con prioridad se vacía y se deja de comprobar nodos

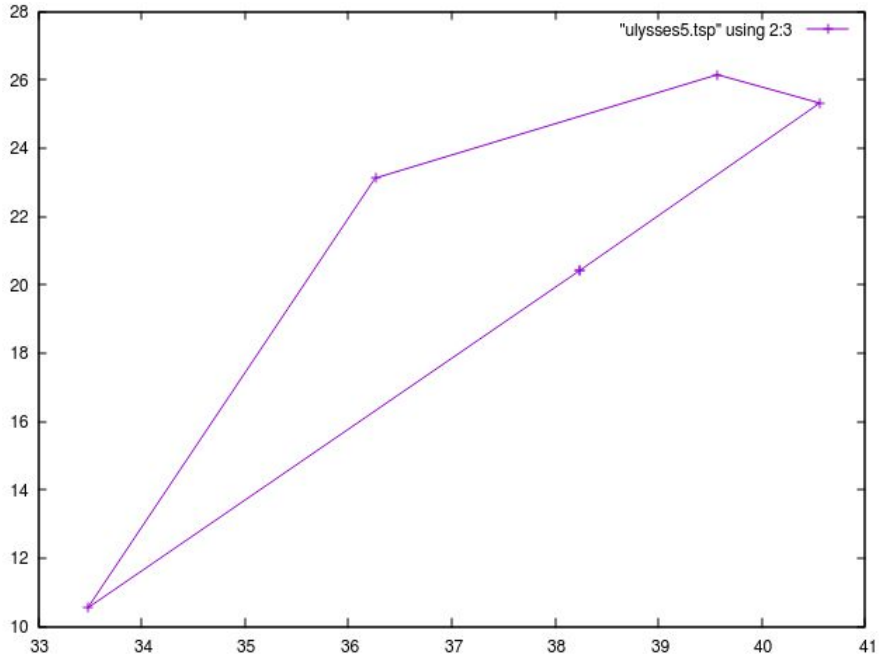


Cuya solucion es:

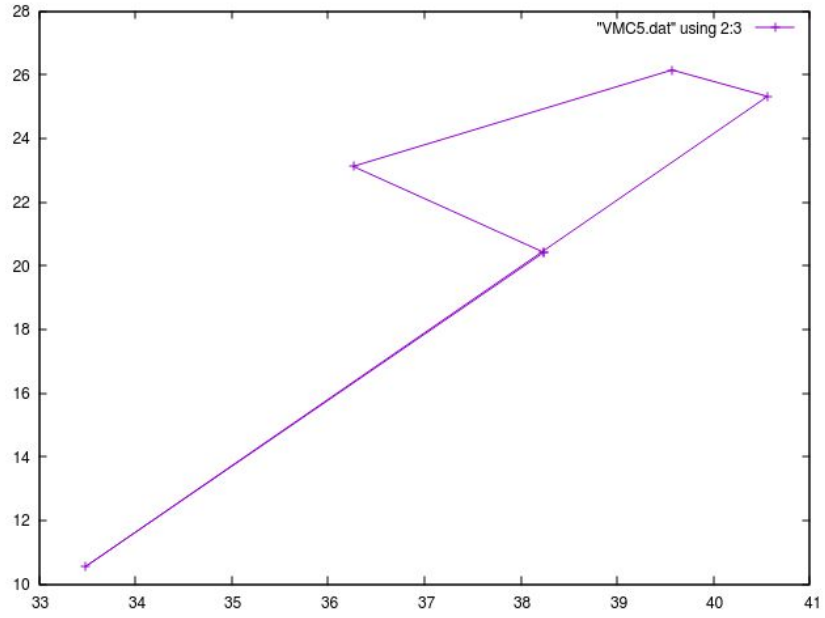


6. Gráficas

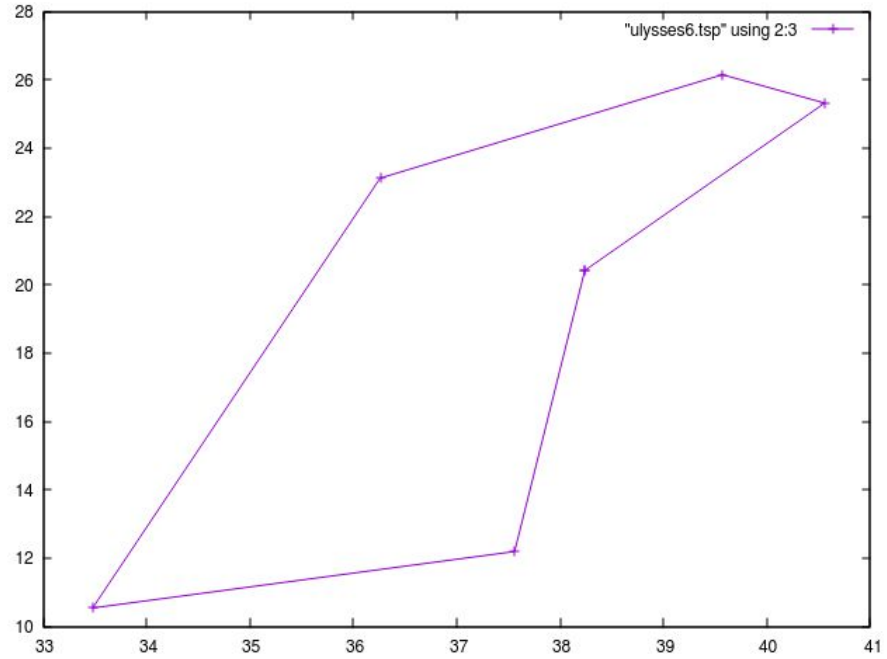
B&B 5 ciudades



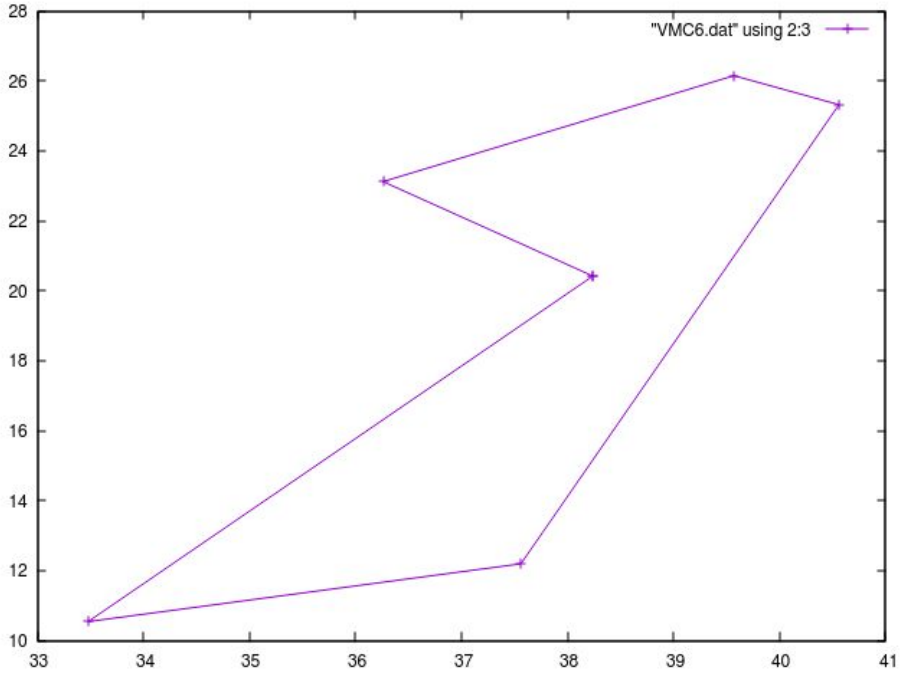
VMC 5 ciudades



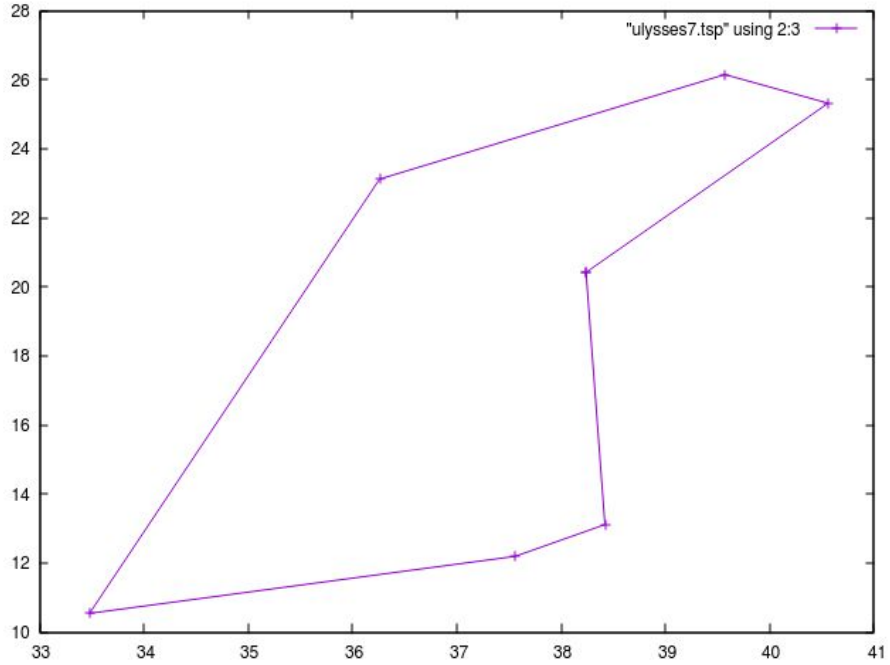
B&B 6 ciudades



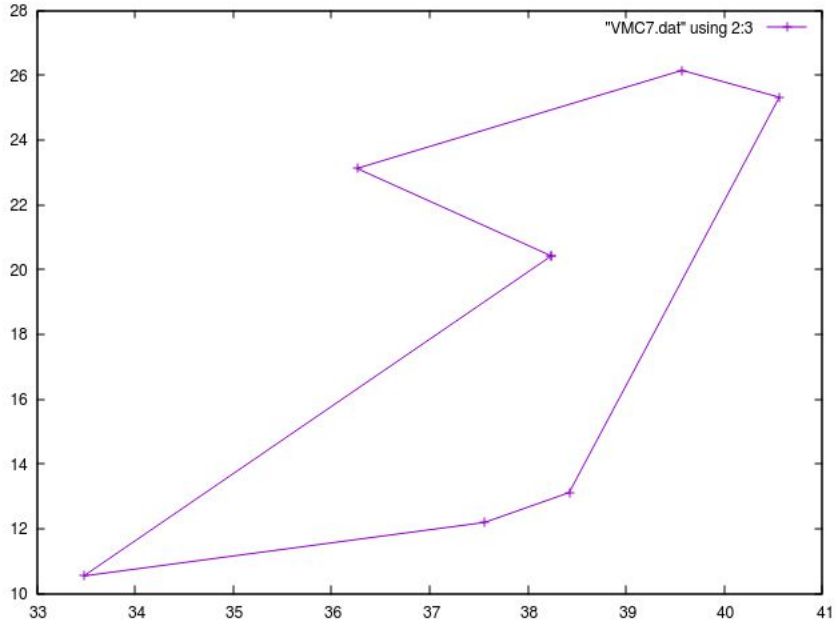
VMC 6 ciudades



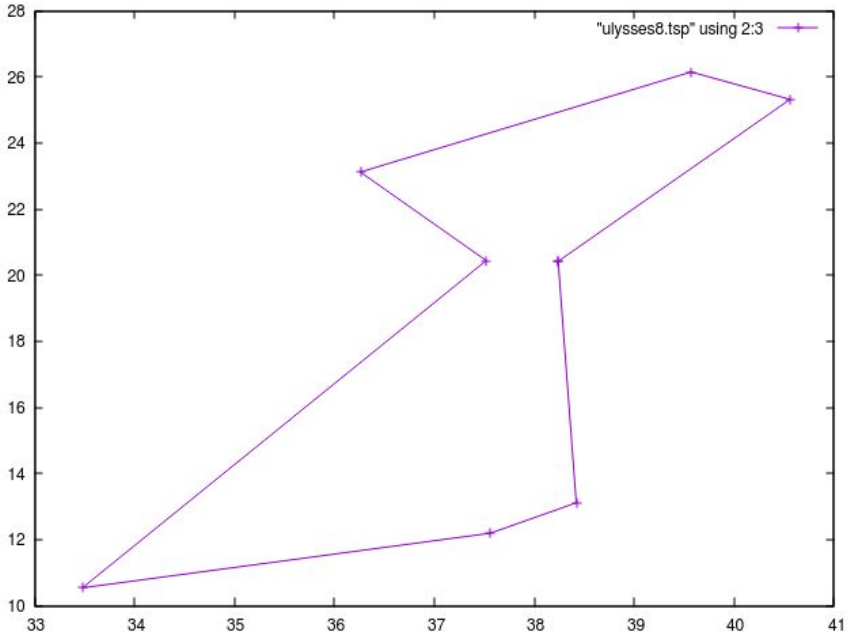
B&B 7 ciudades



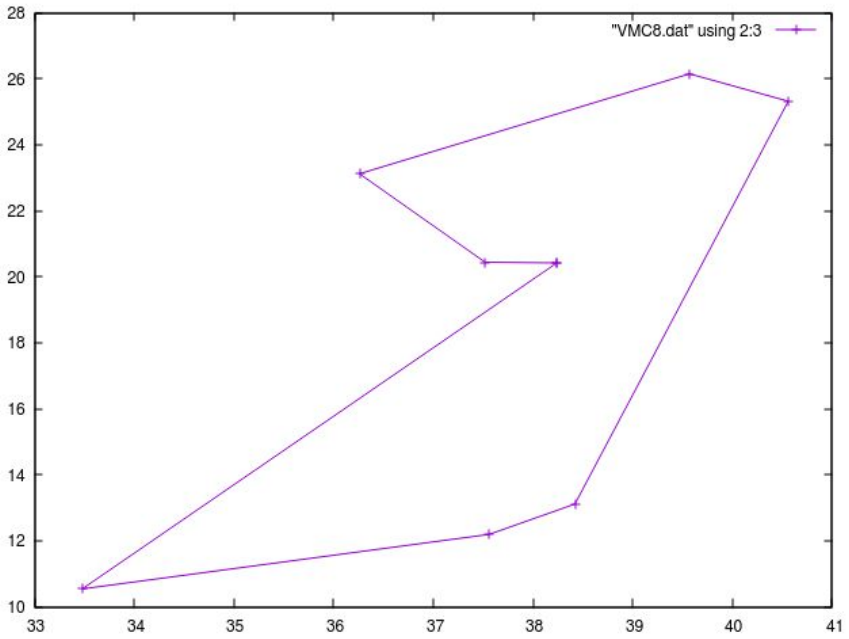
VMC 7 ciudades



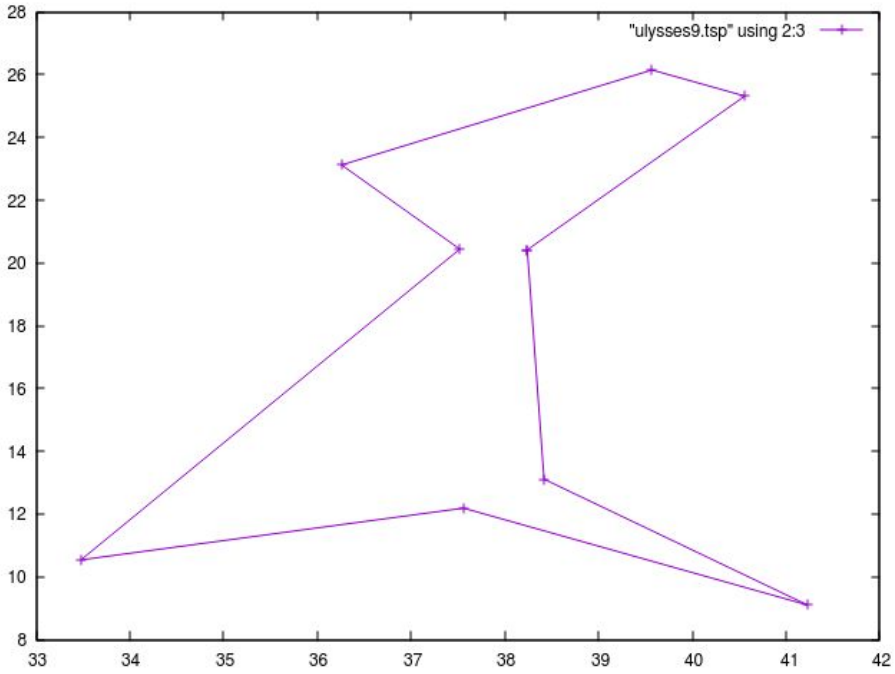
B&B 8 ciudades



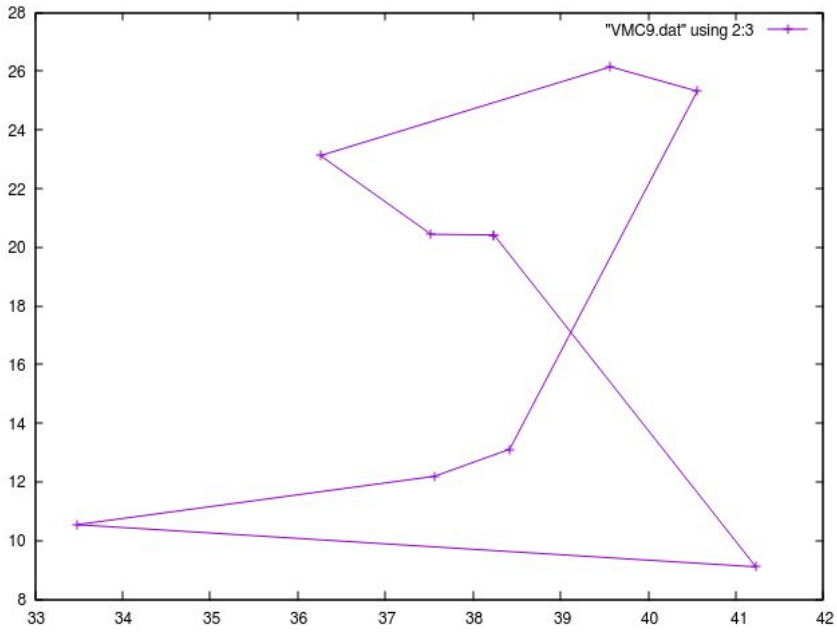
VMC 8 ciudades



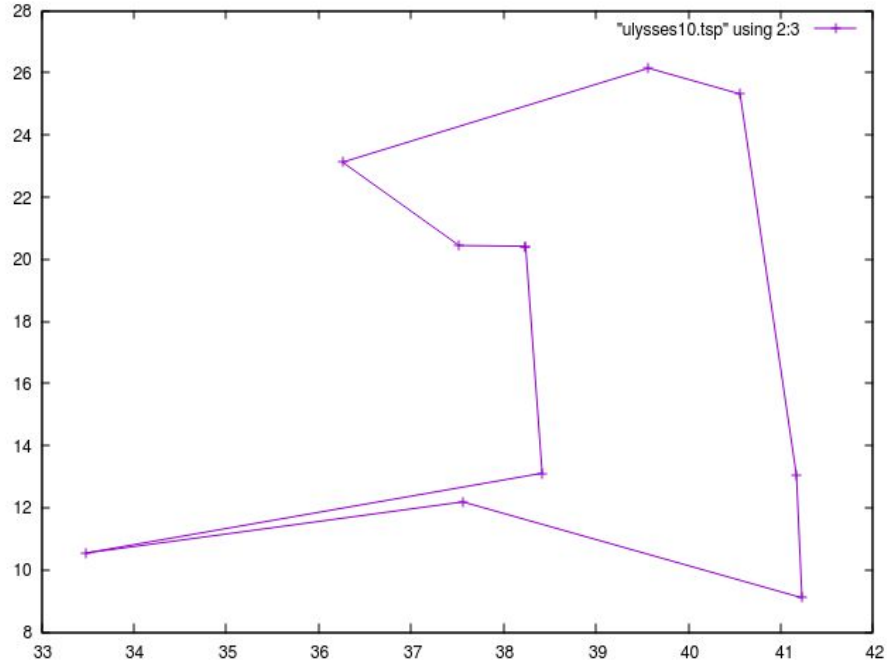
B&B 9 ciudades



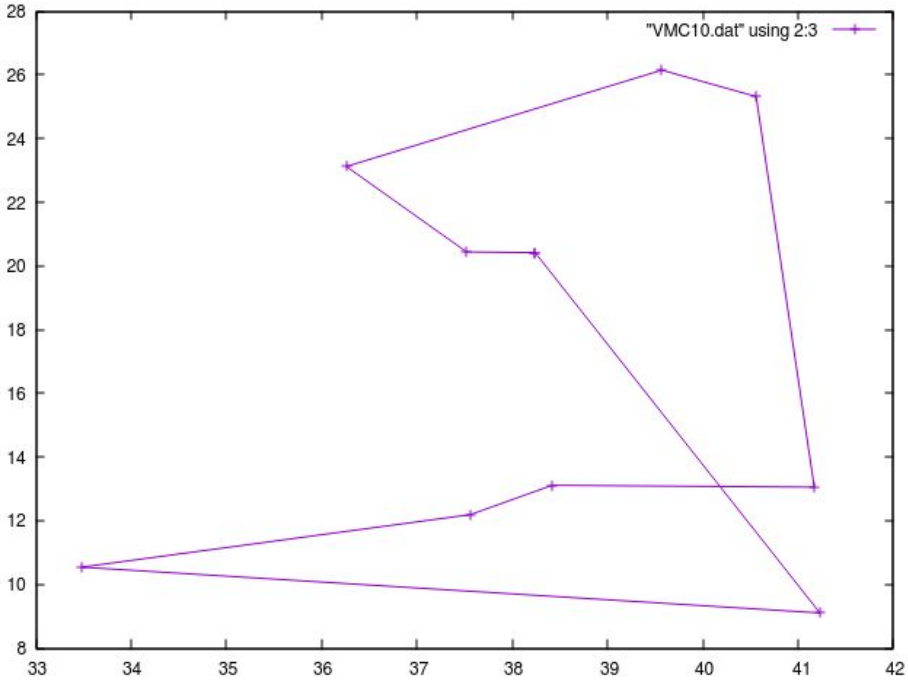
VMC 9 ciudades



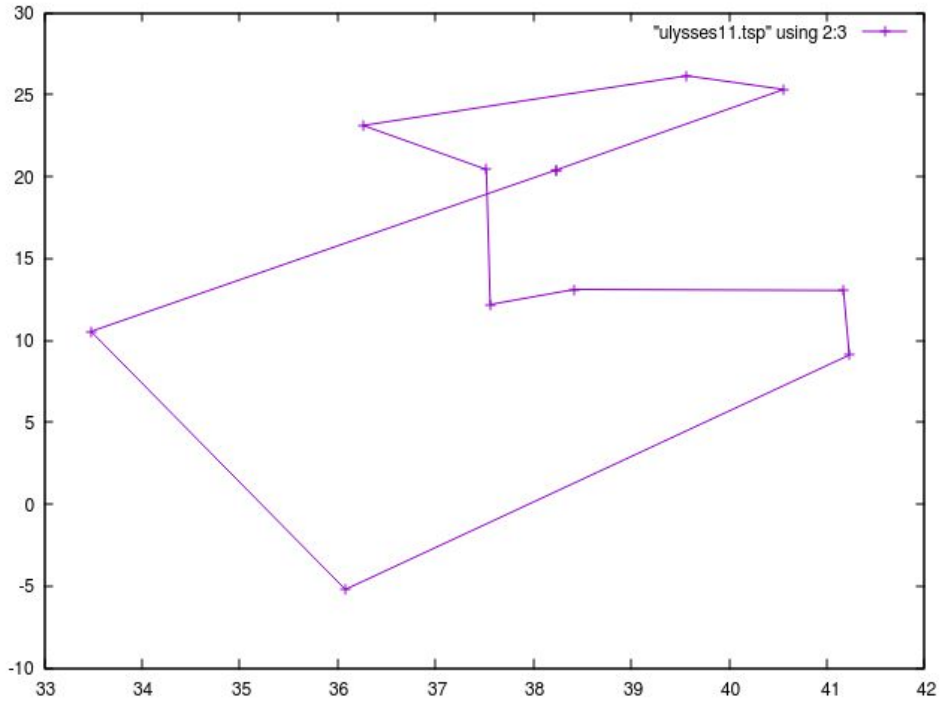
B&B 10 ciudades



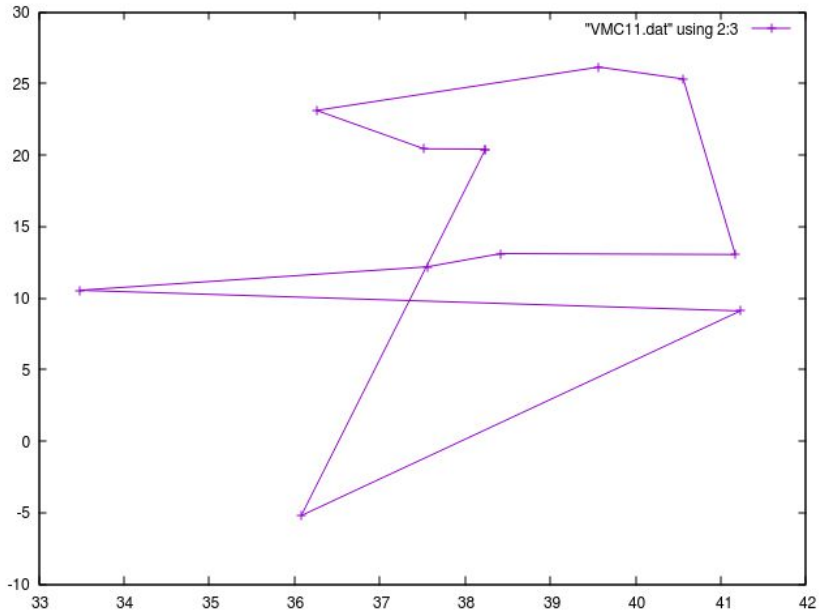
VMC 10 ciudades



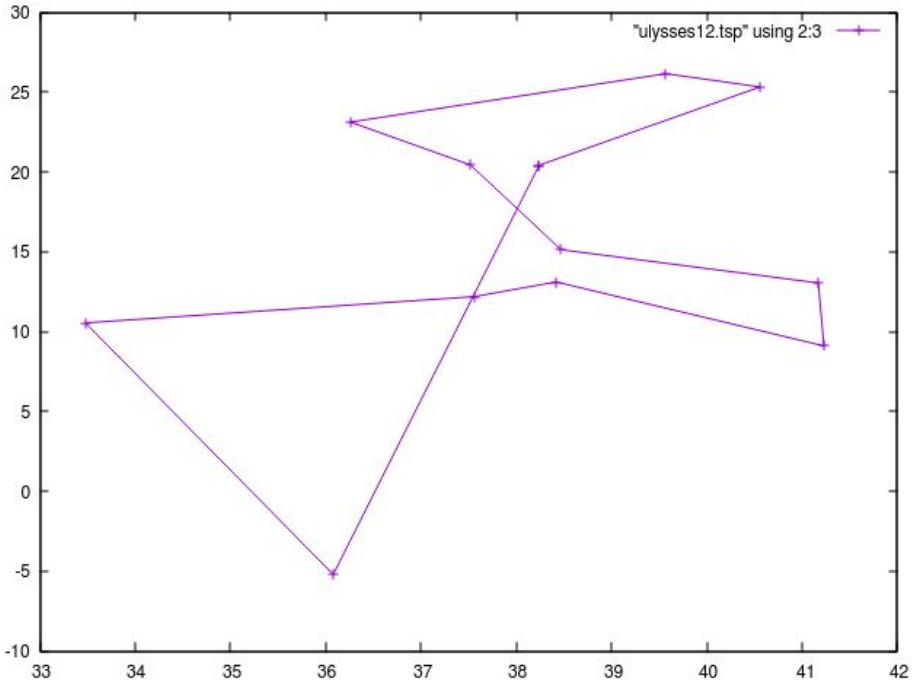
B&B 11 ciudades



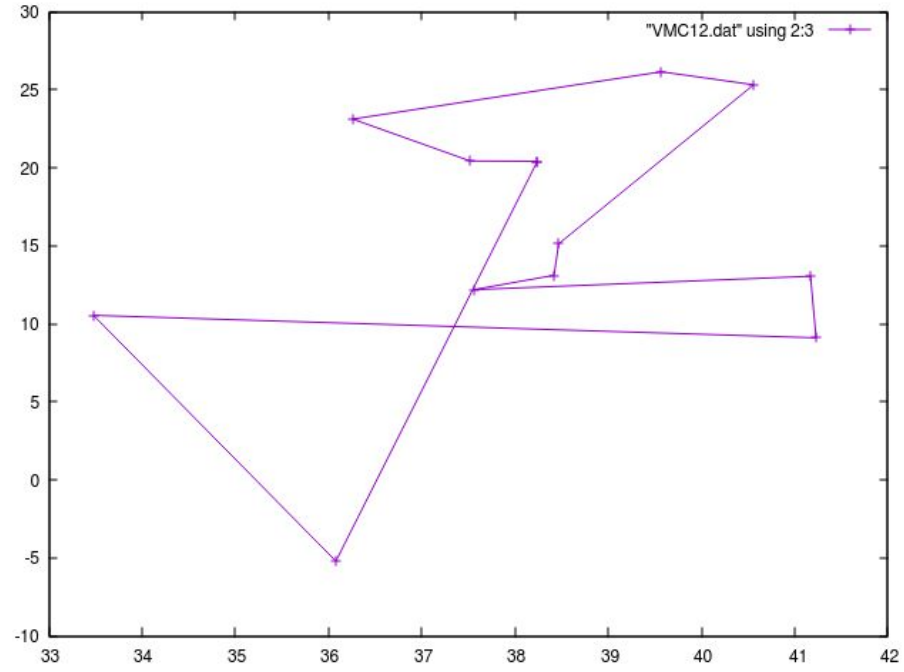
VMC 11 ciudades



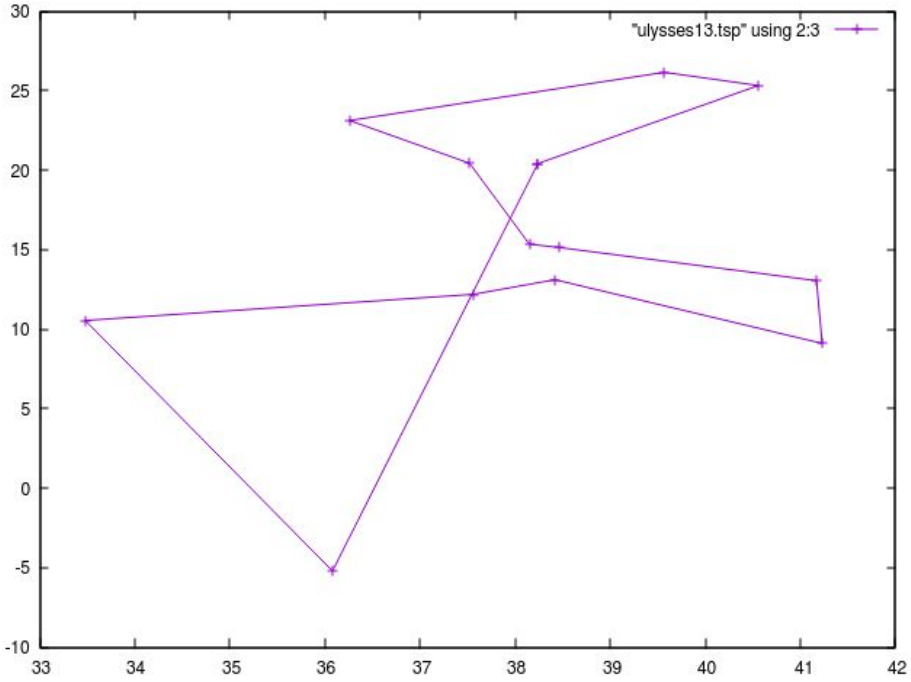
B&B 12 ciudades



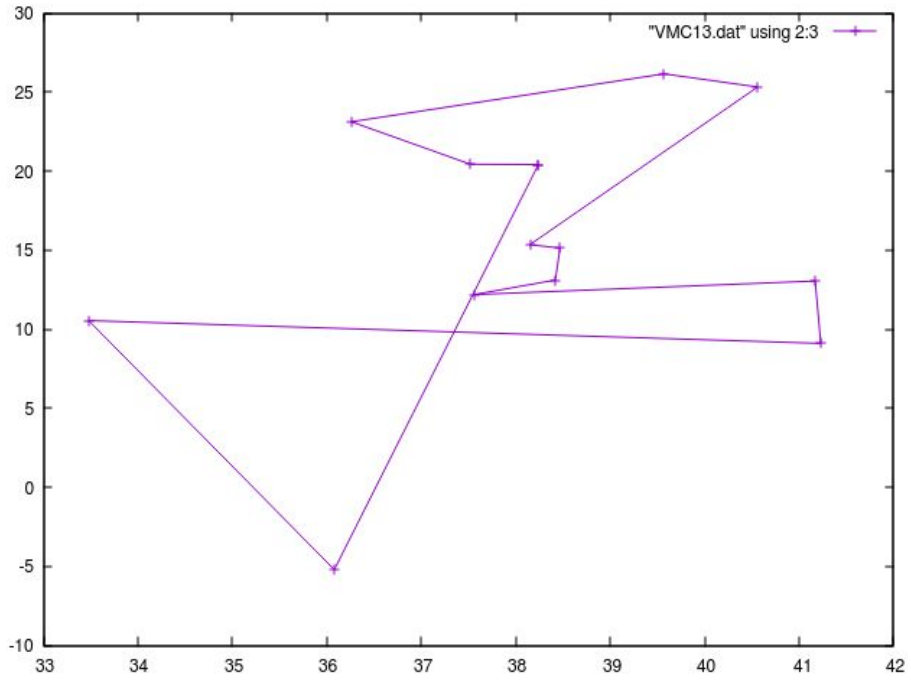
VMC 12 ciudades



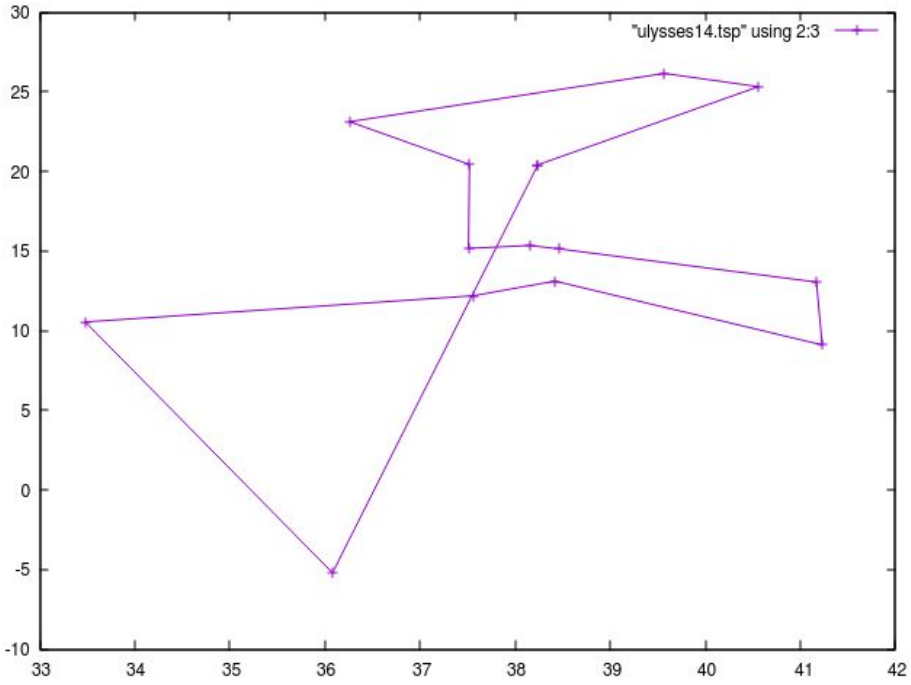
B&B 13 ciudades



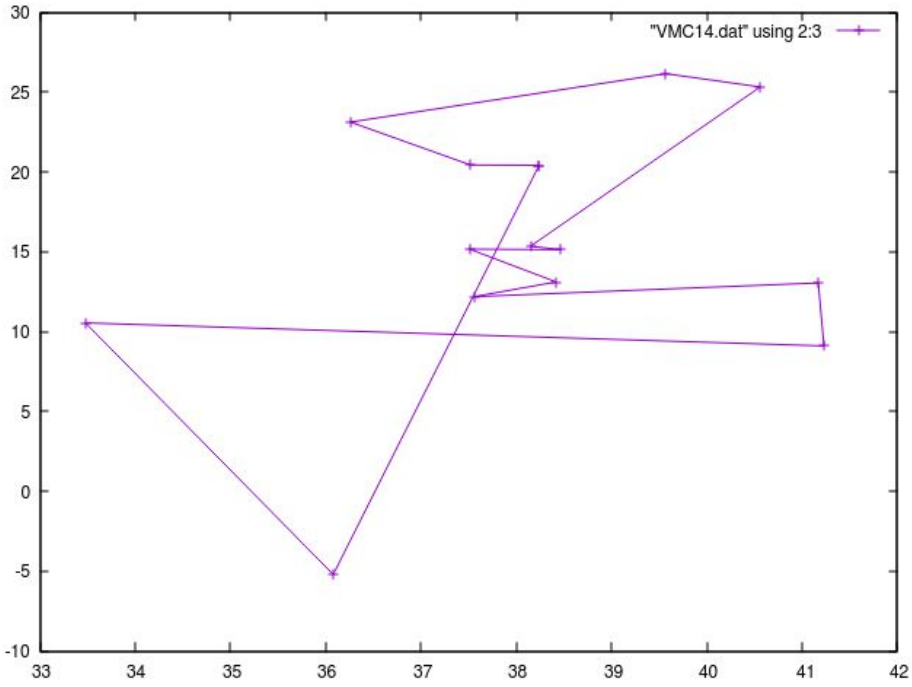
VMC 13 ciudades



B&B 14 ciudades



VMC 14 ciudades



7. Resultados de ejecución

B&B 5 ciudades

```
Ruta: 1 - 5 - 4 - 2 - 3 -
Costo solucion: 35.0512
Nodos expandidos: 28
Numero de podas: 4
Tamaño max cola: 14
Tiempo de ejecucion: 0.000304852

1 38.24 20.42
5 33.48 10.54
4 36.26 23.12
2 39.57 26.15
3 40.56 25.32
1 38.24 20.42
```

B&B 6 ciudades

```
Ruta: 1 - 6 - 5 - 4 - 2 - 3 -
Costo solucion: 36.7434
Nodos expandidos: 93
Numero de podas: 37
Tamaño max cola: 54
Tiempo de ejecucion: 0.00141639

1 38.24 20.42
6 37.56 12.19
5 33.48 10.54
4 36.26 23.12
2 39.57 26.15
3 40.56 25.32
1 38.24 20.42
```

B&B 7 ciudades

```
Ruta: 1 - 7 - 6 - 5 - 4 - 2 - 3 -
Costo solucion: 37.0569
Nodos expandidos: 268
Numero de podas: 182
Tamaño max cola: 160
Tiempo de ejecucion: 0.00579718

1 38.24 20.42
7 38.42 13.11
6 37.56 12.19
5 33.48 10.54
4 36.26 23.12
2 39.57 26.15
3 40.56 25.32
1 38.24 20.42
```

B&B 8 ciudades

```
Ruta: 1 - 3 - 2 - 4 - 8 - 5 - 6 - 7 -
Costo solucion: 37.8274
Nodos expandidos: 823
Numero de podas: 663
Tamaño max cola: 775
Tiempo de ejecucion: 0.0219184

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
5 33.48 10.54
6 37.56 12.19
7 38.42 13.11
1 38.24 20.42
```

B&B 9 ciudades

```
Ruta: 1 - 3 - 2 - 4 - 8 - 5 - 6 - 9 - 7 -
Costo solucion: 46.2622
Nodos expandidos: 4979
Numero de podas: 6123
Tamaño max cola: 4498
Tiempo de ejecucion: 0.145268

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
5 33.48 10.54
6 37.56 12.19
9 41.23 9.1
7 38.42 13.11
1 38.24 20.42
```

B&B 10 ciudades

```
Ruta: 1 - 8 - 4 - 2 - 3 - 10 - 9 - 6 - 5 - 7 -
Costo solucion: 47.776
Nodos expandidos: 18499
Numero de podas: 29250
Tamaño max cola: 17636
Tiempo de ejecucion: 0.648598

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
10 41.17 13.05
9 41.23 9.1
6 37.56 12.19
5 33.48 10.54
7 38.42 13.11
1 38.24 20.42
```

B&B 11 ciudades

```
Ruta: 1 - 3 - 2 - 4 - 8 - 6 - 7 - 10 - 9 - 11 - 5 -
Costo solucion: 72.5113
Nodos expandidos: 275784
Numero de podas: 448879
Tamaño max cola: 139557
Tiempo de ejecucion: 10.5289

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
6 37.56 12.19
7 38.42 13.11
10 41.17 13.05
9 41.23 9.1
11 36.08 -5.21
5 33.48 10.54
1 38.24 20.42
```

B&B 12 ciudades

```
Ruta: 1 - 3 - 2 - 4 - 8 - 12 - 10 - 9 - 7 - 6 - 5 - 11 -
Costo solucion: 79.1562
Nodos expandidos: 587297
Numero de podas: 1436024
Tamaño max cola: 260981
Tiempo de ejecucion: 26.9752

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
12 38.47 15.13
10 41.17 13.05
9 41.23 9.1
7 38.42 13.11
6 37.56 12.19
5 33.48 10.54
11 36.08 -5.21
1 38.24 20.42
```

B&B 13 ciudades

```
Ruta: 1 - 3 - 2 - 4 - 8 - 13 - 12 - 10 - 9 - 7 - 6 - 5 - 11 -
Costo solucion: 79.2791
Nodos expandidos: 4067706
Numero de podas: 10822217
Tamaño max cola: 1860460
Tiempo de ejecucion: 222.266

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
13 38.15 15.35
12 38.47 15.13
10 41.17 13.05
9 41.23 9.1
7 38.42 13.11
6 37.56 12.19
5 33.48 10.54
11 36.08 -5.21
1 38.24 20.42
```

B&B 14 ciudades

```
Ruta: 1 - 3 - 2 - 4 - 8 - 14 - 13 - 12 - 10 - 9 - 7 - 6 - 5 - 11 -
Costo solucion: 80.0851
Nodos expandidos: 29591650
Numero de podas: 83444533
Tamaño max cola: 13302820
Tiempo de ejecucion: 1874.46

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
14 37.51 15.17
13 38.15 15.35
12 38.47 15.13
10 41.17 13.05
9 41.23 9.1
7 38.42 13.11
6 37.56 12.19
5 33.48 10.54
11 36.08 -5.21
1 38.24 20.42
```