# Tarea 2

Universidad de Guanajuato
Eduardo Calvo Martínez

Optimización estocástica
14 de febrero de 2025

## Heurística constructiva

En esta tarea se buscaba comparar la eficiencia de aplicar una búsqueda incremental usando una solución inicial generada aleatoriamente y una solución generada por alguna heúristica constructiva. Para este caso nosotros usamos la heurística para el TSP llamada "Farthest Insertion", el procedimiento es iniciar en un nodo arbitario y luego insertar nuevos nodos en nuestra solución parcial hasta conseguir una solución completa del problema. Para elegir que punto es el que vamos a insertar, buscamos el punto tal que

$$\max_{v \in V \setminus T} \min_{i=1}^{|T|} (d(v, v_i) + d(v_{i+1}, v) - d(v_i, v_{i+1})),$$

Para $V$ el conjunto de nodos y $T$ nuestra solución parcial. La minimización de la suma de distancias nos asegura que insertemos al nuevo nodo en la posición donde se afecta menos al costo, mientras que la maximización nos hace encargarnos primero de los peores nodos. En pseudocódigo es el siguiente

**Algorithm 1** Farthest Insertion

```
procedure FARTHEST_INSERTION(distances)
    Elegimos (aleatoriamente) un nodo incial v_1 y lo agregamos a la solución
    Elegimos el segundo nodo como el más lejano al inicial y lo agregamos
    while V \ T ≠ ∅ do
        Elegimos el nodo v como el más lejano a T
        Buscamos la posición i donde se minimiza (d(v, v_i) + d(v_{i+1}, v) − d(v_i, v_{i+1}))
        Insertamos el nodo v en la posición i de nuestra solución
    end while
end procedure
```

Con esta heurística es que generamos la solución con la que inicia la evaluación incremental.

## Evaluación incremental

La evaluación incremental se efectuó mediante el algoritmo **sthocastic hill climbing** el cual vimos en clase, para esto debemos de revisar los vecinos (distancia *2opt*) de nuestra solución actual y tomar el primero que nos muestre una mejoría en nuestra función objetivo. Dado que calcular todos los vecinos es demasiado caro en memoria, entonces usamos descriptores, en nuestro problema un vecino es aquella solución que tiene dos aristas distintas y para generar estos vecinos en nuestra representación basta con invertir un segmento del vector que representa la solución.

Esto es, si tenemos la solución dada por el vector `[1,5,2,6,7,3,9,4,8]` entonces si invertimos el segmento que va de `4` a `5` tenemos el vector `[1,4,9,3,7,6,2,5,8]` el cual representa casi la misma solución que antes, con la diferencia que la arista $a_{1,5}$ se cambió por la arista $a_{1,4}$ y la arista $a_{4,8}$ se cambió por la arista $a_{5,8}$. Esto nos dice que basta con guardar dos enteros `i` y `j` los cuales nos indican el segmento del arreglo que hay que invertir, además para calcular el coste de nuestro vecino basta con restar el valor de las dos aristas eliminadas y sumar el costo de las nuevas al costo de nuestra solución.

En nuestro código en lugar de generar toda la lista de descriptores lo que hicimos es generar un descriptor aleatorio mientras no hayamos encontrado una mejora en el coste. Si el descriptor no representaba una mejora lo guardamos en un `set` para registrar que ya hemos evaluado ese descriptor. Al momento de generar nuevas soluciones tenemos que verificar en nuestro `set` que no se hubiera evaluado ese descriptor, en cuanto encontremos un vecino que mejora, generamos el vecino y lo cambiamos por la solución actual e iniciamos de nuevo, ahora sobre el vecino. El algoritmo termina cuando no hay algún vecino que mejore.

## Tablas compartivas

A continuación tenemos estadísticos de los tiempos (segundos) de procesamiento de las 50 evaluaciones incrementales que se ejecutaron para cada instancia. Hay una tabla para las evaluaciones cuya solución inicial está dada por la heurística y dada de manera aleatoria

| **Estadístico** | **Instancia d493** | **Instancia d657** | **Instancia dsj1000** |
|---|---|---|---|
| **Heurística** | | | |
| Mínimo | 6.9021 | 15.22247 | 43.09763 |
| Máximo | 19.62105 | 32.781 | 92.77232 |
| Promedio | 12.1276828 | 21.7140396 | 63.04339 |
| **Aleatoria** | | | |
| Mínimo | 9.5845 | 16.85526 | 56.21322 |
| Máximo | 20.90617 | 36.28331 | 93.7943 |
| Promedio | 13.4757182 | 25.156231 | 68.90129 |

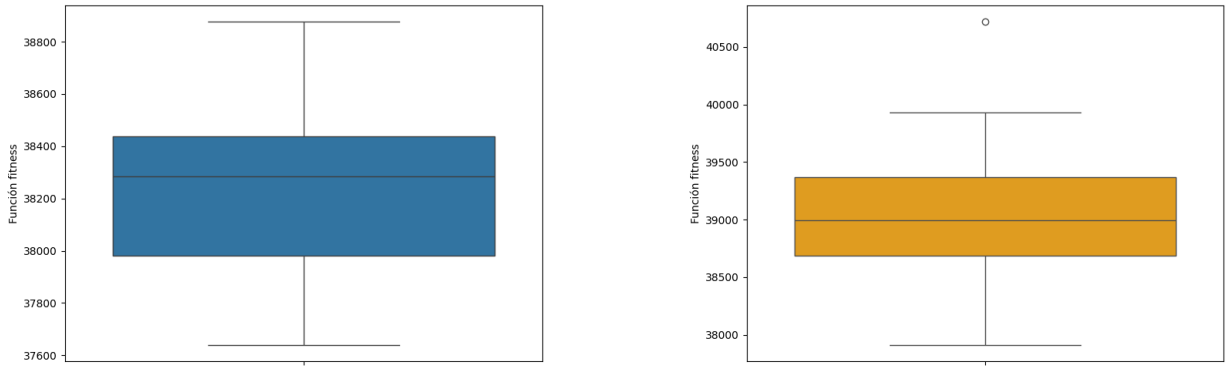
Cuadro 1: Comparación de tiempos

En las siguientes tablas tenemos estadísticos ahora del fitness dados por las dos maneras de generar instancias iniciales y la Tarea 1

| **Estadístico** | **Instancia d493** | **Instancia d657** | **Instancia dsj1000** |
|---|---|---|---|
| **Tarea 1** | | | |
| Mínimo | 403036 | 780010 | 517621000 |
| **Heurística** | | | |
| Mínimo | 37639 | 53371 | 20693425 |
| Máximo | 38877 | 54803 | 21292312 |
| Promedio | 38232.02 | 54154.1 | 21001319.28 |
| **Aleatoria** | | | |
| Mínimo | 37910 | 52909 | 20737636 |
| Máximo | 40720 | 56160 | 601150955 |
| Promedio | 39019.8 | 54874.92 | 21357934.48 |

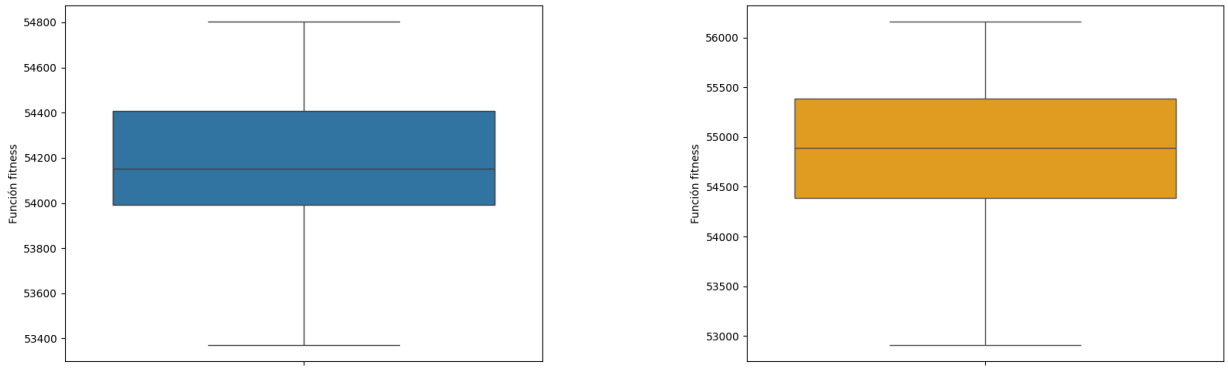
Cuadro 2: Comparación de fitness

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



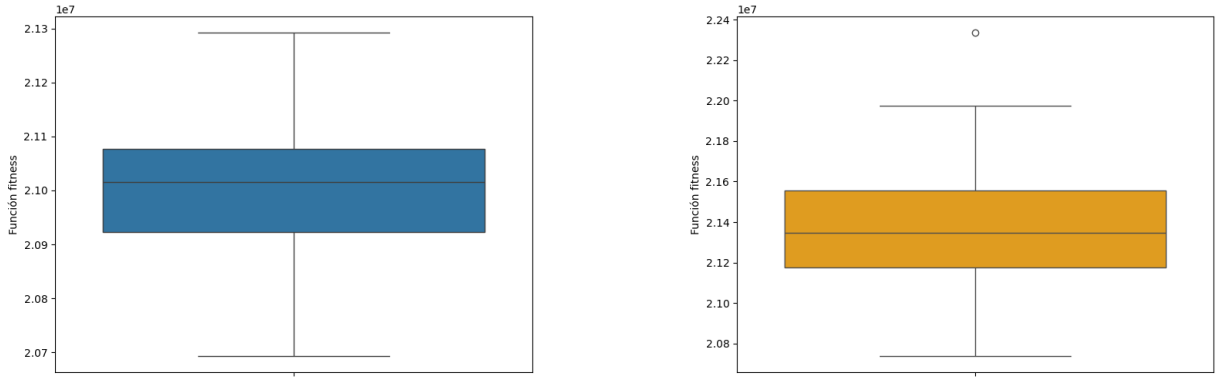

(a) Heurística (b) Aleatoria

Figura 1: BoxPlots de la instancia d493.




(a) Heurística (b) Aleatoria

Figura 2: BoxPlots de la instancia d657.




(a) Heurística (b) Aleatoria

Figura 3: BoxPlots de la instancia dsj1000.

## Conclusiones

Como podemos ver, el uso de la heurística se ve reflejado en costos mucho menores, estamos hablando de que los costos con la heurística son en promedio más pequeños que con la generación aleatoria además de que el tiempo de procesamiento es menor con la heurística que con la solución aleatoria. Por lo que por ahora usar la heurística parece darnos una mejora en la búsqueda sobre el espacio de soluciones.