Tarea 1

Universidad de Guanajuato
Eduardo Calvo Martínez

Optimización estocástica
7 de febrero de 2025

Planteamiento del problema

El problema que voy a tratar durante la asignatura es el traveling salesman problem (TSP), el problema se define como sigue. Dado un conjunto de n ciudades y las distancias entre cada par de ellas, se busca determinar el recorrido de menor longitud que permita visitar las n ciudades exactamente una vez y regresar a la ciudad de origen.

En la versión simétrica que es la que trabajaré durante el curso, la distancia de dos ciudades i y j es la misma en ambos sentidos. Es decir

$$d_{ij} = d_{ji}, \quad \forall i \neq j.$$

En otras palabras esto nos dice que el grafo que representa el problema **no es dirigido**. En este caso nuestras instancias nos dan coordenadas de los datos y la distancia entre nodos es el techo de la distancia euclidiana entre los puntos. En este caso las instancias que utilizaremos son "pla7397", "pla33810", "pla85900", de la TSP lib y los mejores valores conocidos para cada una son 23260728, 66048945 y 142382641 respectivamente.

Generador de Soluciones aleatorias

Notemos que la soluciones de nuestro problema van a estar dadas por vectores de permutaciones, es decir que el vector representa el orden en que se van a visitar las ciudades. Entonces para generar soluciones aleatorias lo que hacemos es tomar un vector con todos los números de 0 hasta $n-1$ (pues solo tenemos que visitar cada ciudad exactamente una vez) y luego les damos un orden aleatorio

```
vector<int> randSolution(unordered_map<int, pair<float,float>> &nodeToCoord
    , long long int &cost){
     vector<int> posCities;
     cost = 0;

     for(int i = 1; i <= nodeToCoord.size(); i++) posCities.push_back(i);
     randomShuffle(posCities, 91);

     for(int i = 0; i < posCities.size(); i++) {
          int node1 = posCities[i], node2 = posCities[(i+1)%posCities.size()
             ];
          pair<float,float> x = nodeToCoord[node1], y = nodeToCoord[node2];
          cost += ceil_2D(x, y);
     }
     return posCities;
}
```

Listing 1: Función generadora de soluciones aleatorias

El código anterior lo que hace llenar un vector vacío como se dijo antes y mediante la función `randomShuffle` usando `seed` como semilla los ordena aleatoriamente. Luego calcula la distancia entre un nodo y su siguiente

para sumarlo en la función de costo, así hasta que llega al último nodo del vector donde se toma la distancia de este con el primer nodo. Las funciones `2Dceil` y `randomShuffle` solo calculan el techo de la distancia euclidiana entre los dos puntos y hacen el orden aleatorio como acontinuación

```
void randomShuffle(vector<int> &v, const int &seed){
    mt19937 gen(seed);
    shuffle(v.begin(), v.end(), gen);
    return;
}

int ceil_2D(const pair<float,float> &x, const pair<float,float> &y){
    float first = x.first - y.first, second = x.second - y.second;

    return int(ceil(sqrt(pow(first, 2) + pow(second, 2))));
}
```

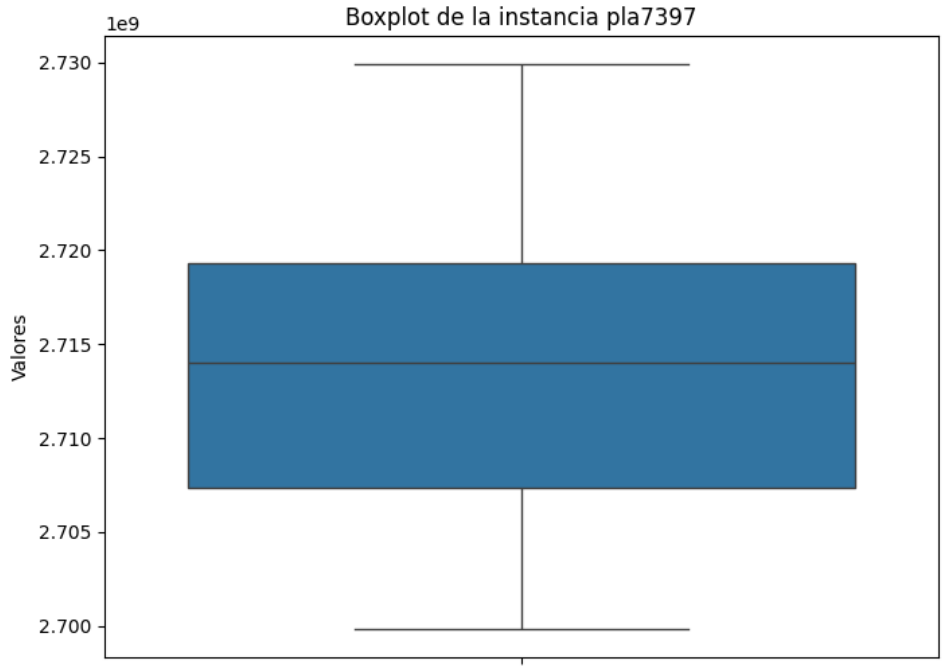
Dado que nuestra función de costo necesita hacer el recorrido para calcular el costo de una olución, entonces podemos decir que tiene complejidad $O(n)$. La generación de la solución en este caso es $O(\texttt{shuffle}+n)$ pues se combina el gasto de la función shuffle con la generación del vector.

Para correr en el cluster se paralelizo el código y se usaron 32 hilos para ejecutar la busqueda de las mejores soluciones aleatorias generadas

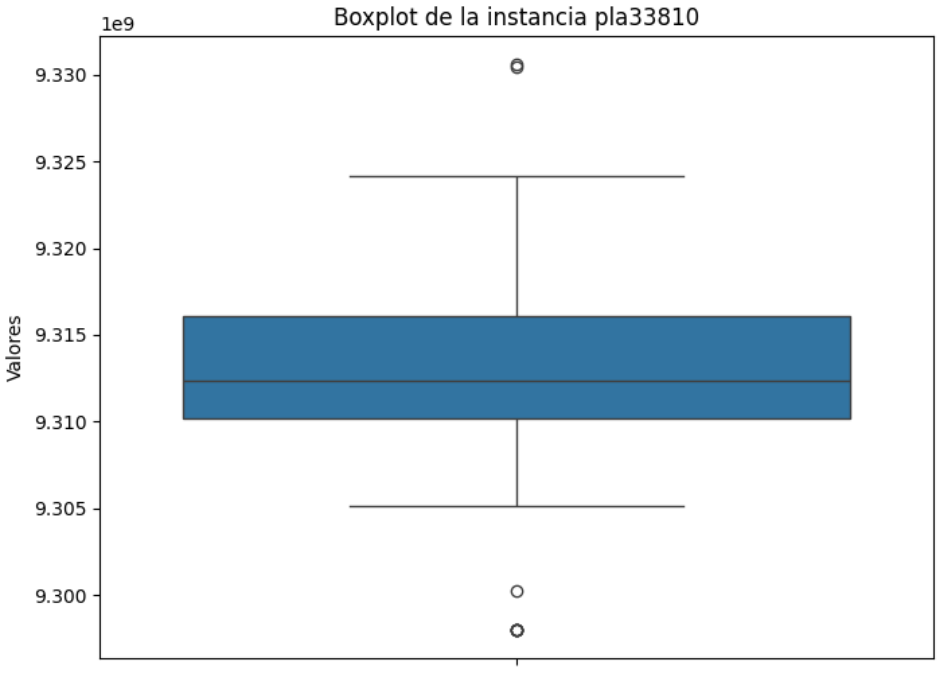
```
omp_set_num_threads(32);
#pragma omp parallel for firstprivate(ref, smallCost) private(i)
        for(i = 0; i < 100; i++){     //Buscamos 100 veces la mejor de
           100000 soluciones
          int tid = omp_get_thread_num();
          printf("Este␣es␣el␣hilo␣%d\n", tid);
          vector<int> bestSol;
            for (j = 0; j < ref; j++){
                long long int pivCost;
                vector<int> sol = randSolution(nodeToCoord, pivCost);

                if (pivCost < smallCost){
                        smallCost = pivCost;
                }
            }
            if(archivo != NULL) fprintf(archivo,"%lld\n", smallCost);
            bestCostes.push_back(smallCost);
        }
```

Los boxplot de cada .txt con soluciones para cada instancia son los siguientes

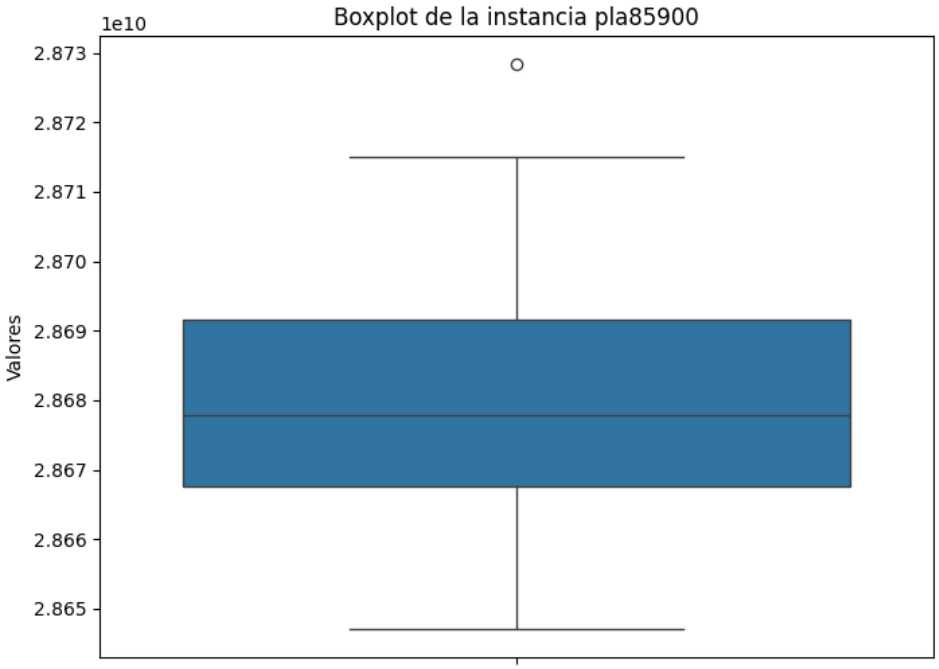


(a) Boxplot de la instancia pla7397



(b) Boxplot de la instancia pla33810

Figura 1: BoxPlots de las instancias.



(a) Boxplot de la instancia pla85900

Figura 2: BoxPlots de las instancias pt2.

Los papers que se usarán como referencia son "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic" de Keld Helsgaun y "Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problem" de David Applegate, Robert Bixby, Vasek Chvátal y William Cook. Además en la página de la TSPLib se especifican los mejores valores para cada instancia de la librería.