práctica 3:

ALGORITMOS VORACES (GREEDY)





**ÍNDICE**

**1. Descripción del problema.**

**2. Viajante de Comercio basado en cercanía.**

**3. Viajante de Comercio basado en inserción.**

**4. Viajante de Comercio: Estrategia propia.**

**5. Comparativa Viajante de Comercio.**

**6. Contenedores en un barco.**

**7. Conclusiones.**

1. **Descripción del problema.**

Para esta práctica lo que se nos pide es resolver 2 problemas. Uno de estos problemas es común a todos los grupos, es decir, debe ser resuelto por todo grupo que realice la práctica, mientras que el otro es uno de los 4 problemas descritos en el guion de la práctica y asignado por sorteo.

Así pues, el primero de los problemas es *El Viajante de Comercio*. Este problema consiste en que 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.

Para dicho problema se nos pedía realizar tres versiones diferentes:

* Basada en cercanía: estrategia basada en el vecino más cercano, es decir, dada una ciudad inicial *v0*, se agrega como ciudad siguiente aquella *vi* (no incluida en el circuito) que se encuentre más cercana a *v0*.
* Basada en inserción: la idea es comenzar con un recorrido parcial, que incluya algunas de las ciudades, y luego extender este recorrido insertando las ciudades restantes mediante algún criterio de tipo voraz.
* Estrategia propia: versión distinta del problema diseñada por cada grupo.

Respecto al segundo problema, a nuestro grupo se le ha asignado por sorteo el problema 3.4 del guion de prácticas, denominado *Contenedores en un barco*. Dicho problema nos plantea la siguiente situación:

Se tiene un buque mercante cuya capacidad de carga es de K toneladas y un conjunto de contenedores *c1,...,cn* cuyos pesos respectivos son *p1,...,pn* (expresados también en toneladas).

Teniendo en cuenta que la capacidad del buque es menor que la suma total de los pesos de los contenedores, se nos pedía diseñar un algoritmo que maximizara el número de contenedores cargados (demostrando su optimalidad) y otro que intentara maximizar el número de toneladas cargadas.

**2. Viajante de Comercio basado en cercanía.**

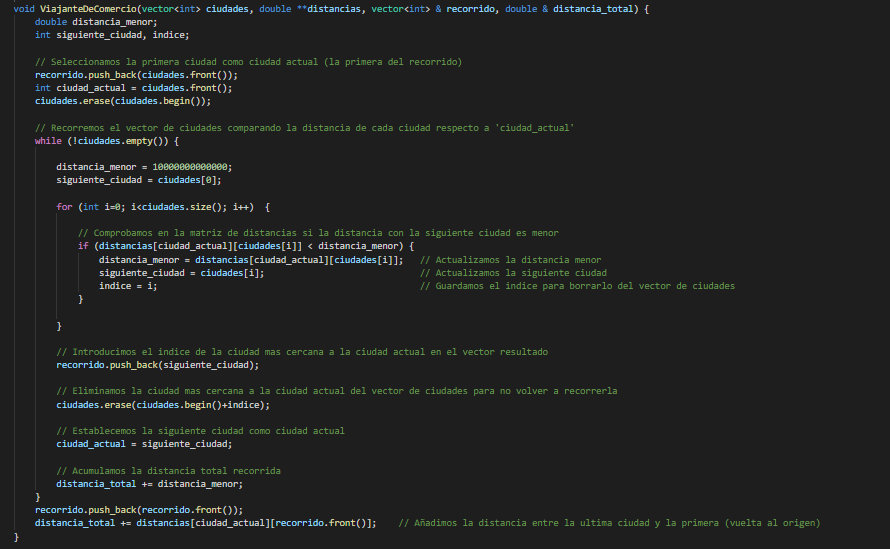
**Descripción detallada del algoritmo**

El objetivo del algoritmo es encontrar una ruta o recorrido que pase por todas las ciudades siguiendo el criterio del vecino más cercano. Para ello, dada una lista de ciudades y seleccionando la ciudad número **1** como primera ciudad del recorrido (ciudad actual), la siguiente ciudad en recorrer será aquella que tenga la mínima distancia con respecto a la ciudad actual y no haya sido recorrida antes. Dicha distancia se calcula mediante la distancia Euclídea:

C:\Users\eduil\AppData\Local\Microsoft\Windows\INetCache\Content.Word\euclides.png

Tras la elección de la siguiente ciudad, esta se establece como ciudad actual y se realiza el mismo proceso de elección anterior con respecto a ella.

**Cálculo de la eficiencia teórica**

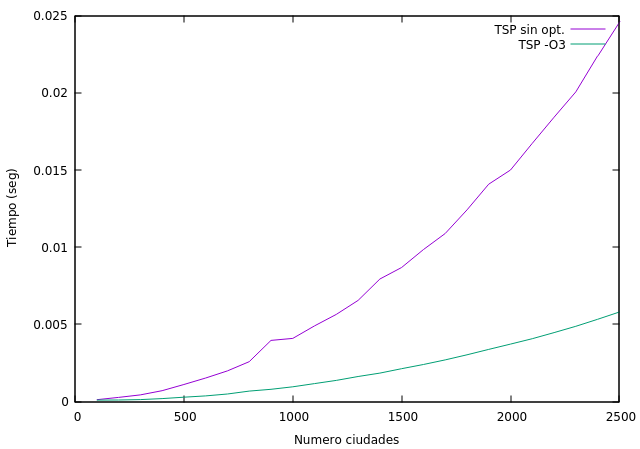
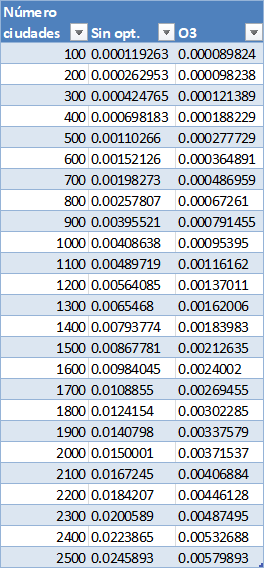


Captura de pantalla del código del algoritmo con comentarios explicativos

Como podemos observar, la función ejecuta dos bucles anidados. El primero de ellos (el externo: bucle***while***) se ejecuta tantas veces como ciudades haya puesto que en cada iteración se escoge una ciudad y, por tanto, se elimina del vector de las ciudades para que no se vuelva a recorrer. El segundo (el interno: bucle ***for***) se ejecuta tantas veces como ciudades queden por recorrer. En su interior nos encontramos con una sentencia condicional que contiene solo operaciones elementales, por lo tanto, no es significativo a la hora de definir la eficiencia teórica.

Con este análisis, podemos reconocer que se trata de una eficiencia de orden **O(n^2)**.

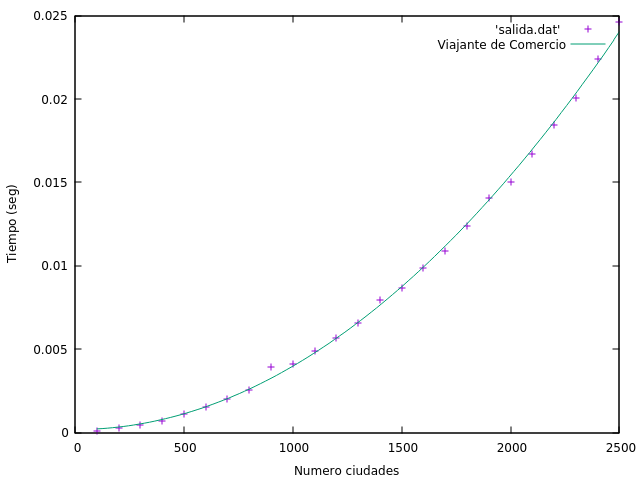
**Cálculo de la eficiencia empírica**



Para calcular la eficiencia empírica tanto de la versión sin optimizar como de la versión optimizada (-O3), se ha ejecutado el programa con distinto número de ciudades. El estudio ha sido realizado partiendo de **100** ciudades hasta **2500** incrementando el número en **100 en cada ejecución**.

**Cálculo de la eficiencia híbrida**

Para calcular la eficiencia híbrida, necesitamos ajustar los datos obtenidos en la eficiencia empírica para f(x)=ax^2 + bx + c, haciendo uso de la herramienta Gnuplot, y se obtienen los siguientes datos:

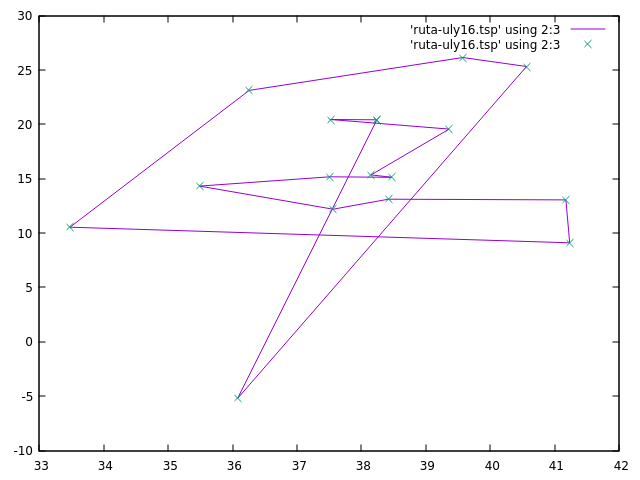


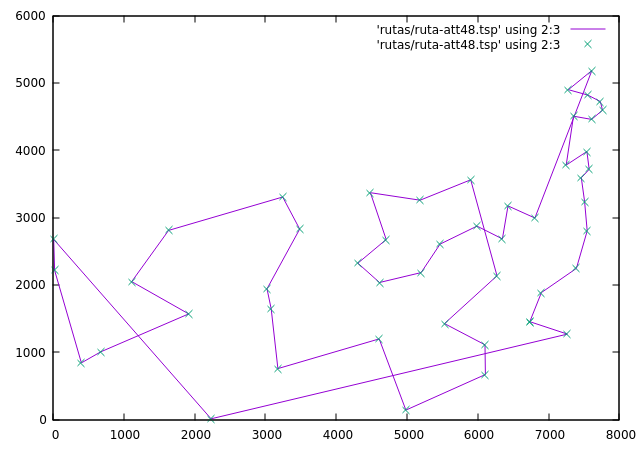
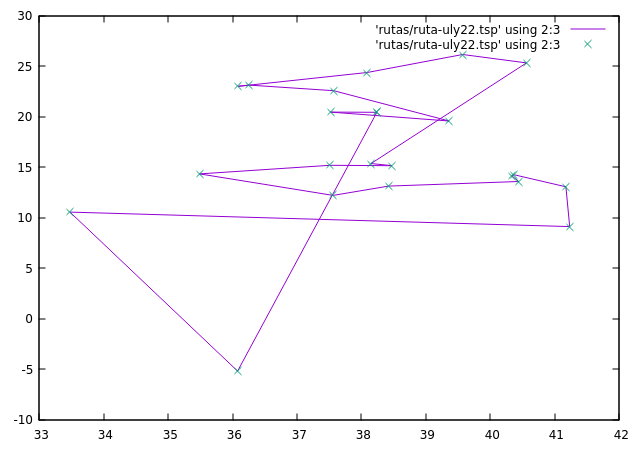
**Comparación con la solución óptima de los archivos TOUR**

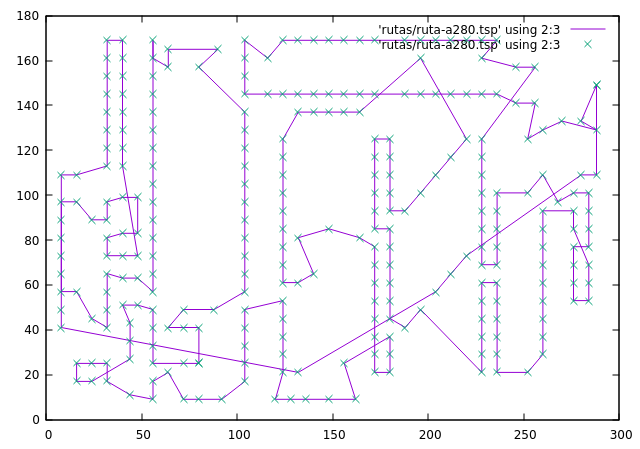
Para llevar a cabo esta comparación vamos a medir la distancia total recorrida para cada secuencia de ciudades dada. Así pues, nos queda lo siguiente:

* Ulysses16:
  + Camino total óptimo: 74.1087
  + Camino total nuestro: 104.735
  + Nuestro algoritmo se aproxima en un 70.76% al ideal
* Ulysses22:
  + Camino total óptimo: 75.6651
  + Camino total nuestro: 89.6408
  + Nuestro algoritmo se aproxima en un 84.40% al ideal
* Att48:
  + Camino total óptimo: 33523.7
  + Camino total nuestro: 40526.4
  + Nuestro algoritmo se aproxima en un 82.72% al ideal
* A280:
  + Camino total óptimo: 2586.77
  + Camino total nuestro: 3148.11
  + Nuestro algoritmo se aproxima en un 82.17 % 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 80.01%, es decir, realiza un 19.99% más que el algoritmo óptimo. A continuación se adjuntas capturas de pantalla del resultado gráfico de los datos anteriores.

****



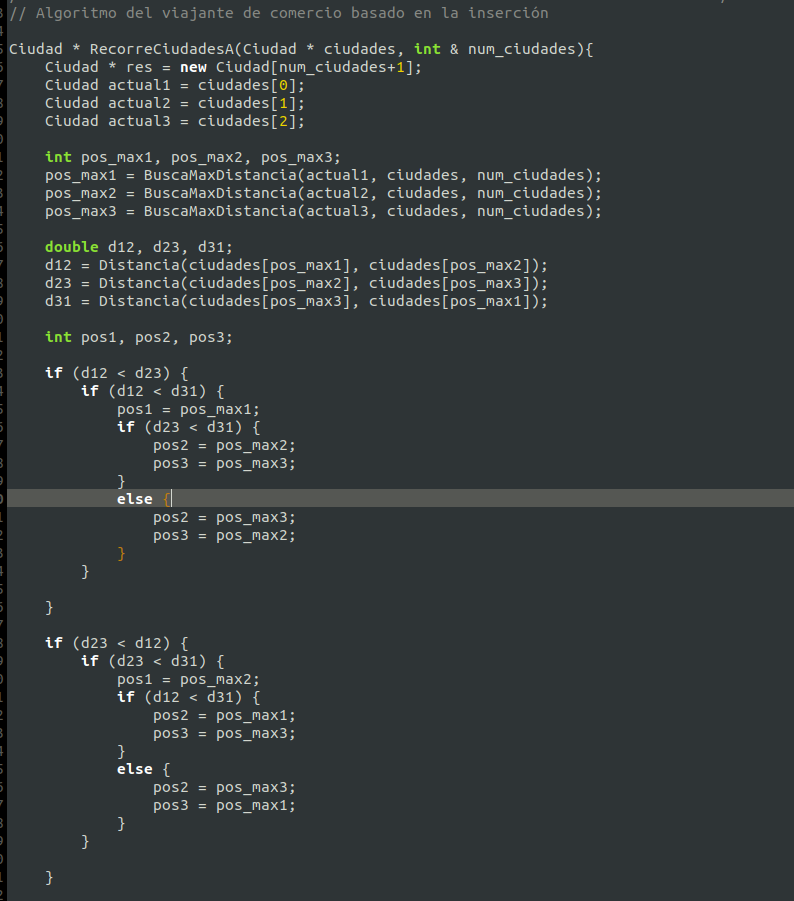


**2. Viajante de Comercio basado en inserción.**

**Descripción detallada del algoritmo**

En esta práctica se nos pide implementar varias soluciones para el problema del viajante de comercio, en este caso se trata de la segunda versión, basada en el método de inserción.

En las estrategias de inserción, se comienza con un recorrido parcial, que incluya algunas de las ciudades, y luego extender el recorrido añadiendo al mismo las ciudades restantes. En primer lugar vamos a explicar cómo se construye el recorrido inicial. Una opción sería la siguiente, vamos a escoger las tres ciudades que formen un triángulo lo más grande posible. Para ello, podemos escoger por ejemplo la ciudad que se encuentre más al Oeste, posteriormente la que está más al Este, y por último la ciudad que se encuentra más al Norte. De esta forma nos aseguramos de que el tamaño del triángulo sea lo mayor posible. Una vez elegido el recorrido inicial, debemos recorrer el vector de ciudades no seleccionadas y se van insertando de una en una en cada posición posible. Por último, tendremos que hacer uso del criterio de inserción denominado inserción más económica, que consiste en elegir, de entre todas las ciudades no visitadas, aquella que provoque el menor incremento posible en la longitud total del circuito. Es decir, para cada ciudad no visitada se busca la posición que resulta en menor incremento de la longitud del circuito, y se selecciona la ciudad (y posición) que genera el menor incremento en cada paso. Por tanto, seleccionaremos aquella ciudad que nos proporcione el mínimo de los mínimos calculados para cada una de las ciudades. Todo el código de insercion.cpp se encuentra documentado para una mayor comprensión del funcionamiento del mismo. A continuación se añaden capturas de pantalla del código en cuestión, pero solo de la parte que corresponde al algoritmo en sí mismo. Para más información consulte el archivo de nombre insercion\_damian.cpp.



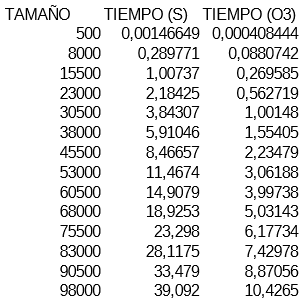


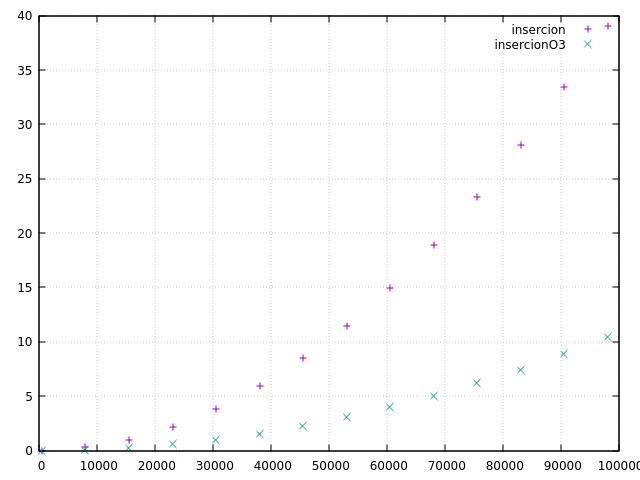
**Cálculo de la eficiencia teórica**

Para la eficiencia teórica solamente tenemos que analizar el código (disponible en insercion.cpp) para darnos cuenta de que el número de operaciones 2n + n\*(2\*n) satura a O(n²), por lo que el algoritmo es cuadrático.

**Cálculo de la eficiencia empírica**

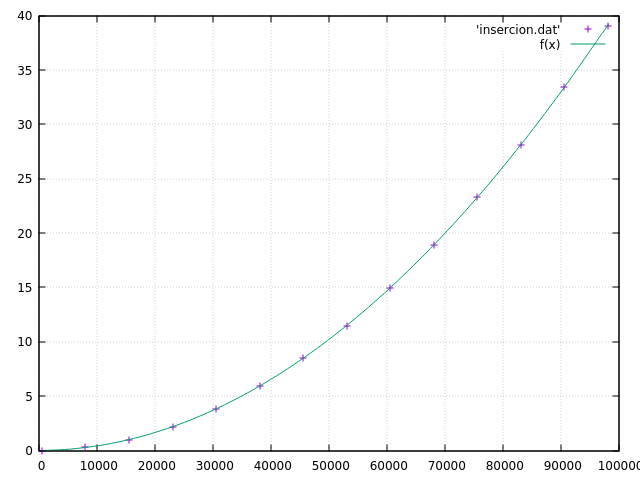
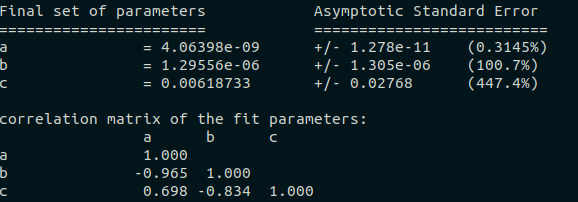
Para la eficiencia empírica se han llevado a cabo las siguientes ejecuciones:





Como podemos observar, el tiempo al compilar utilizando la opción de optimización es O3 es notablemente más bajo que el tiempo resultante de una compilación normal. Aproximadamente unas 4 veces menor. A continuación se añade una comparación realizada mediante la herramienta gnuplot, que se usará más adelante para los cálculos de eficiencia híbrida.

**Cálculo de la eficiencia híbrida**

Para el análisis de la eficiencia híbrida vamos a hacer uso otra vez de la herramienta conocida como gnuplot. Para ello definiremos una función **f(x) = ax² + bx + c**, y la representaremos junto a la curva de la ejecuciones empíricas. A continuación se muestra una gráfica con la comparación de la curva ideal.

Por tanto el resultado sería: **f(x) = 4.06398e-09\*x² + 1,29556e-06\*x + 0,00618733**

**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: 74.01087
* Camino total nuestro: 100
* Nuestro algoritmo se aproxima en un 74,0109 %

- Ulysses22:

* Camino total óptimo: 75,6651
* Camino total nuestro: 115
* Nuestro algoritmo se aproxima en un 65,7957 %

- Att48:

* Camino total óptimo: 33523,7
* Camino total nuestro: 61949
* Nuestro algoritmo se aproxima en un 54,1214 %

- A280:

* Camino total óptimo: 2586,77
* Camino total nuestro: 6525
* Nuestro algoritmo se aproxima en un 39,64 %

Viendo los datos obtenidos a partir de los ficheros que nos proporciona la práctica tenemos que nuestro algoritmo se aproxima en media un 58,392%.**4. Viajante de Comercio: Estrategia propia.**

**Descripción del problema**

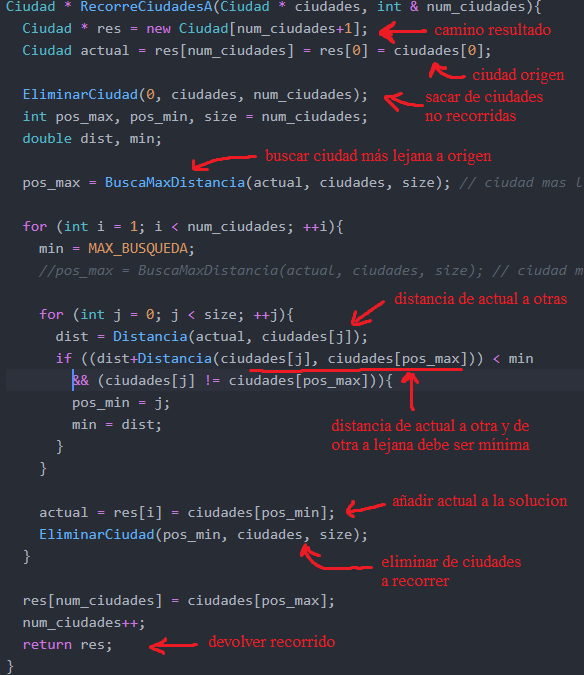
En esta práctica se nos pide implementar varias soluciones para el problema del viajante de comercio, en este caso esta es la tercera versión, la versión inventada por los alumnos.

**Descripción de la solución**

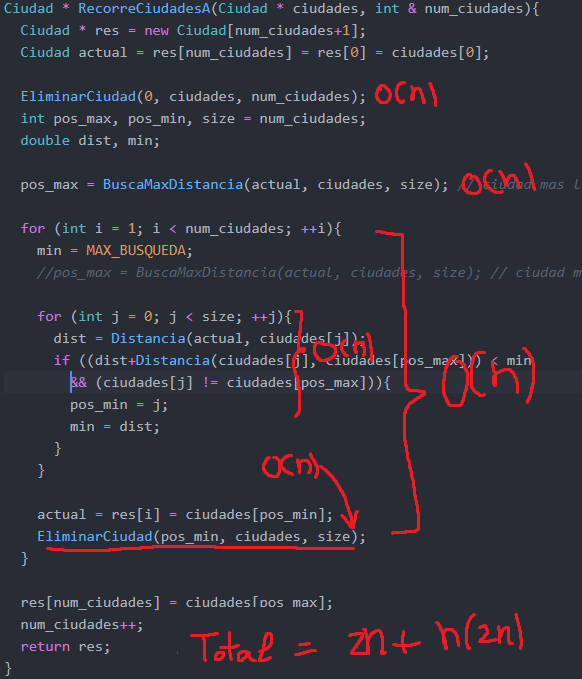
La solución desarrollada se inspira en el algoritmo A\* (estrella) fuertemente usada en IA para búsqueda de caminos evitando obstáculos. Como algoritmo que Greedy que es este, también evalúa en cada momento la mejor opción de las disponibles, al igual que el algoritmo tsp por cercanía del cual también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



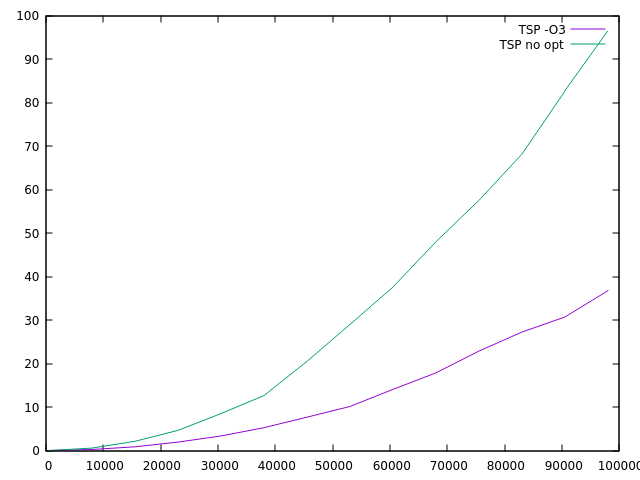
**Análisis de la eficiencia teórica, empírica e híbrida**

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.

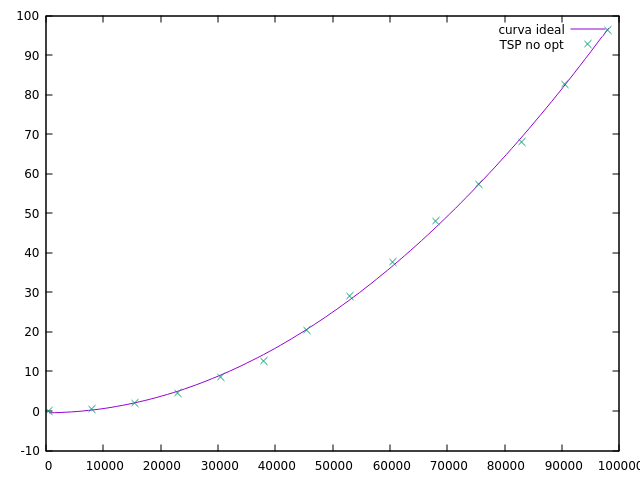
Para la eficiencia empírica se han llevado a cabo las siguientes ejecuciones:

|  |  |  |  |
| --- | --- | --- | --- |
| **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**

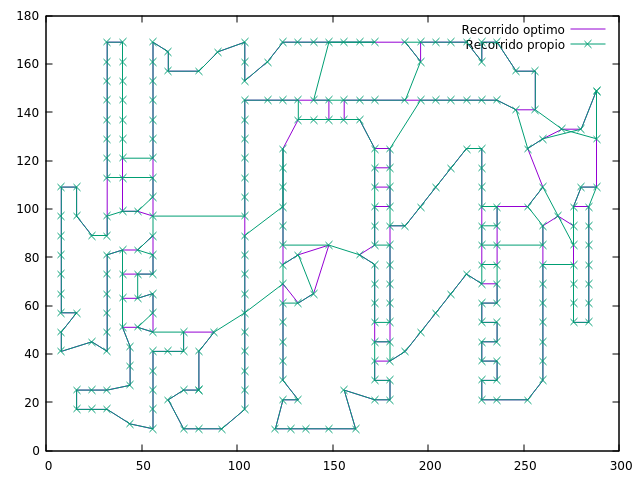
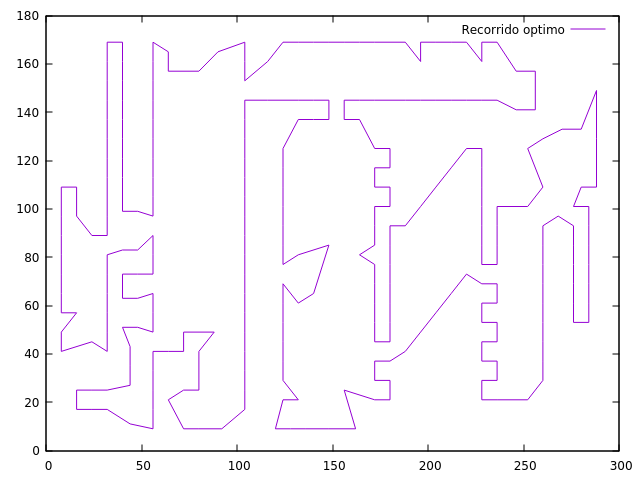
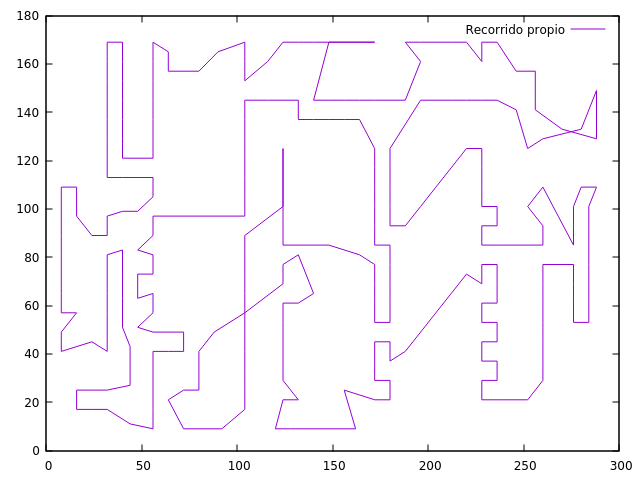
Para llevar a cabo esta comparación vamos a medir la distancia total recorrida para cada secuencia de ciudades dada. Así pues, nos queda lo siguiente:

* 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).



**6. Contenedores en un barco.**

**Descripción detallada del algoritmo.**

a)

***seleccion*** -> devuelve el contenedor de menor peso, un bucle que compara todos los contenedores y devuelve el menor.

***crearBuque*** -> devuelve un vector con las cargas, un bucle que rellena el vector mientras que el peso no supere a k.

***genera vector*** -> genera un vector del 1 al 100.

b)

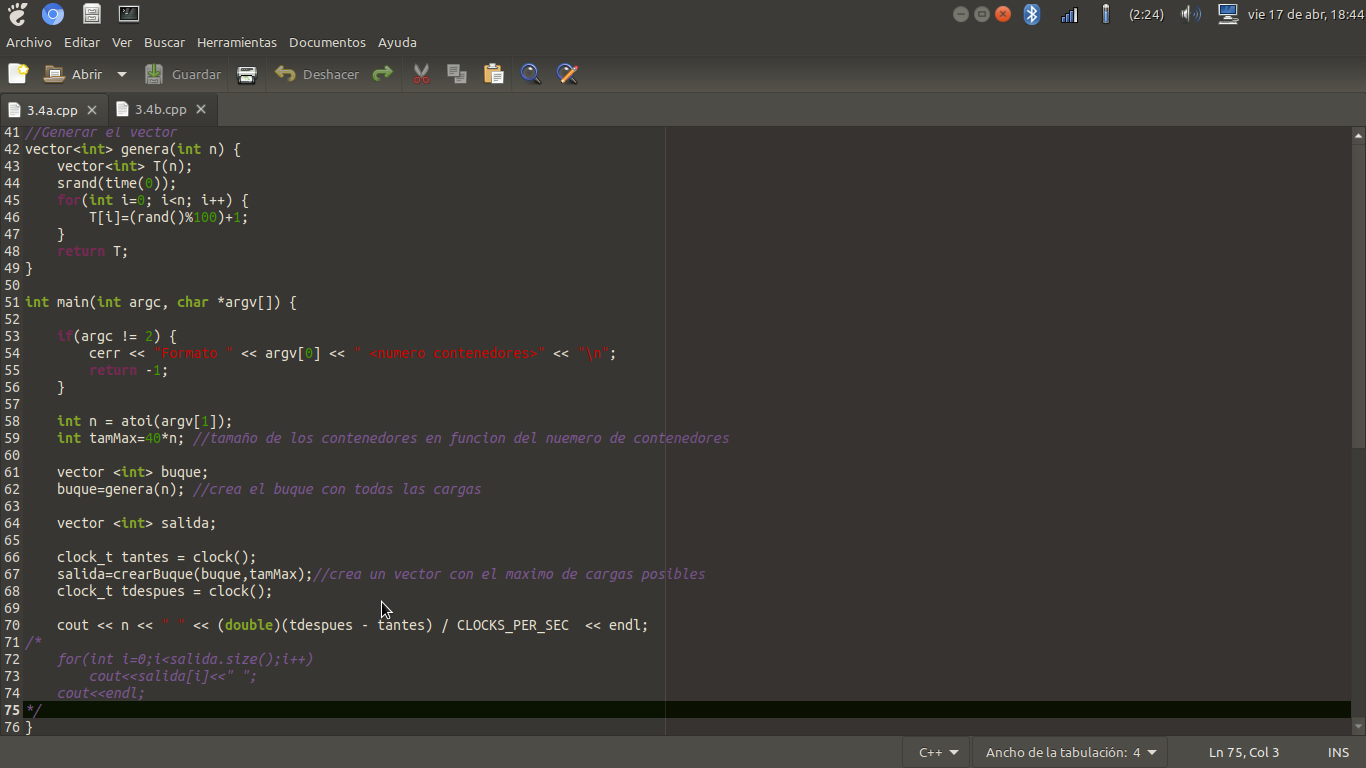
***seleccion*** -> devuelve el contenedor de peso máximo, un bucle que compara todos los contenedores y devuelve el máximo.

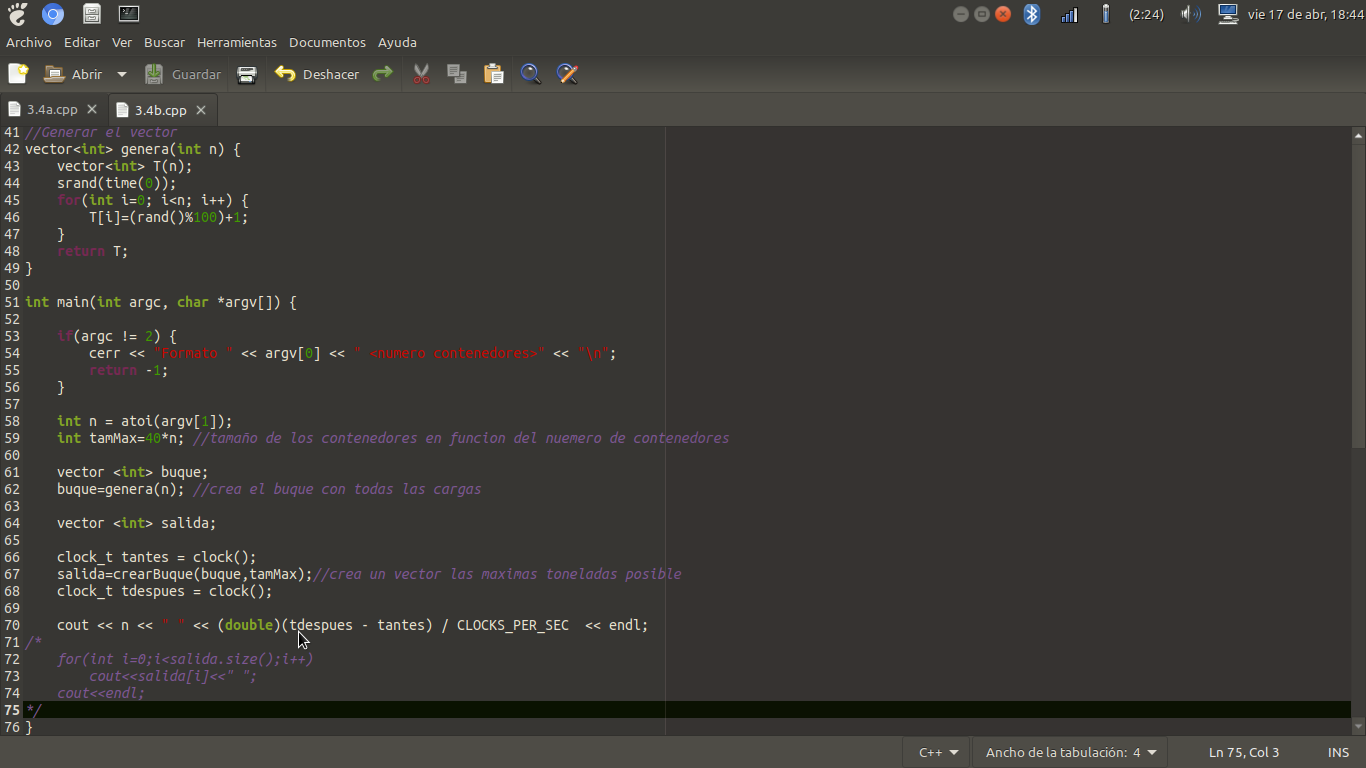
***crearBuque*** -> devuelve un vector con las cargas, un bucle que rellena el vector mientras que el peso no supere a k.

***genera vector*** -> genera un vector del 1 al 100.

**Cálculo de la eficiencia teórica**

La eficiencia Teórica de este algoritmo es **O(n³)** ya que son tres bucles(crear vector, buscar el máximo/más peso y crear vector de salida).

a)

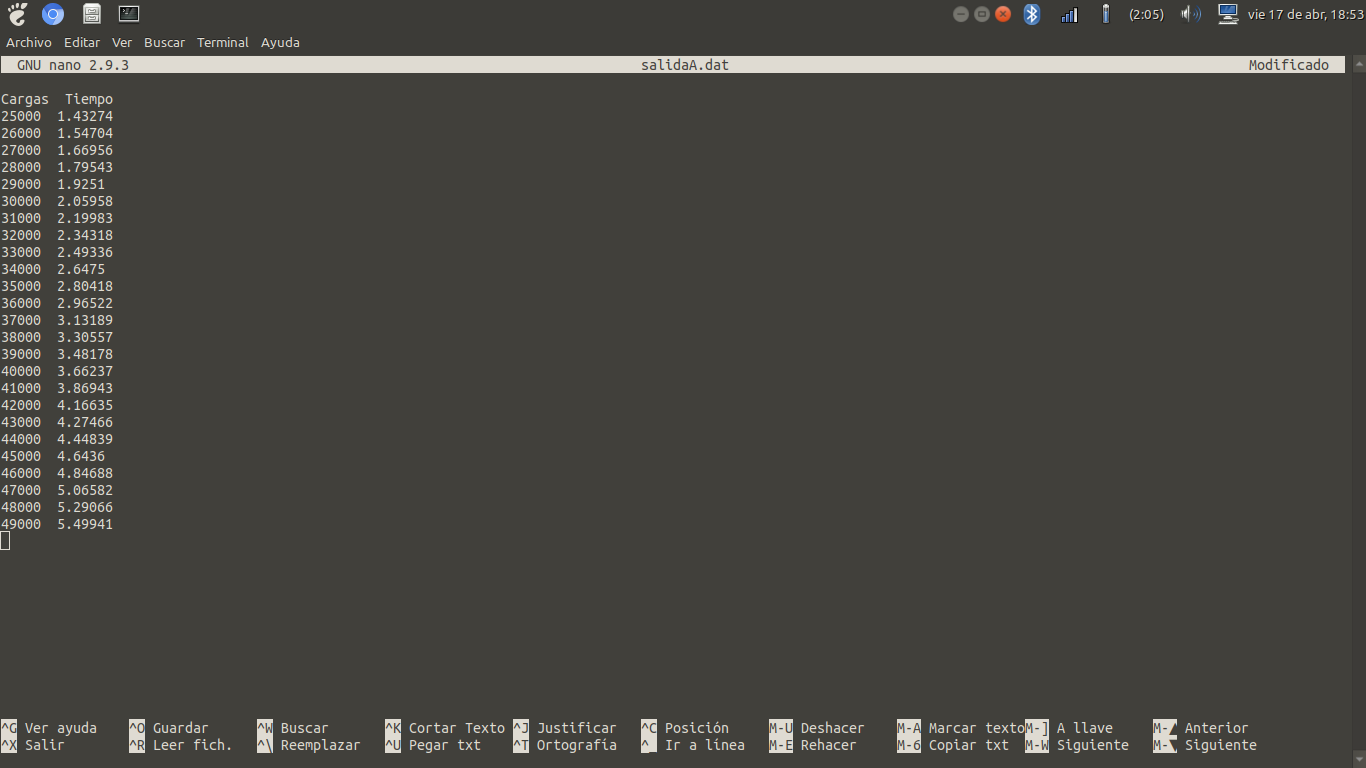
b)

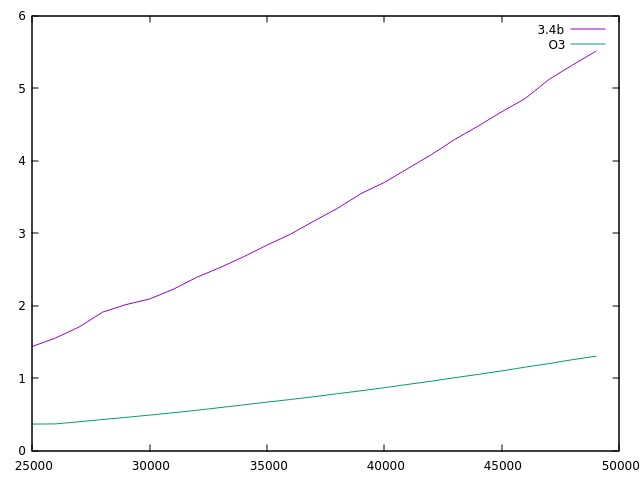
**Cálculo de la eficiencia empírica**

Al ejecutar el algoritmo nos damos cuenta que con forme aumenta el número de elementos del vector aumenta el tiempo de finalización del algoritmo.

a)

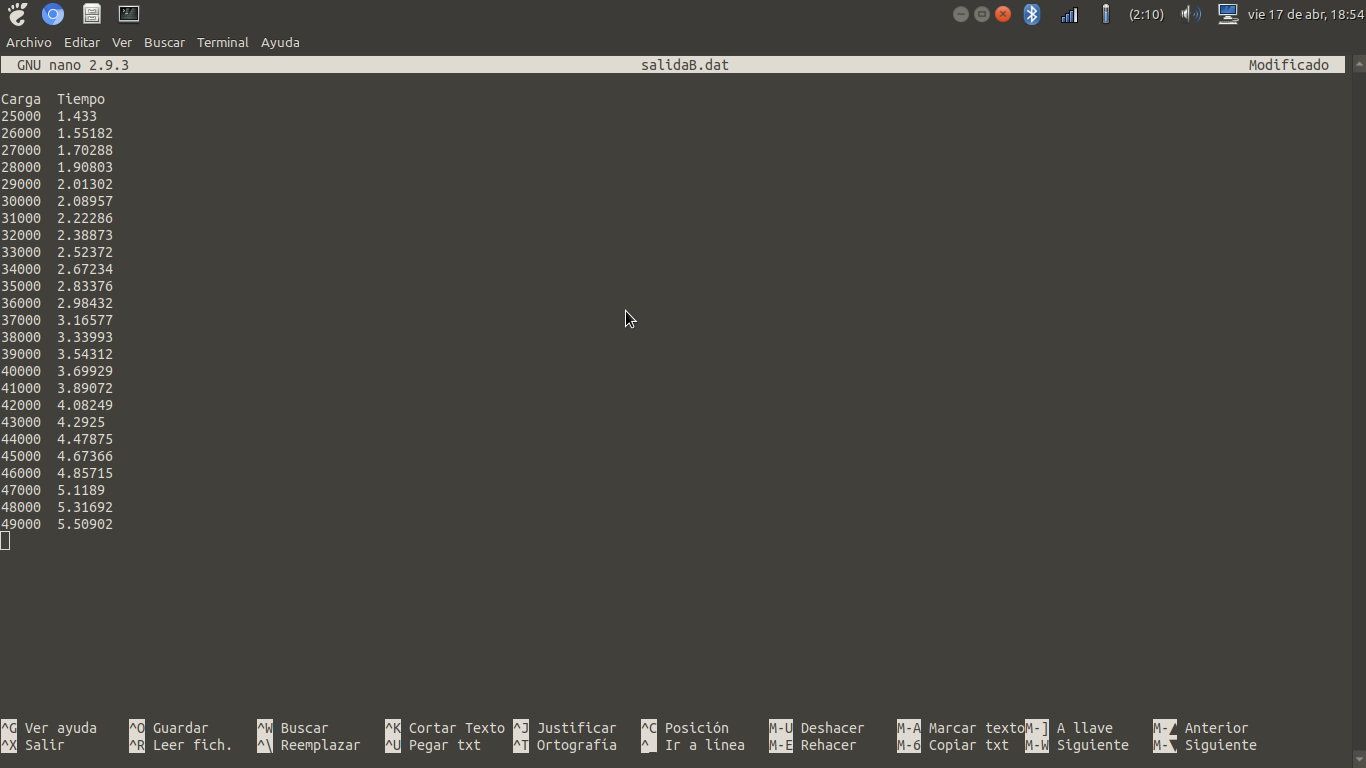
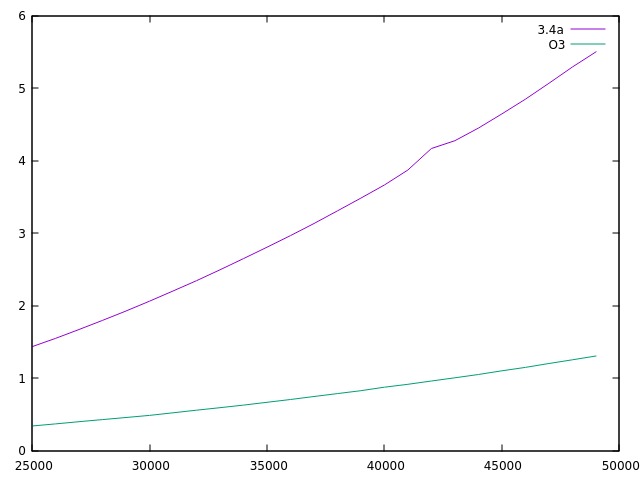
La grafica de estos valores es:





b)

La grafica de estos valores es:



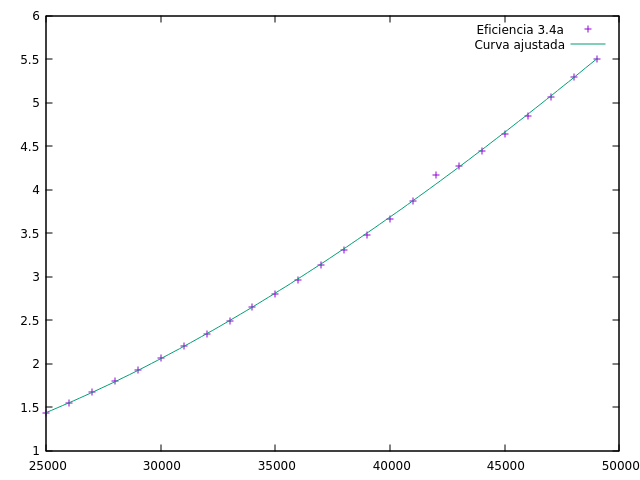
**Cálculo de la eficiencia híbrida**

Al ser la eficiencia Teórica: O(n³) → T (n) = a0×n³+ a1×n² + a2×n + a3

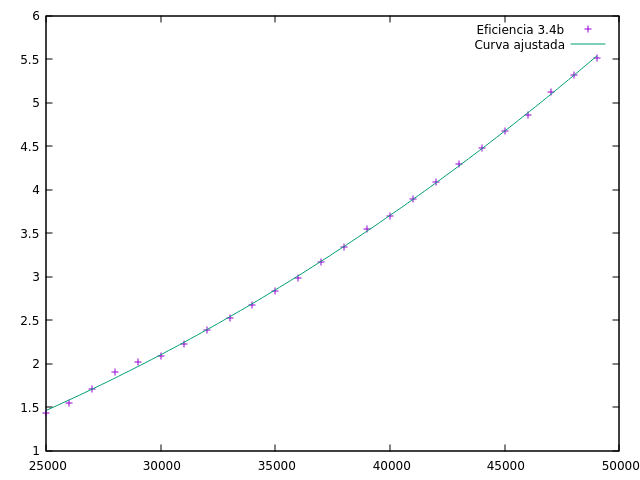
Tras unas pocas iteraciones, nos sale los resultados del proceso de regresión de la función a ajustar.

La gráfica comparando las dos funciones (sin ajustar y la ajustada) es:

a)



b)



**7. Conclusiones.**

Analizando el problema del viajante de comercio nos damos cuenta de que es un problema NP completo. El orden de eficiencia para resolver este problema por fuerza bruta sería exponencial O(n!), un orden de eficiencia factorial implica (para un tamaño mayor que un umbral especifico) un tiempo demasiado elevado como para ser considerado una solución.

Intentando resolver el problema del viajante de comercio mediante las técnicas Greedy, se han implementado y evaluado 3 algoritmos. Comparando estos algoritmos vemos que el algoritmo de cercanía es el más eficiente en tiempo con unos tiempos bastante aceptables y equilibrados para el recorrido que nos da, evaluando dicho recorrido y comparándolo con el recorrido óptimo tenemos una aproximación del 80.99% que es bastante buena. El segundo mejor algoritmo en aproximación (de media) al recorrido óptimo, es el algoritmo de estrategia propia con una aproximación del 70.345%, aunque es el más ineficiente en tiempo llegando a tardar aproximadamente el triple que el algoritmo de inserción. El algoritmo de inserción es bastante eficiente en tiempo, pero los resultados que nos da no son tan buenos como los otros, pues se aproxima en un poco más del 50% al algoritmo óptimo lo cual implica que haríamos el doble de camino si usásemos este algoritmo.

La conclusión que obtenemos de la realización de esta práctica es que en el mundo en el que vivimos hay problemas elementales y naturales cuyas resoluciones son imprescindibles en nuestro día a día, pero que dichos problemas no tienen una solución óptima trivial ni tan eficiente como nos gustaría.

Los algoritmos Greedy nos dan así pues una solución aceptablemente eficiente en tiempo de ejecución y una solución que no siempre es la óptima pero que se aproxima bastante a esta y nos permite optimizar los recursos de los que disponemos, así como el coste de la realización de la tarea que busquemos llevar a cabo, aunque no siempre se pueden aplicar estos algoritmos Greedy ni son válidos para cualquier problema que se tenga que resolver.