# VERSIÓN 3 TSP: INVENTADA

Los puntos tratados en esta parte del guion son los siguientes:

- Descripción del problema
- Descripción de la solución
- Análisis de la eficiencia teórica, empírica e híbrida
- Comparación con la solución óptima de los archivos tour

## Descripción del problema

En esta práctica se nos pide implementar varias soluciones para el problema del viajante de comercio, en este caso esta es la tercera versión, la versión inventada por los alumnos.

## Descripción de la solución

La solución desarrollada se inspira en el algoritmo A* (estrella) fuertemente usada en IA para búsqueda de caminos evitando obstáculos. Como algoritmo que Greedy que es este, también evalúa en cada momento la mejor opción de las disponibles, al igual que el algoritmo tsp por cercanía del cual también se inspira.

Una vez aclarado lo anterior, expliquemos el algoritmo con detalle. Este algoritmo intenta recorrer todas las ciudades una sola vez y volver a la ciudad de origen. Recibe un conjunto de ciudades y su tamaño y devuelve el camino que se haya calculado. Para llevar a cabo este cálculo obtiene la primera ciudad de la colección, esta será la ciudad origen (también la llamaremos actual) y obtiene la ciudad que más lejos se encuentre de ella, a continuación, busca una ciudad cuya distancia a la ciudad actual sumada a la distancia desde esa ciudad hasta la ciudad más lejana sea mínima. Una vez encontrada esa ciudad se convierte en la actual, se saca de la colección de ciudades por recorrer y se pasa a la de ciudades recorridas (el resultado final). Se repite el algoritmo hasta que ya no queden más ciudades por recorrer.

A continuación de describe la implementación del algoritmo por encima para que se vea de forma más clara.

```
Ciudad * RecorreCiudadesA(Ciudad * ciudades, int & num_ciudades){
  Ciudad * res = new Ciudad[num_ciudades+1];
  Ciudad actual = res[num_ciudades] = res[0] = ciudades[0];

  EliminarCiudad(0, ciudades, num_ciudades);
  int pos_max, pos_min, size = num_ciudades;
  double dist, min;

  pos_max = BuscaMaxDistancia(actual, ciudades, size); // ciudad mas l

  for (int i = 1; i < num_ciudades; ++i){
    min = MAX_BUSQUEDA;
    //pos_max = BuscaMaxDistancia(actual, ciudades, size); // ciudad m

    for (int j = 0; j < size; ++j){
      dist = Distancia(actual, ciudades[j]);
      if ((dist+Distancia(ciudades[j], ciudades[pos_max])) < min
        && (ciudades[j] != ciudades[pos_max])){
        pos_min = j;
        min = dist;
      }
    }

    actual = res[i] = ciudades[pos_min];
    EliminarCiudad(pos_min, ciudades, size);
  }

  res[num_ciudades] = ciudades[pos_max];
  num_ciudades++;
  return res;
}
```

camino resultado

ciudad origen

sacar de ciudades no recorridas

buscar ciudad más lejana a origen

distancia de actual a otras

distancia de actual a otra y de otra a lejana debe ser minima

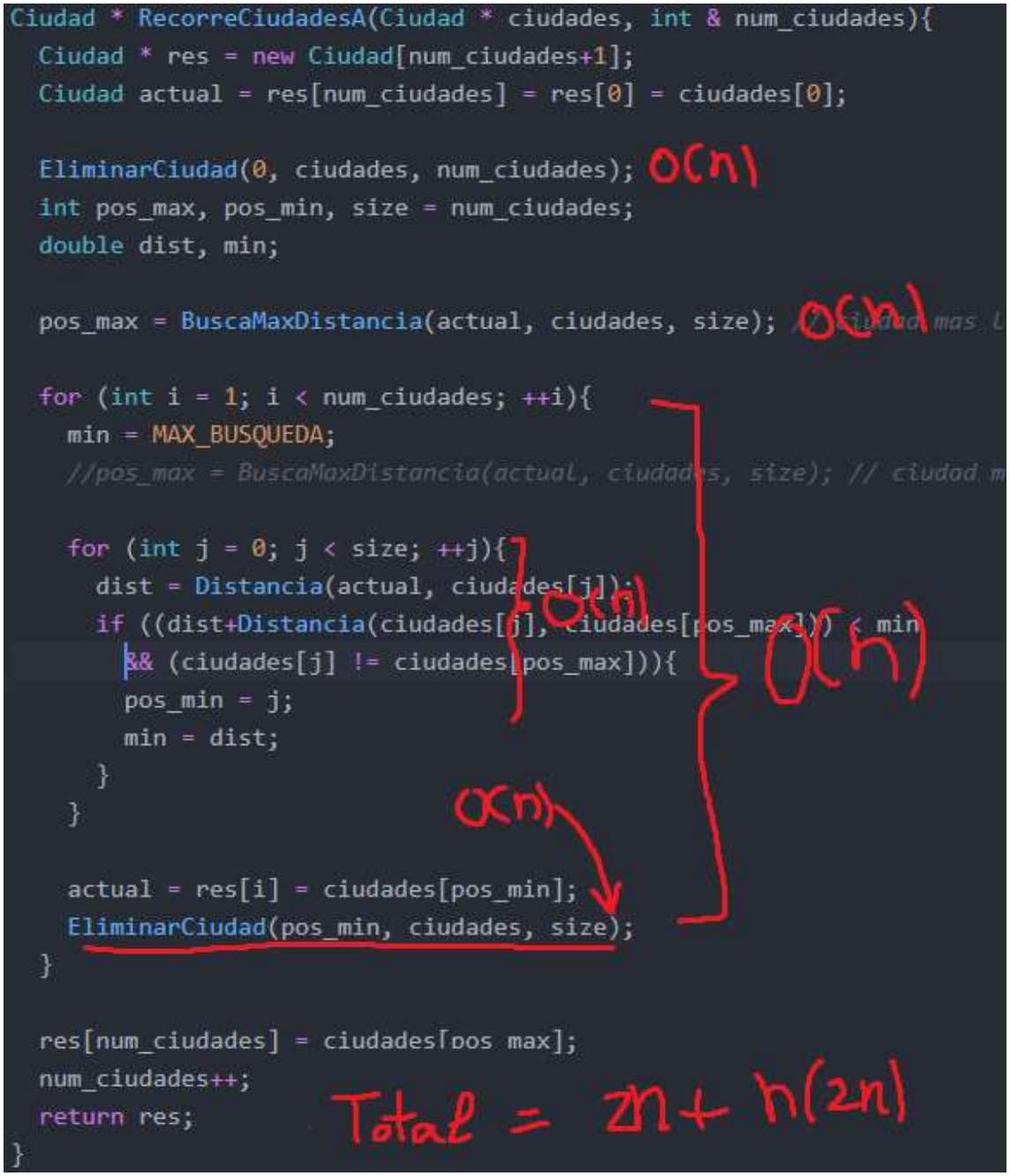
añadir actual a la solucion

eliminar de ciudades a recorrer

devolver recorrido

## Análisis de la eficiencia teórica, empírica e híbrida

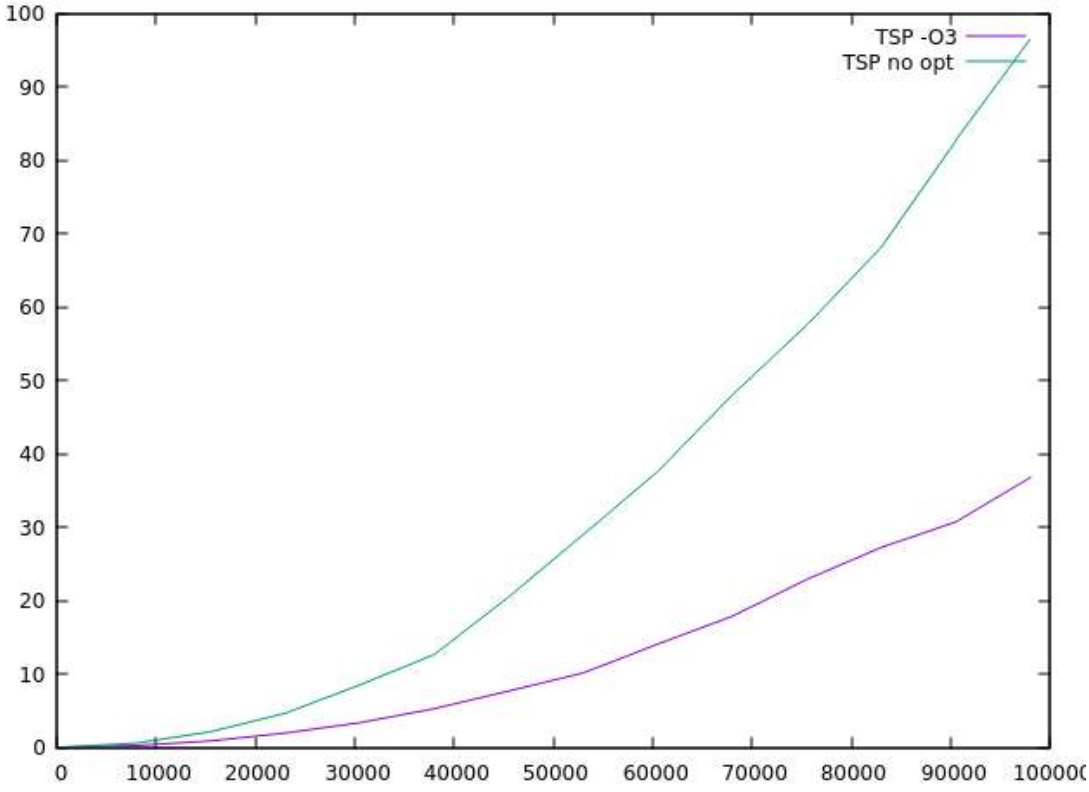
Para la eficiencia teórica analizamos el código y vemos que El número de operaciones 2n + n*(2*n) esto satura a O(n^2), por eso este algoritmo es cuadrático.



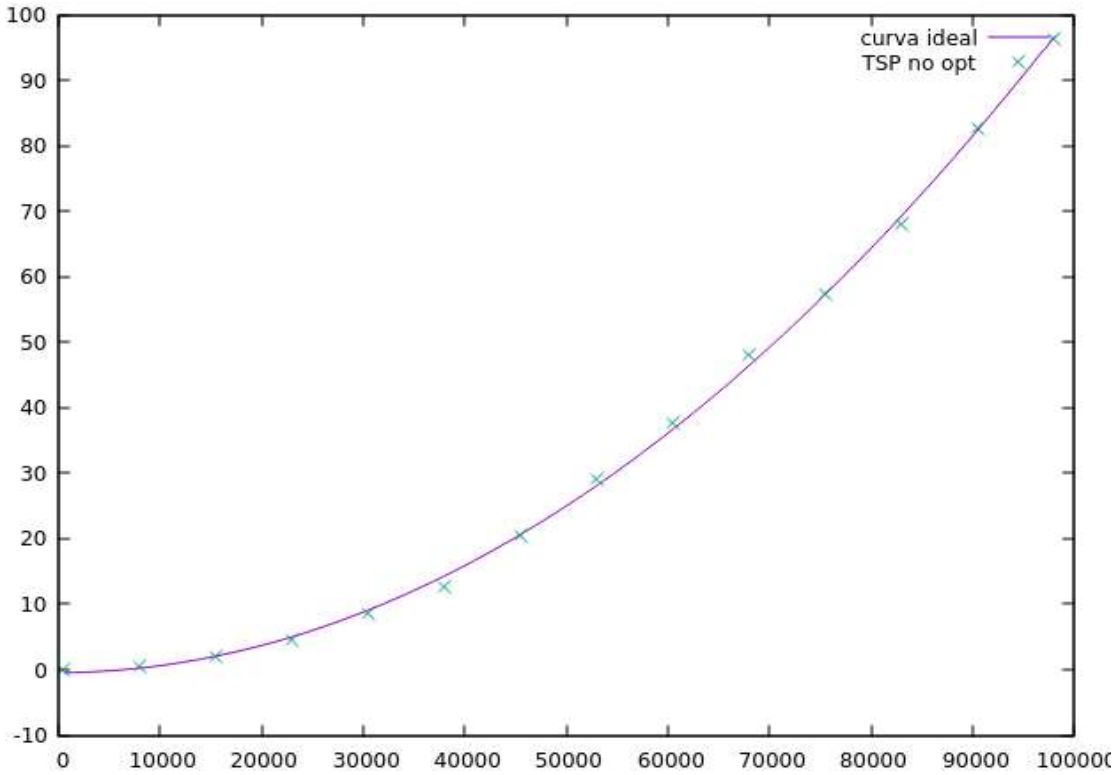
Para la eficiencia empírica se han llevado a cabo las siguientes ejecuciones:

| Tamaño | Tiempo(seg) | Tiempo_O3(seg) |
|---|---|---|
| **500** | 0.0022628 | 0.0010828 |
| **8000** | 0.560301 | 0.230045 |
| **15500** | 2.12466 | 0.860803 |
| **23000** | 4.61507 | 1.93805 |
| **30500** | 8.48417 | 3.34006 |
| **38000** | 12.6273 | 5.25399 |
| **45500** | 20.4889 | 7.6582 |
| **53000** | 28.9671 | 10.1153 |
| **60500** | 37.5709 | 14.047 |
| **68000** | 47.9777 | 17.8327 |
| **75500** | 57.5057 | 22.8542 |
| **83000** | 68.1205 | 27.2306 |
| **90500** | 82.6114 | 30.7036 |
| **98000** | 96.4003 | 36.7367 |

Así pues, podemos ver en esta tabla que la versión optimizada es bastante más eficiente que la versión sin optimizar, siendo de media un 262% más eficiente la versión optimizada con –O3



Para el análisis de la eficiencia hibrida hacemos uso de la herramienta Gnuplot. Definimos una función f(x) y la representamos junto a la curva de las ejecuciones empíricas. En la siguiente grafica se puede observar la comparación con la curva ideal de las ejecuciones vistas previamente.




***Final set of parameters*** ***Asymptotic Standard Error***

========== ==========

***a0*** ***= 1.00408e-08*** ***+/- 2.981e-10 (2.969%)***

***a1*** ***= 7.52904e-06*** ***+/- 3.043e-05 (404.2%)***

***a2*** ***= -0.470629*** ***+/- 0.6455 (137.2%)***

***correlation matrix of the fit parameters:***

| | ***a0*** | ***a1*** | ***a2*** |
|---|---|---|---|
| ***a0*** | ***1.000*** | | |
| ***a1*** | ***-0.965*** | ***1.000*** | |
| ***a2*** | ***0.698*** | ***-0.834*** | ***1.000*** |

Siendo así pues ***f(x) = 1.00408e-08*x^2 + 7.52904e-06*x - 0.470629***

## Comparación con la solución óptima de los archivos TOUR

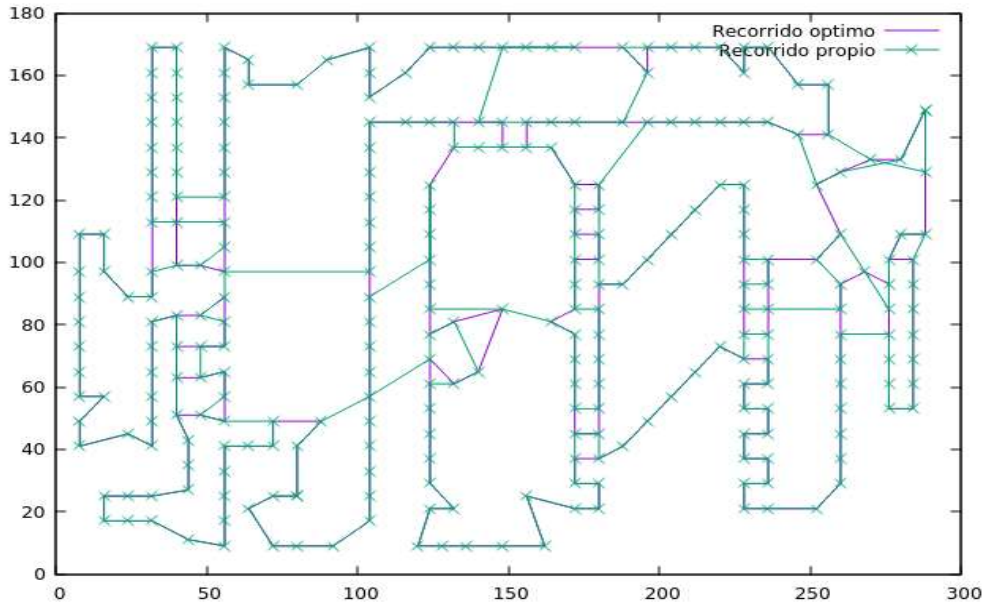
Para llevar a cabo esta comparación vamos a medir la distancia total recorrida para cada secuencia de ciudades dada. Así pues, nos queda lo siguiente:

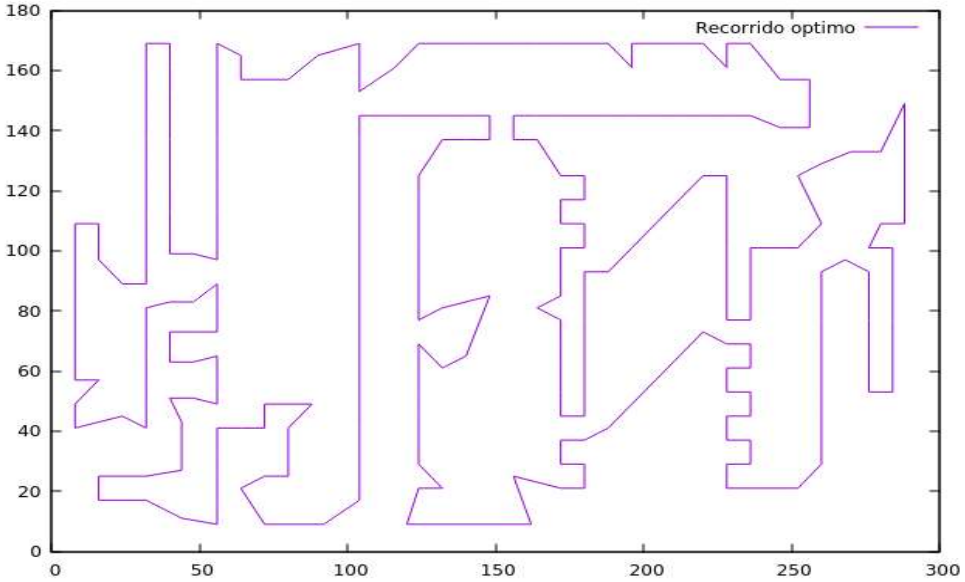
- Ulysses16:
    - Camino total óptimo:  74.1087
    - Camino total nuestro: 92.2465
    - Nuestro algoritmo se aproxima en un 80.34% al ideal
- Ulysses22:
    - Camino total óptimo:  75.6651
    - Camino total nuestro: 119.276
    - Nuestro algoritmo se aproxima en un 63.45% al ideal
- Att48:
    - Camino total óptimo:  33523.7
    - Camino total nuestro: 74124.3
    - Nuestro algoritmo se aproxima en un 45.23% al ideal
- A280:
    - Camino total óptimo:  2586.77
    - Camino total nuestro: 2800.73
    - Nuestro algoritmo se aproxima en un 92.36% al ideal

Viendo los datos obtenidos a partir de los ficheros que nos proporciona la práctica tenemos que nuestro algoritmo se aproxima en media un 70.345%, es decir, realiza un 29.66% más que el algoritmo óptimo.

A continuación, veremos una representación gráfica del recorrido de las ciudades, para el ejemplo a280.opt.TOUR, el más mayor de los ejemplos en cuanto a número de ciudades. Para esto hemos hecho uso de la herramienta de representación gráfica Gnuplot.

Esta sería una comparación de ambos recorridos, el óptimo y nuestro recorrido propio. Las cruces ‘x’ son las ciudades a recorrer y en rosa aparece nuestro recorrido mientras que en verde aparece el recorrido óptimo. Podemos ver que son casi idénticos, salvo pequeños detalles, lo cual confirma el porcentaje de aproximación demostrado teóricamente. También se muestran los recorridos por separado (imágenes 2, 3).


Recorrido optimo
Recorrido propio
0
20
40
60
80
100
120
140
160
180
0
50
100
150
200
250
300


Recorrido optimo
0
20
40
60
80
100
120
140
160
180
0
50
100
150
200
250
300


Recorrido propio
0
20
40
60
80
100
120
140
160
180
0
50
100
150
200
250
300