Algorítmica
Práctica 3. Greedy TSP
2018-2019

UNIVERSIDAD
DE GRANADA

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

ÍNDICE

1. Introducción

En su formulación más sencilla, el problema del viajante de comercio (TSP, por Traveling Salesman Problem) se define como sigue: 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 mınima. Más formalmente, dado un grafo G, conexo y ponderado, se trata de hallar el ciclo hamiltoniano de mınimo peso de ese grafo. Una solución para TSP es una permutación del conjunto de ciudades que indica el orden en que se deben recorrer.

Para el cálculo de la longitud del ciclo no debemos olvidar sumar la distancia que existe entre la última ciudad y la primera (hay que cerrar el ciclo). Por su interés teórico y práctico, existe una variedad muy amplia de algoritmos para abordar la solución del TSP y sus variantes (siendo un problema NP-Completo, el diseño y aplicación de algoritmos exactos para su resolución no es factible en problemas de cierto tamaño). Nos centraremos en una serie de algoritmos aproximados de tipo greedy y evaluaremos su rendimiento en un conjunto de instancias del TSP. Para el diseño de estos algoritmos, utilizaremos dos enfoques diferentes:

a) Estrategias basadas en alguna noción de cercanıa
    Partes 1 y 3 de esta documentación.

b) Estrategias de inserción.
    Partes 2 de esta documentación.

2. Parte 1 (VMC)

    a. Planteamiento

Esta parte se basa en una aproximación mediante el Vecino Más Cercano, en la cual, cada vez que el viajero visita una ciudad, la próxima ciudad a visitar se elige mediante el criterio de cercanía, es decir, de entre todas las ciudades disponibles, se elige la que menos distancia tenga, independientemente de los demás criterios.

    b. Elementos de la solución del problema

Conjunto Candidatos	Todas las ciudades
Conjunto Seleccionados	Las ciudades no visitadas
Función Solución	Ruta con distancia mínima entre ciudades
Función Selección	La ciudad no visitada con menor distancia desde donde está el Viajero
Función Factibilidad	La distancia de la ruta es mínima
Función Objetivo	Elegir la ciudad origen mediante la cual pase por todas las ciudades con una distancia mínima

c. Pseudocódigo

```
Lectura de ciudades
rutaCandidate = rutaLeida
Para cada ciudad origen
    ciudadCandidata = ciudadContiguaOrigen
    Mientras haya ciudades sin visitar
        Calcular distancia a cada ciudad sin visitar
        Redondear distancia
        Si coste mejora
            Actualiza ciudad candidata
        Añade ciudad candidata
    Cuando se hayan visitado todas
        Añadir ciudad origen
    Calcular coste ruta
    Si coste mejora
        Actualizar ruta candidata
```

Como se puede ver en el pseudocódigo anterior, primero se inicializa la ruta candidata a la ruta leída por fichero, acto seguido, se toma cada ciudad como origen y se calcula su ruta más óptima.

Entonces, para cada ciudad origen, se calcula la distancia mínima a cada ciudad no visitada y cuando se tenga, se añade como siguiente ciudad a visitar, en el momento en el que no haya más ciudades por visitar, se añade la ciudad origen para cerrar el ciclo.

Cuando se tenga la ruta candidata, se calcula su coste total para quedarse con una ruta especificada por su ciudad de origen.

Archivo con el código comentado: **parte1_VMC.cpp**

3. Parte2 (Inserción)

    a. Planteamiento

En las estrategias de 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 greedy. Para poder implementar este tipo de estrategia, hay que definir cómo se construye el recorrido parcial , cual es el criterio para insertar el siguiente nodo en el recorrido parcial y donde se inserta el nodo seleccionado.

    b. Construcción del recorrido parcial inicial.

Hemos construido el resultado parcial a partir de las tres ciudades que forman el triángulo más grande posible, seleccionando las ciudades más al norte, más al este y más al oeste. El resultado parcial estará formado por las tres ciudades y la primera ciudad se insertará al final del conjunto para representar que es un ciclo.

    c. Selección del nodo siguiente a insertar en el recorrido parcial.

Para la selección del nodo siguiente a insertar se selecciona aquel con menor recorrido parcial al ser añadido en el conjunto del recorrido parcial.

    d. Lugar de inserción del nodo seleccionado.

El nodo seleccionado se inserta en la posición del recorrido parcial entre los nodos que menor recorrido parcial produzcan.

e. Elementos de la solución del problema.

Conjunto Candidatos	Todas las ciudades
Conjunto Seleccionados	En la primera iteración las tres ciudades más alejadas. Las ciudades no seleccionadas actuales.
Función Solución	Ruta con distancia mínima entre ciudades
Función Selección	La ciudad no visitada con menor distancia actual en el recorrido parcial.
Función Factibilidad	La distancia de la ruta es mínima
Función Objetivo	Elegir un recorrido parcial principal y seleccionar las ciudades minimizando su recorrido parcial al ser añadidas.

f. Pseudocódigo

```
Lectura de ciudades
inicializar el recorrido parcial usando tres ciudades
Eliminar las ciudades del recorrido parcial de las ciudades disponibles

Para cada ciudad disponible
    calculamos el posible recorrido si se insertara esa ciudad
    Almacenar la ciudad con la que el recorrido es el minimo

Actualizar conjunto origen
    Eliminar la ciudad seleccionada

```

Como se puede ver en el pseudocódigo anterior, primero se carga la lista de ciudades con la ruta leída por fichero, acto seguido, se inicializa el recorrido parcial con las tres ciudades comentadas anteriormente y se eliminan del conjunto de ciudades origen.

Entonces, para cada ciudad disponible calculamos el posible recorrido si se inserta esa ciudad y añadimos la ciudad con menor recorrido parcial al ser añadida al conjunto de ciudades de recorrido parcial y la eliminamos del conjunto de ciudades origen.

Archivo con el código comentado: **parte2_Insercion.cpp**

4. Parte 3 (Mejora VMC 2-opt)

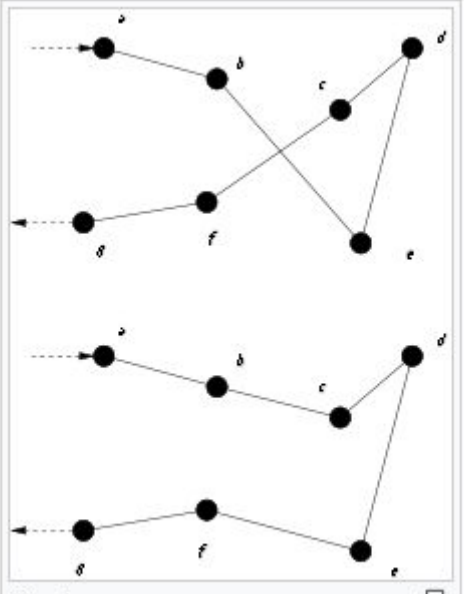
    a. Planteamiento

En este apartado se nos pide buscar una solución alternativa para este problema, que no fuera el VMC ni Inserción, se nos ocurrió que se podría probar yendo de dos en dos ciudades (o de n en n) y a partir de ahí calcular una nueva ruta, pero esto al fin y al cabo no deja de ser VCM ya que hay que relacionar 'n' ciudades entre sí. También se nos ocurrió probar con algún tipo de condición, como el vals (baile), yendo a la ciudad más próxima la derecha, otra vez la más próxima a la derecha y luego a la izquierda, hasta completar la ruta, pero pensándolo decidimos no hacerlo, ya que lo consideramos un algoritmo muy aleatorio y que dependiendo de las ciudades puede salir muy bien, o puede salir muy mal, siendo la probabilidad de que saliera mal más elevada que de que saliera bien.

De esta forma, nos pusimos a investigar algoritmos greedy de mejora y encontramos los algoritmos k-opt, los cuales se basan en intercambiar los vértices entre dos o más ciudades, de forma que si se tuviera un cruce, este cruce se solucionaría.

(Ver ***k*-opt heuristic, or Lin–Kernighan heuristics)** en:
https://en.wikipedia.org/wiki/Travelling_salesman_problem

Para esta implementación se ha elegido el 2-opt. (https://en.wikipedia.org/wiki/2-opt)



2-opt

b. Elementos de la solución del problema

Conjunto Candidatos	Todas las ciudades
Conjunto Seleccionados	Las ciudades no visitadas
Función Solución	Ruta sin cruces entre 4 ciudades consecutivas
Función Factibilidad	La ruta mejora el coste total
Función Selección	Las ciudades con un cruce
Función Objetivo	Obtener una ruta óptima sin cruces entre 4 ciudades consecutivas

c. Pseudocódigo

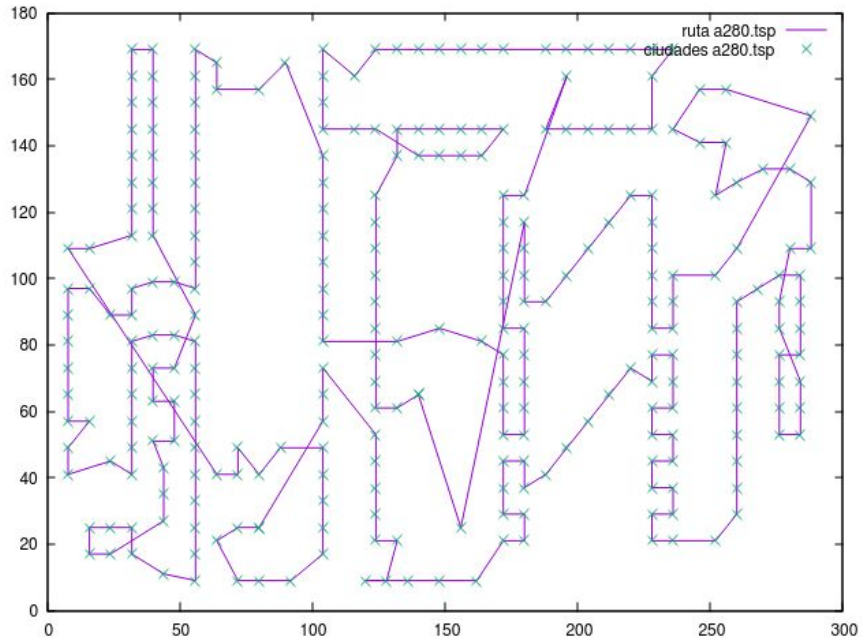
```
Calcular ruta VMC
    ...  (igual que parte 1 VMC)

Mientras no hay mejora:
    Calcular coste ruta actual (ruta actual inicial es la optima calculada en VMC)
    Para cada ciudad i:
        Para cada ciudad j = i+1:
            intercambia ciudades desde i a j:
                Se añade a ruta objetivo desde ciudad 0
                    hasta ciudad i-1 en orden de la ruta actual
                Se añade a ruta objetivo desde ciudad j
                    hasta ciudad i en orden inverso de la ruta actual
                Se añade a ruta objetivo desde ciudad j+1
                    hasta ciudad final en orden de la ruta actual
            Calcular coste ruta objetivo
            Redondear coste
            Si el coste mejora:
                Se asigna ruta objetivo como actual
```

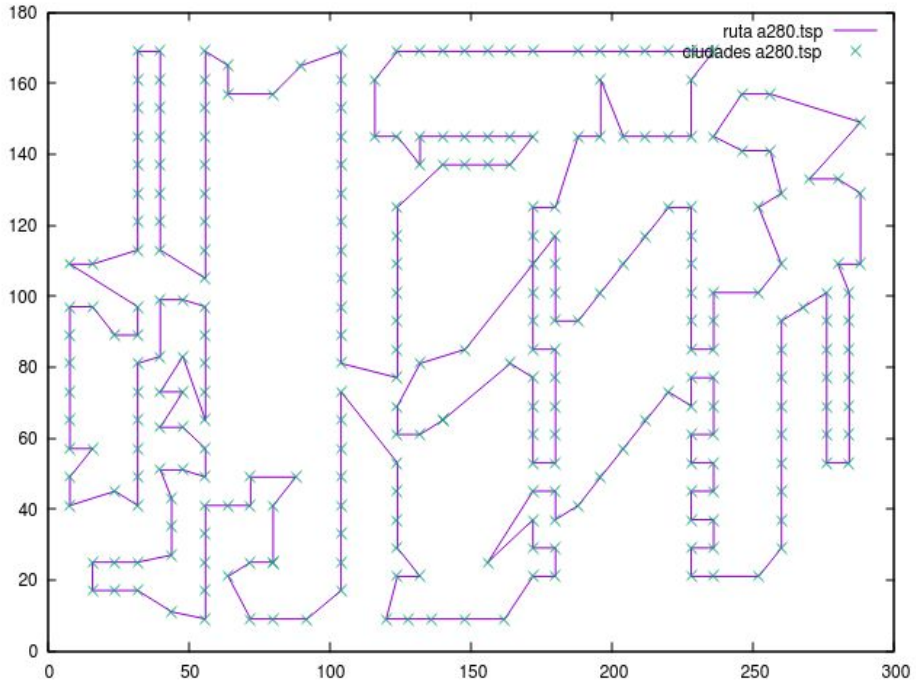
Archivo con el código comentado: **parte3_2-opt.cpp**

5. Comparativa de gráficas entre algoritmos

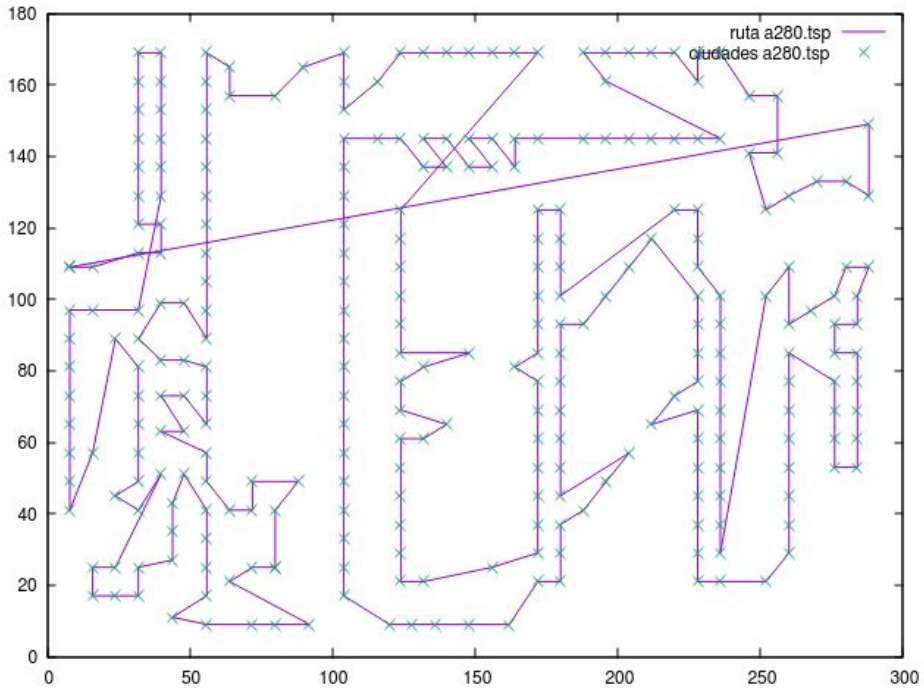
    1. a280



a280_vmc

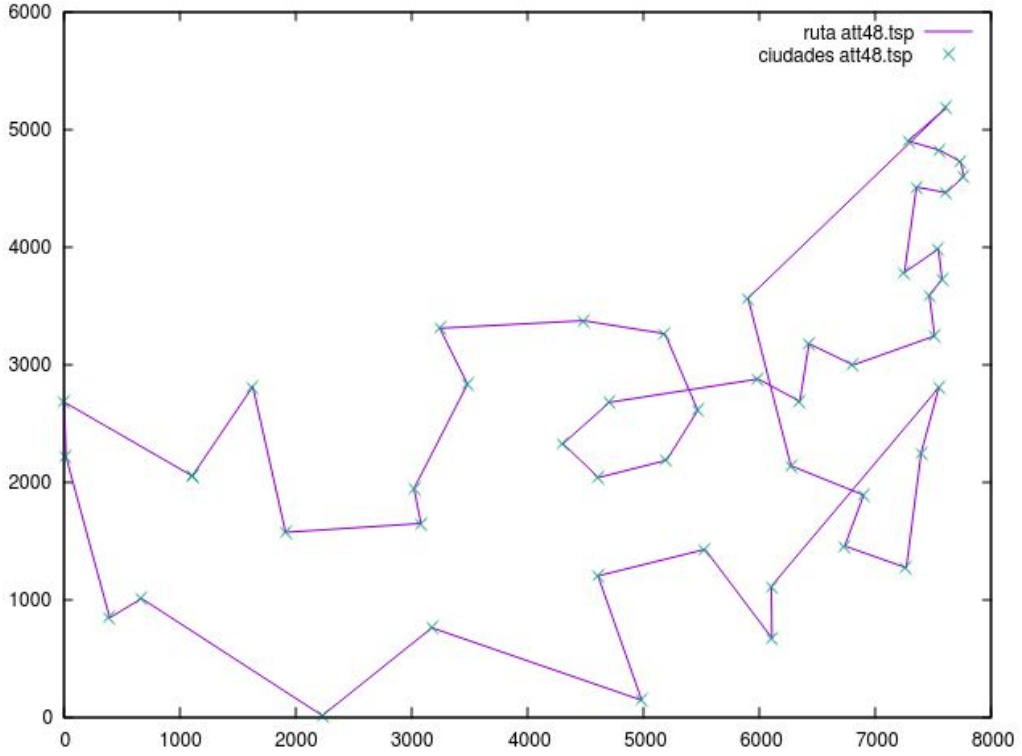


a280_2-opt

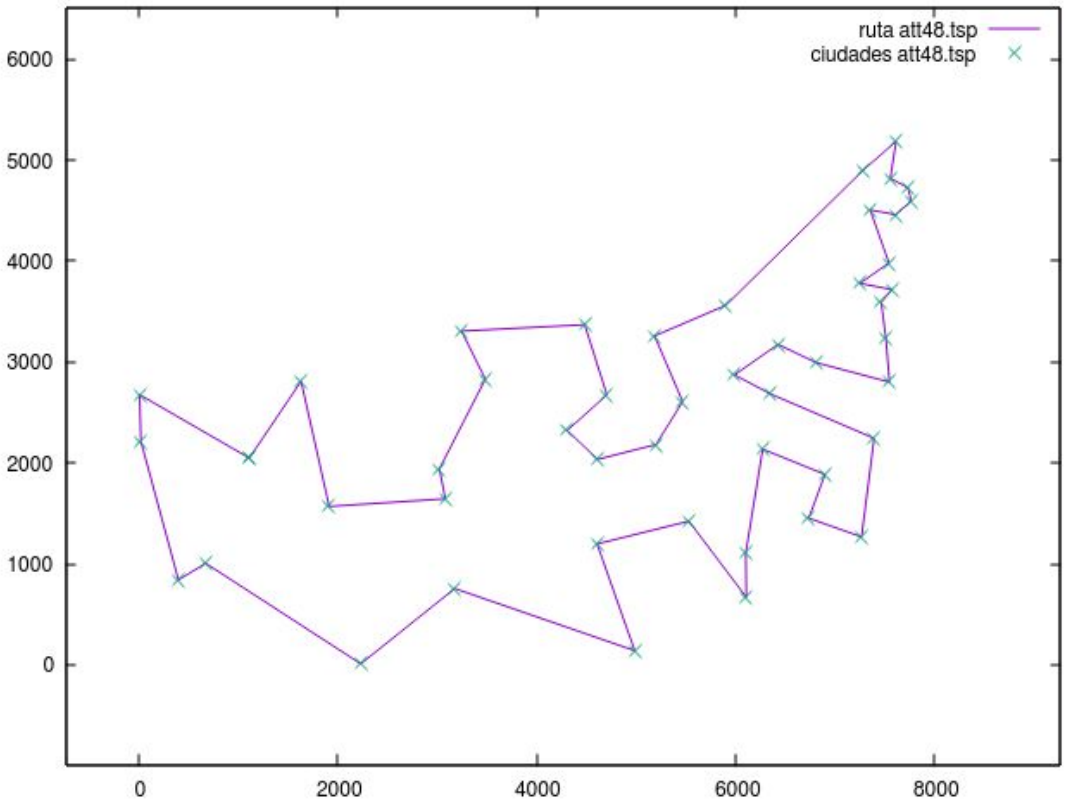


a280_insercion

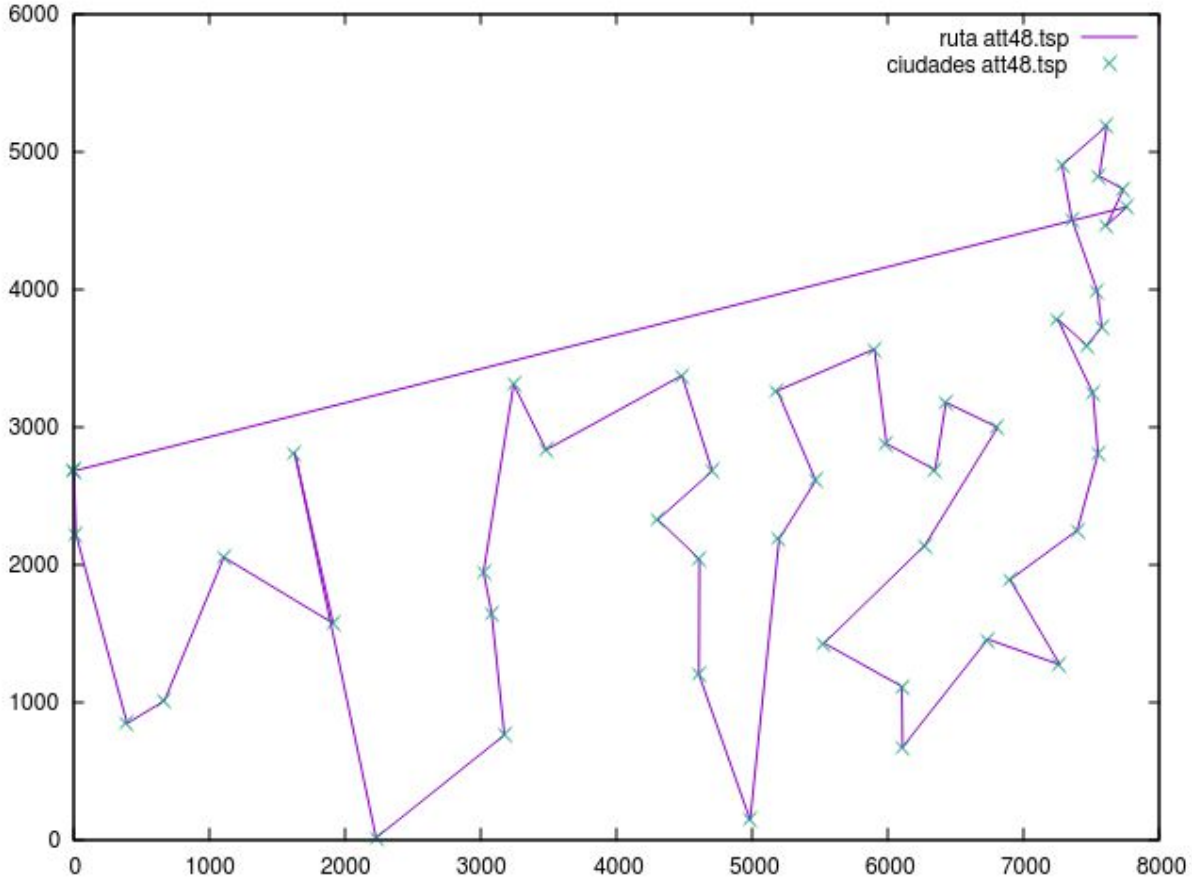
2. att48



att48_vmc

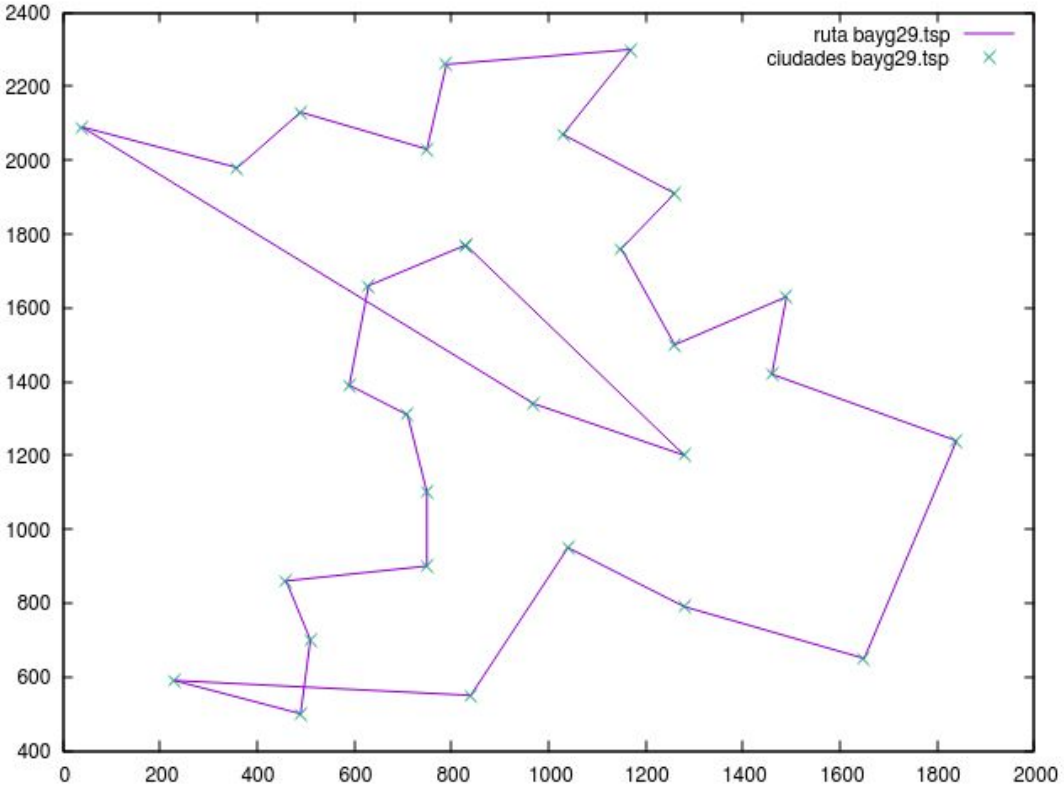


att48_2-opt

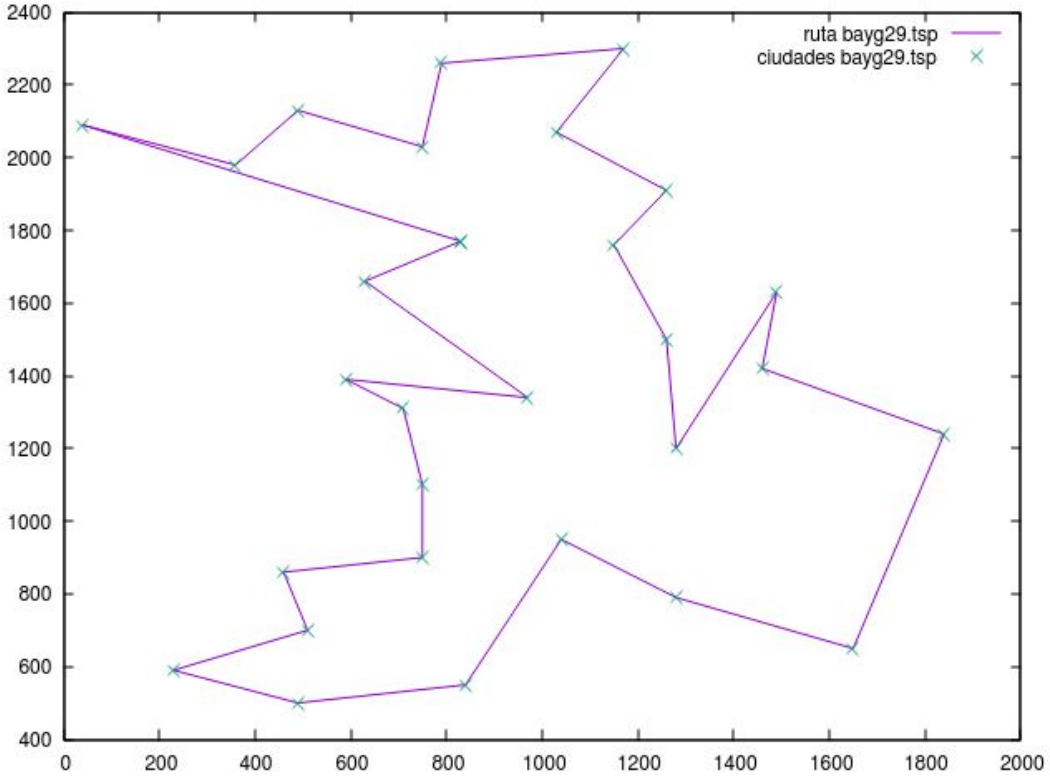


att48_insercion

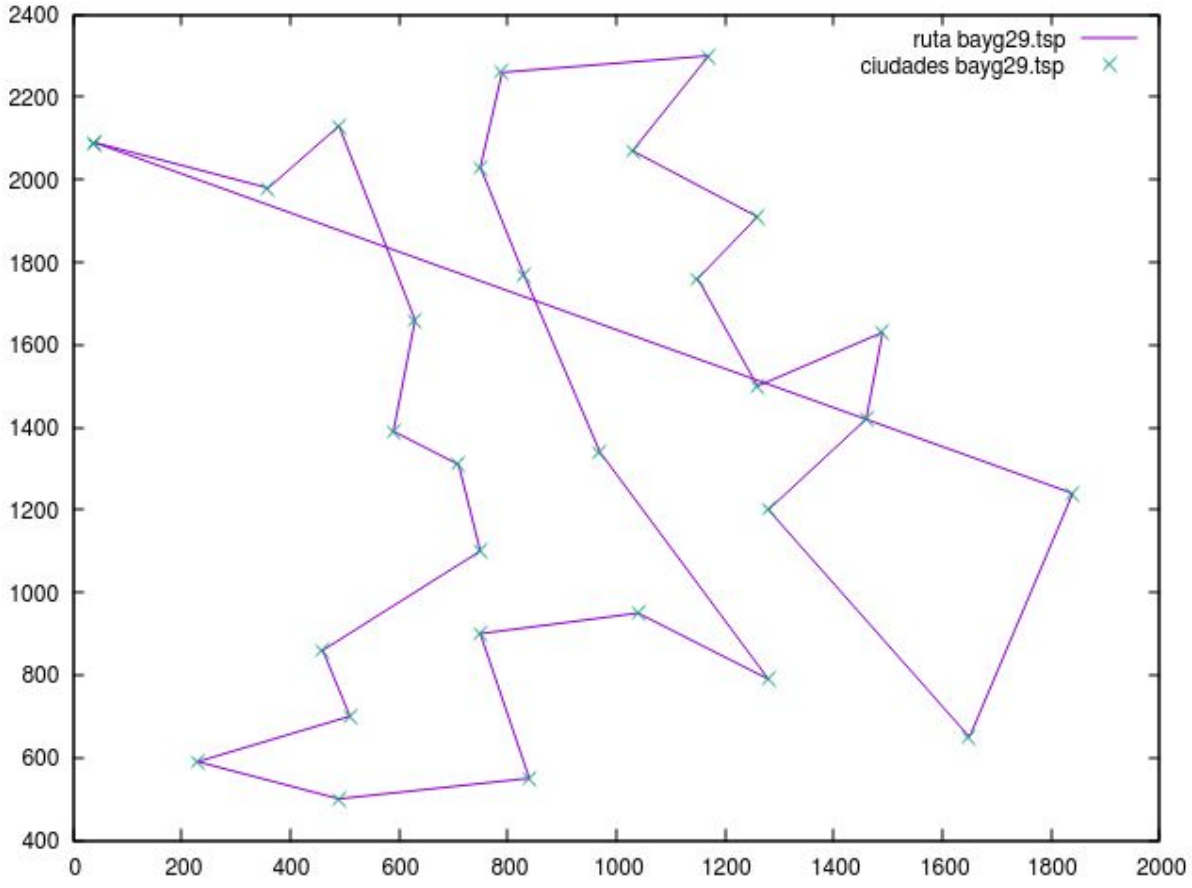
3. bayg29



bayg29_vmc

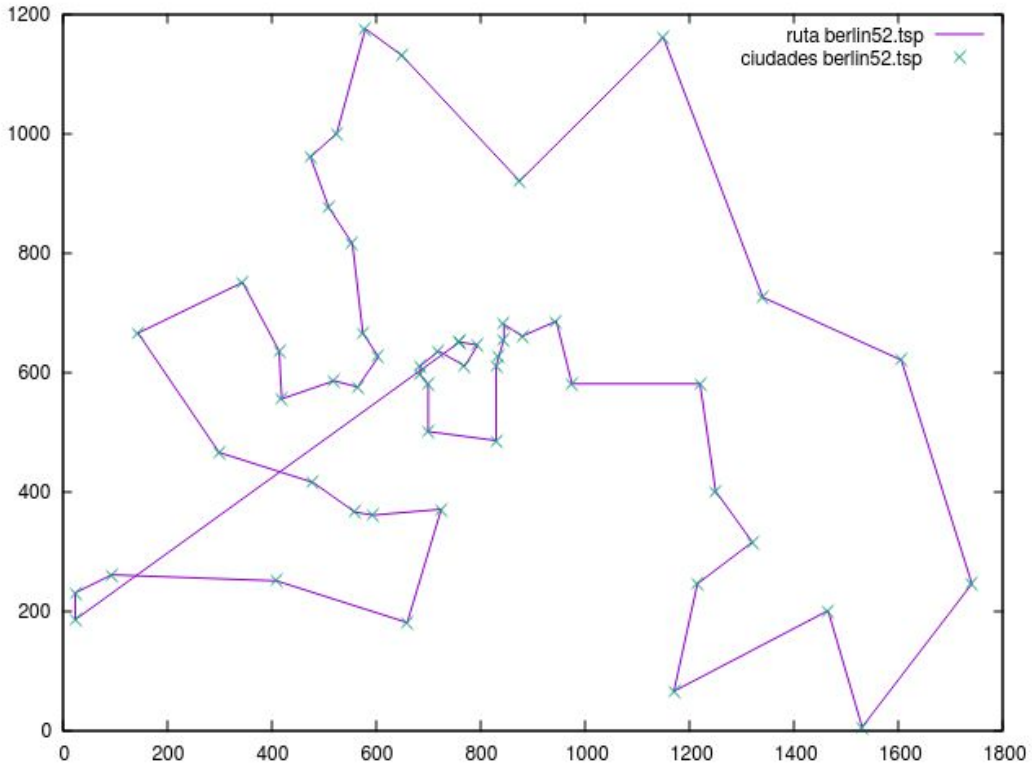


bayg29_2-opt

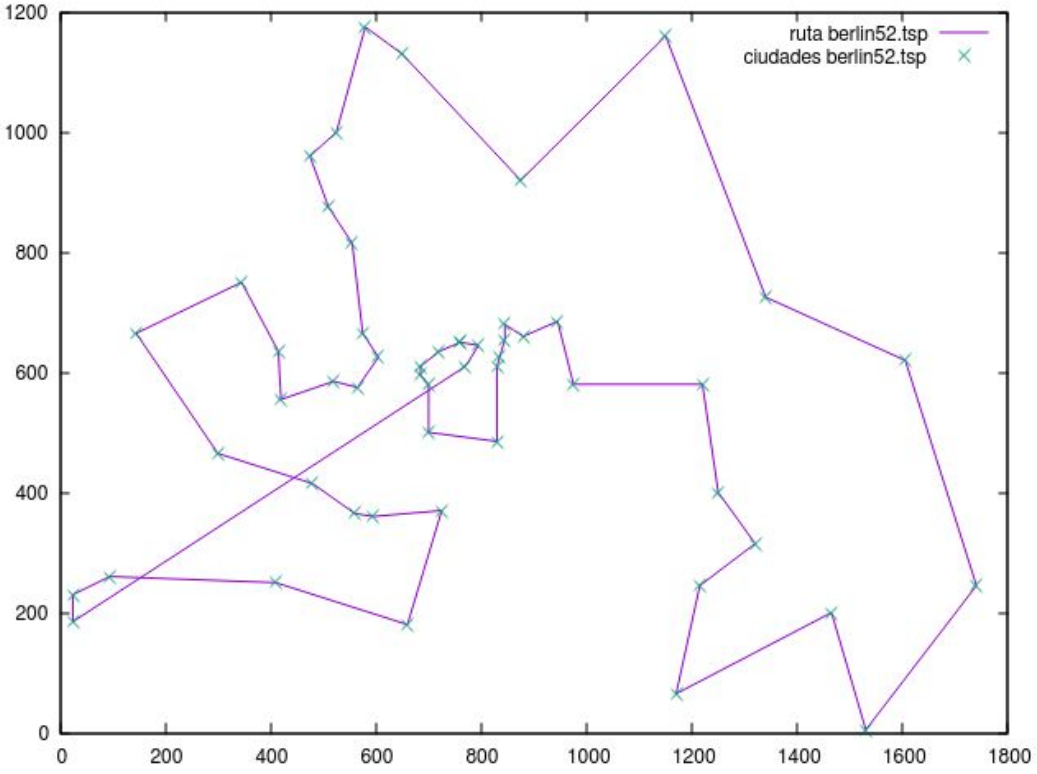


bayg29_insercion

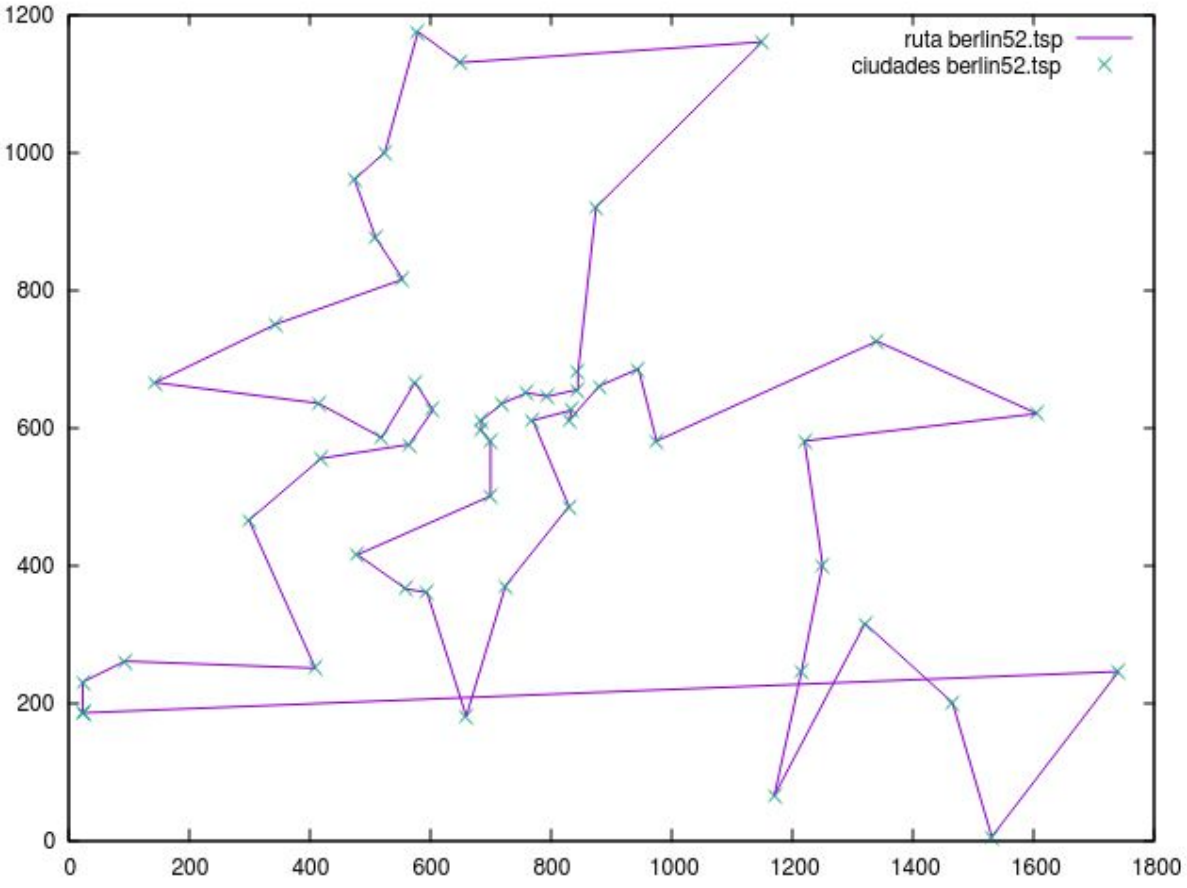
4. berlin52



berlin52_vmc

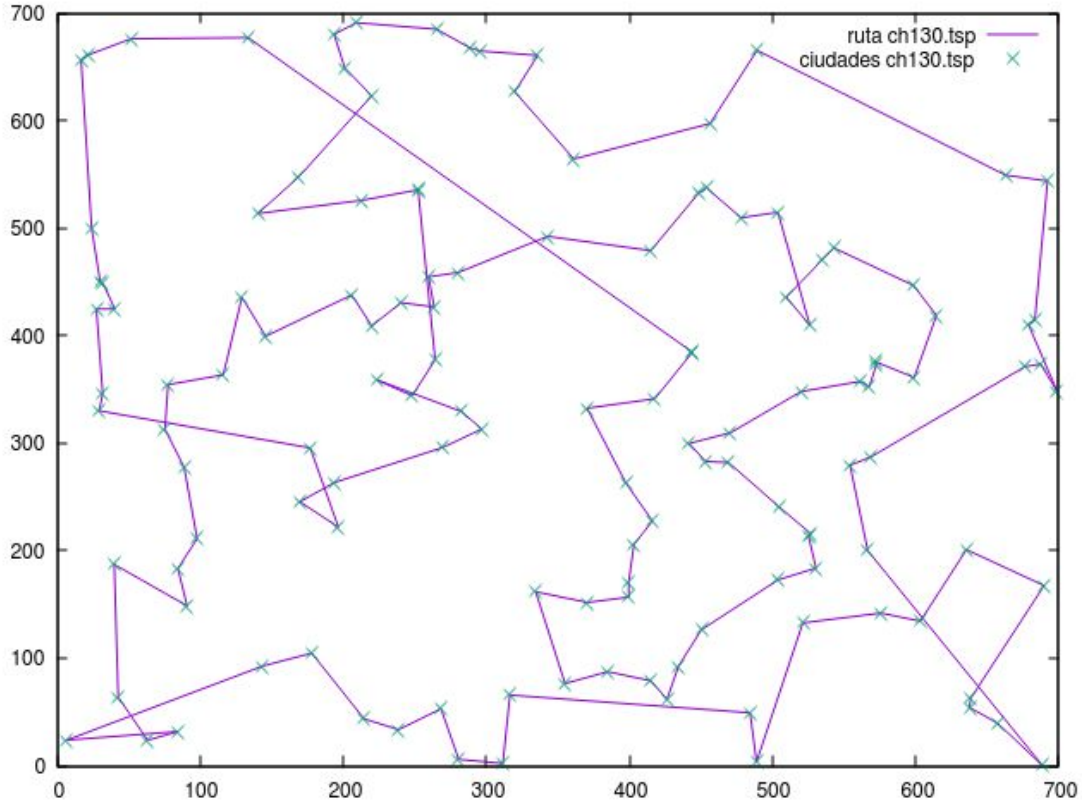


berlin52_2-opt

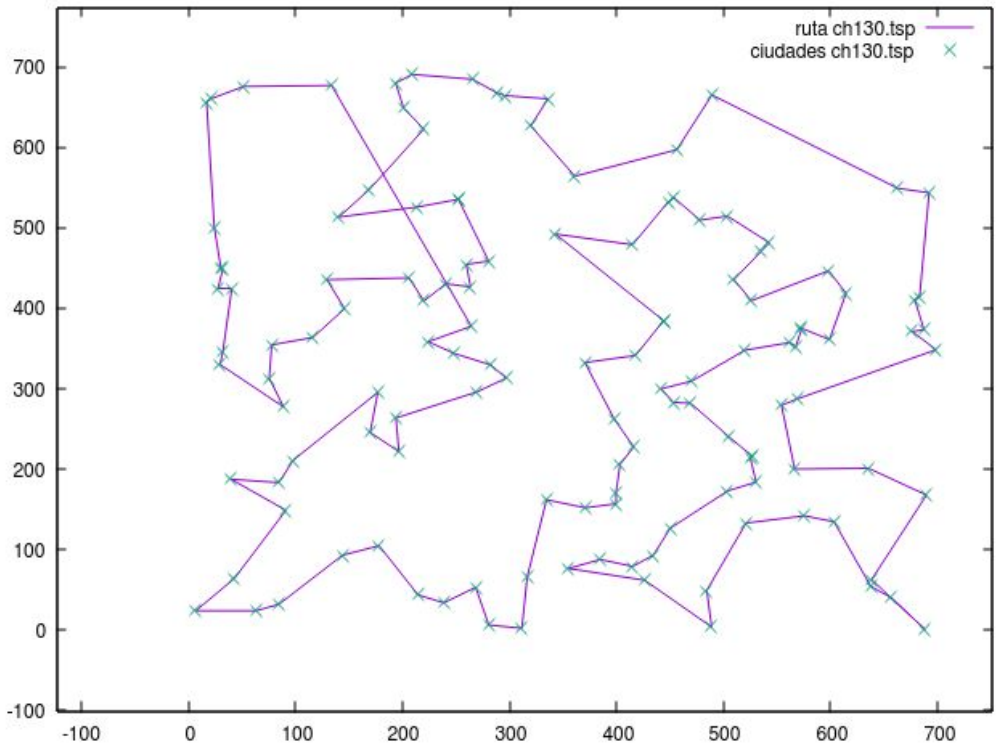


berlin52_insercion

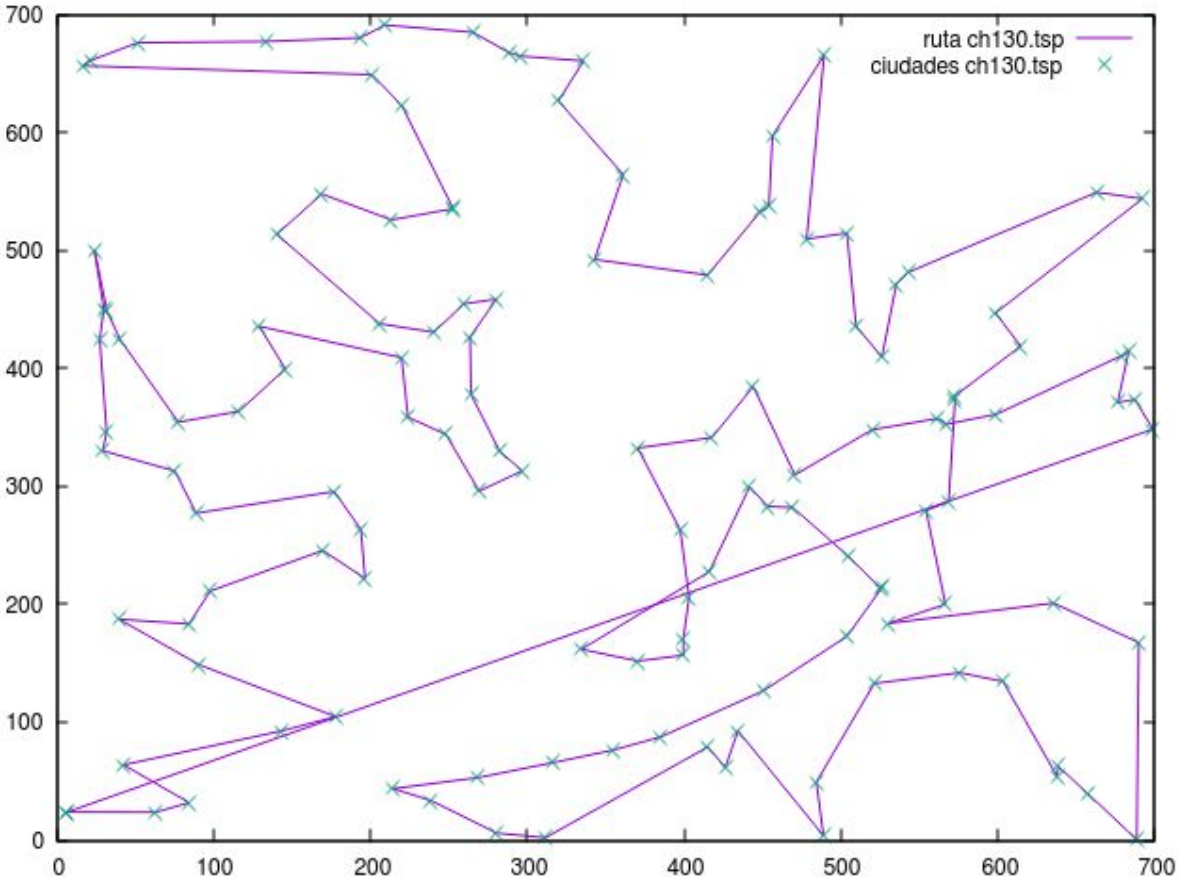
5. ch130



ch130_vmc

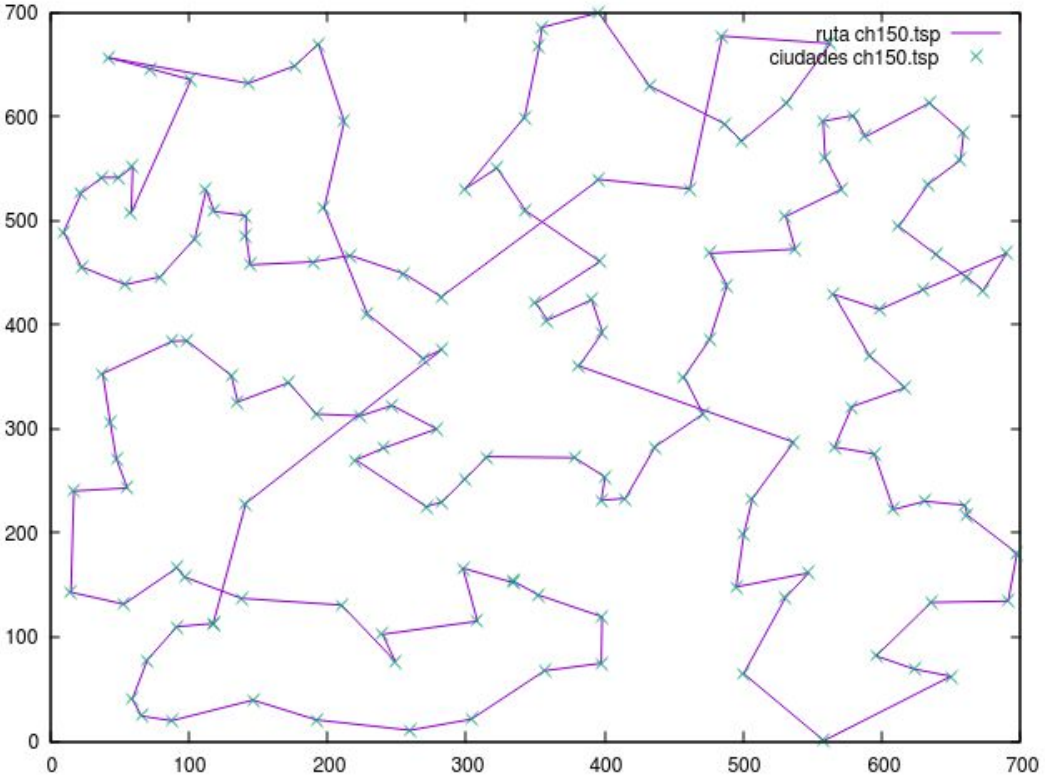


ch130-2_opt

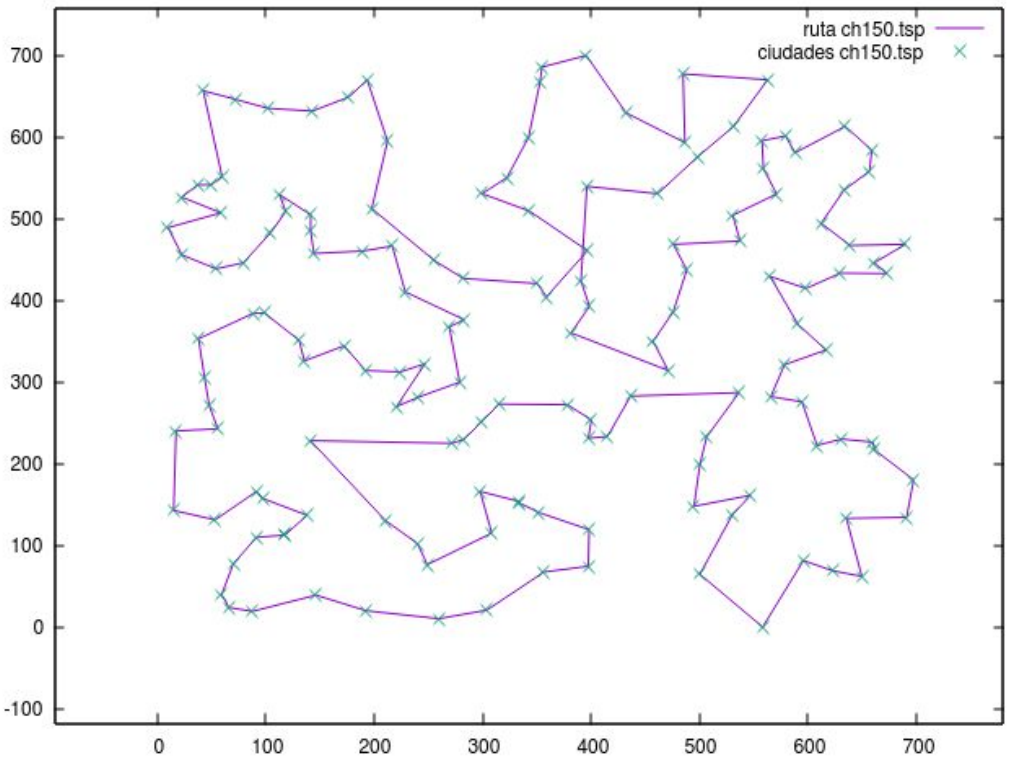


ch130_insercion

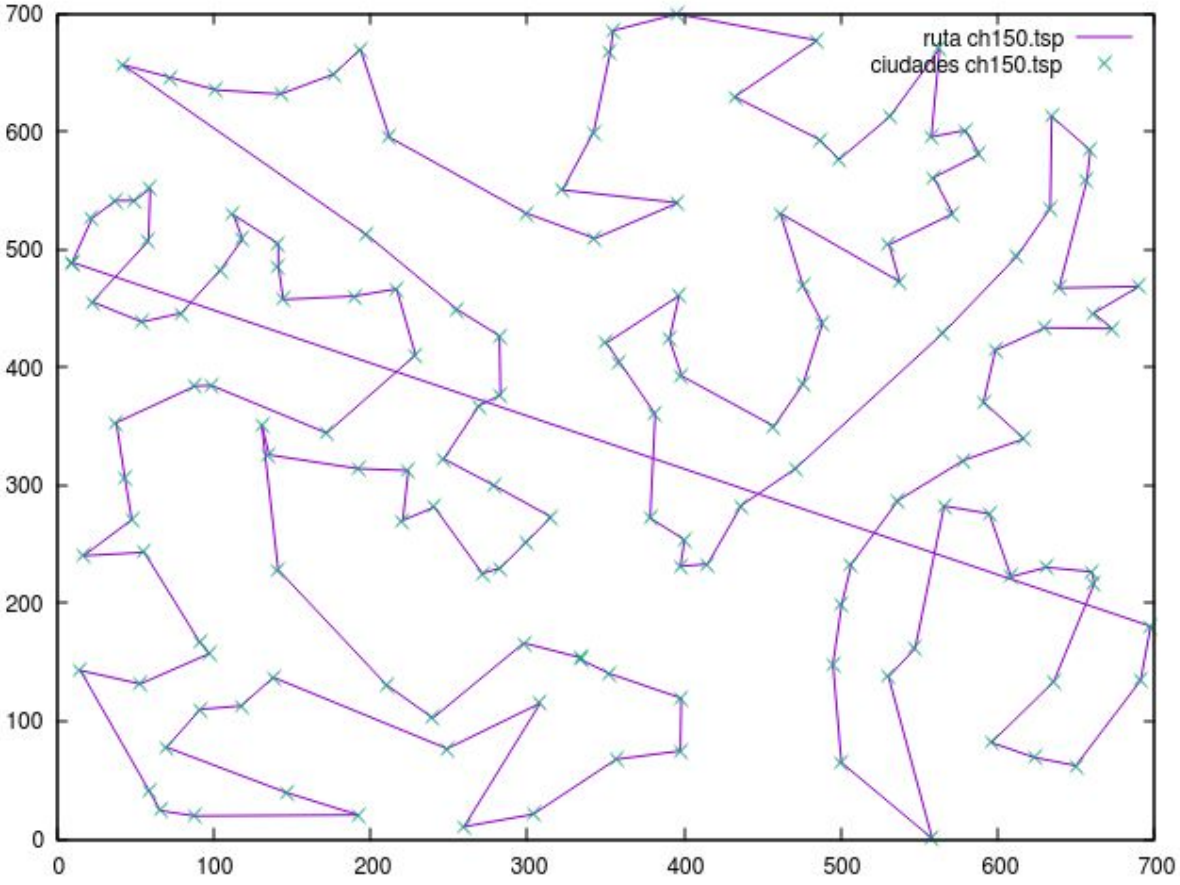
6. ch150



ch150_vmc

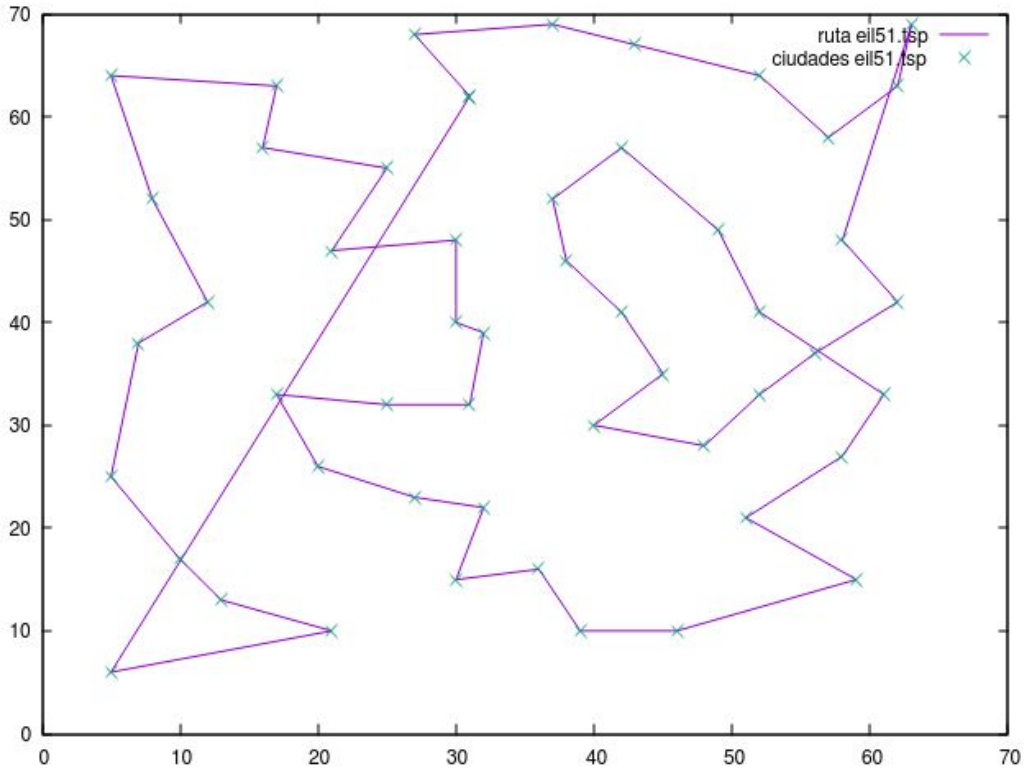


ch150-2_opt

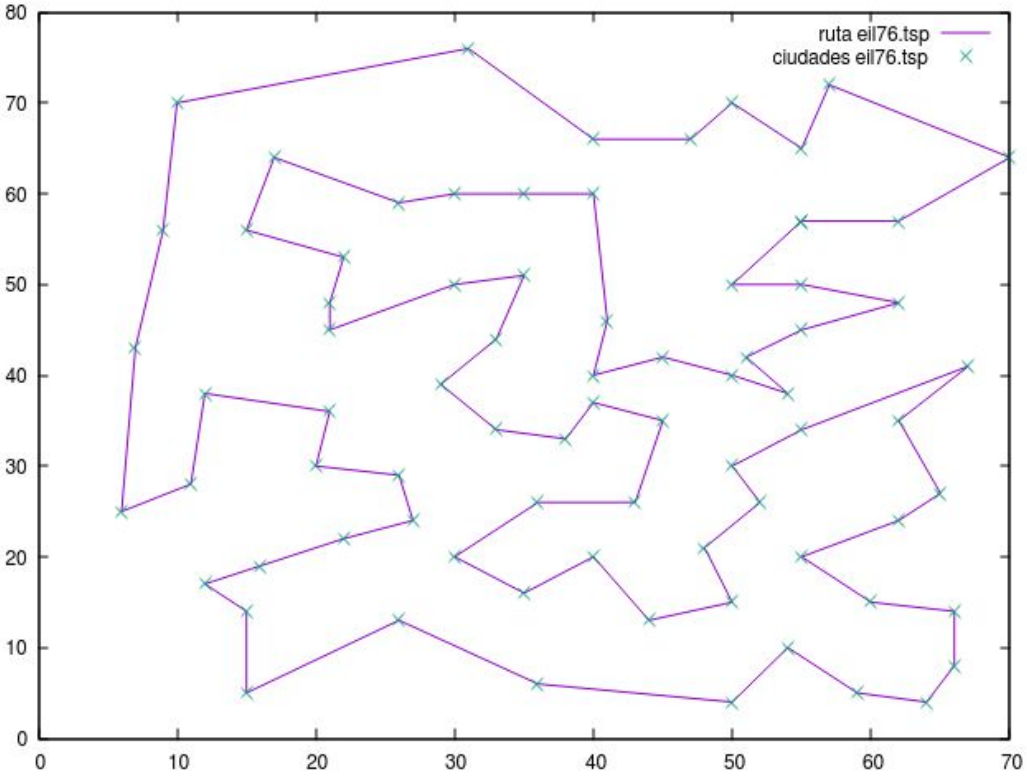


ch150_insercion

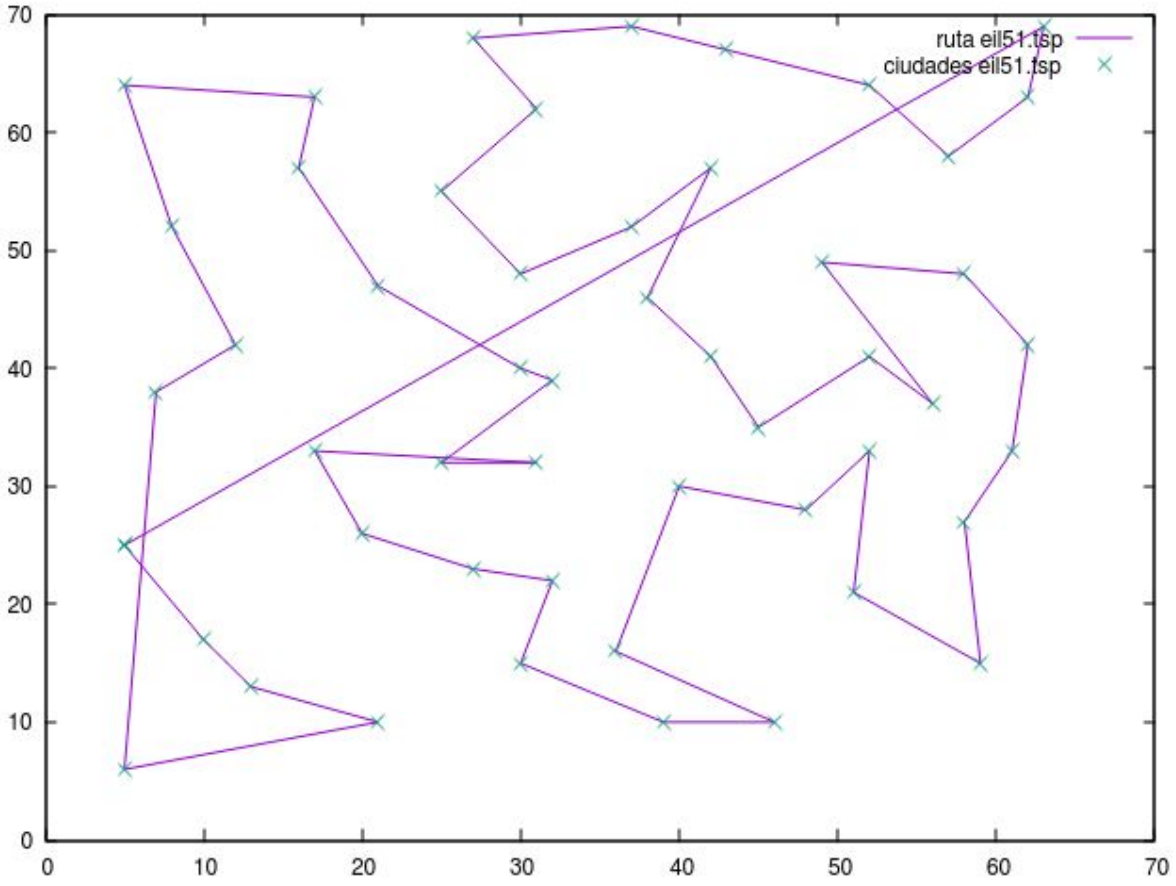
7. eil51



eil51_vmc

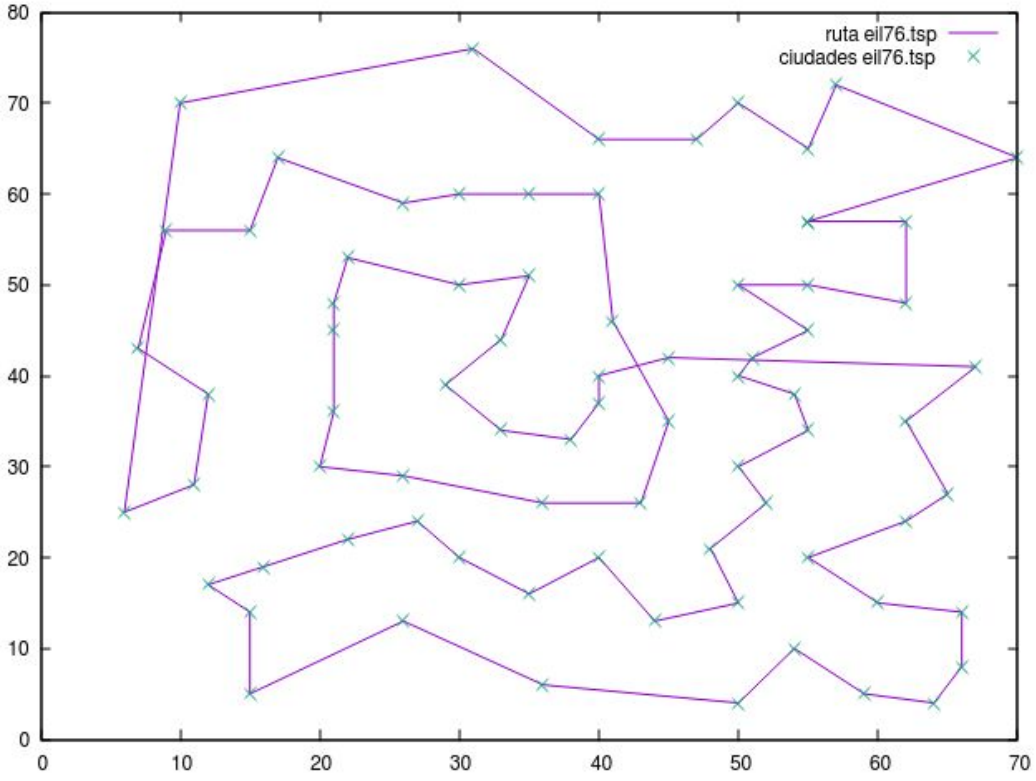


eil51_opt

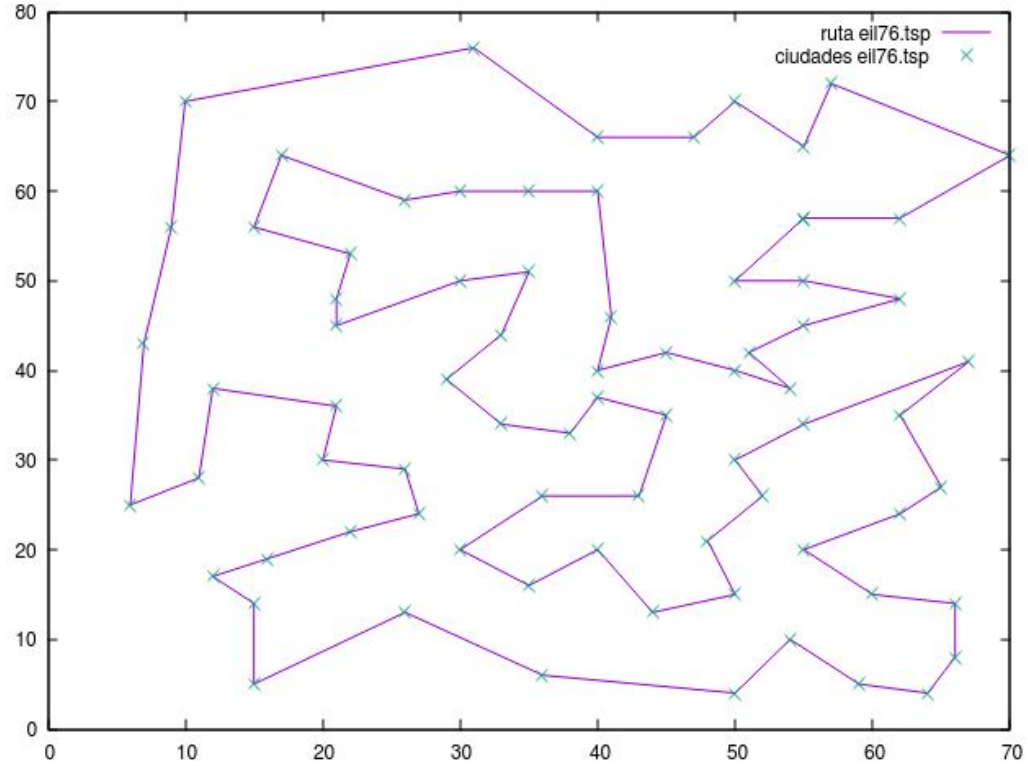


eil51_insercion

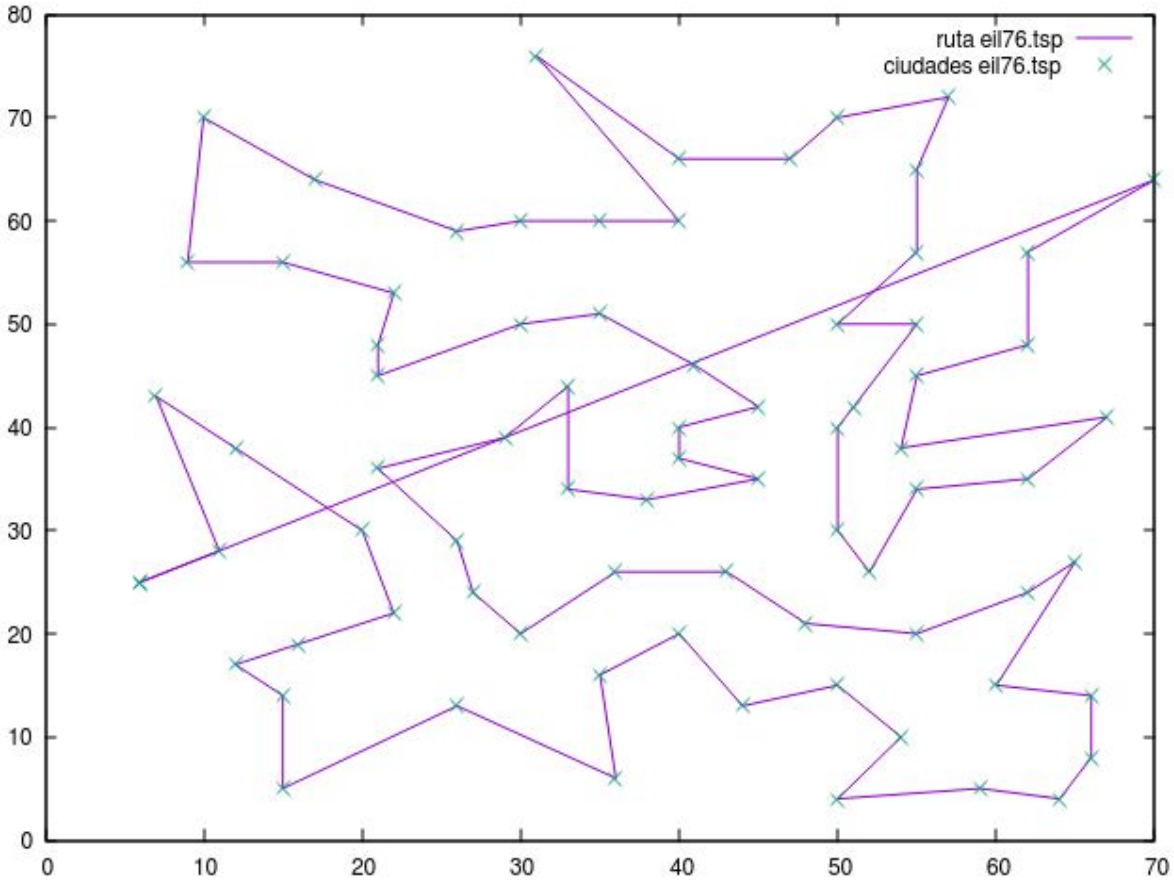
8. eil76



eil76_vmc

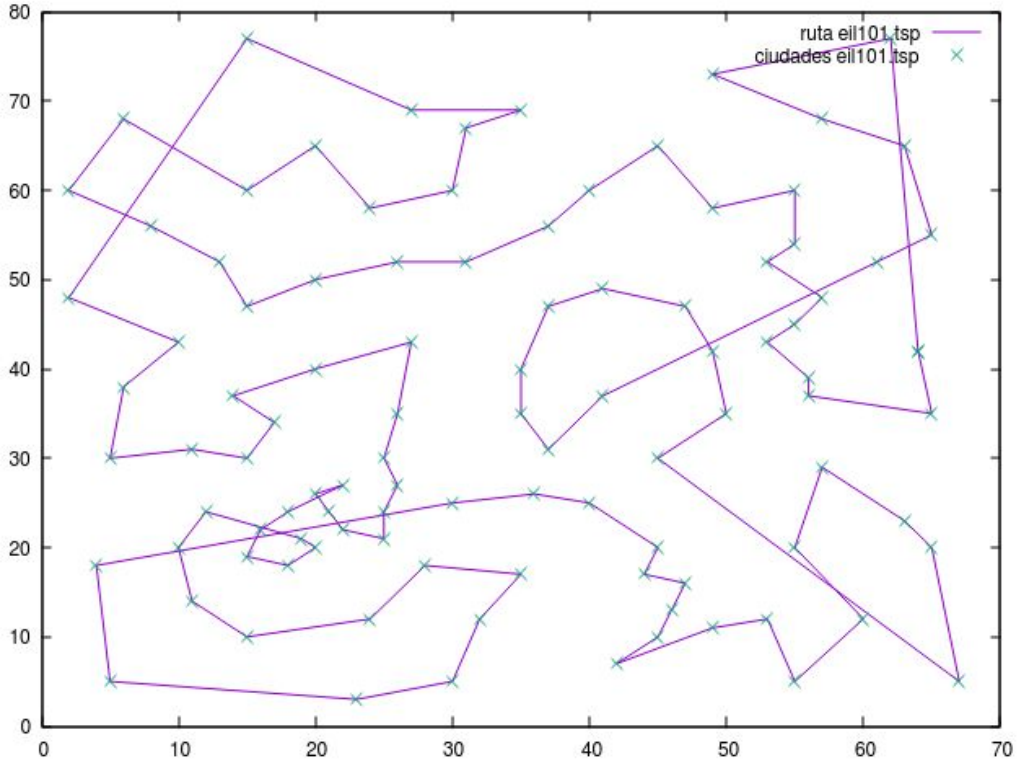


eil76_2-opt

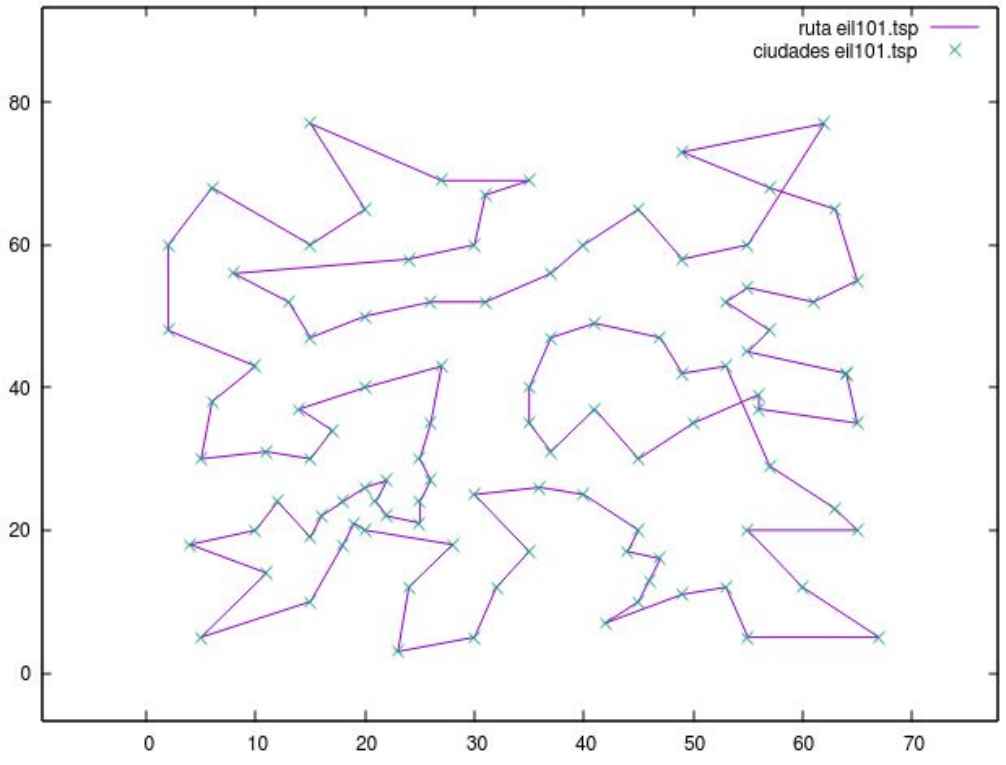


eil76_insercion

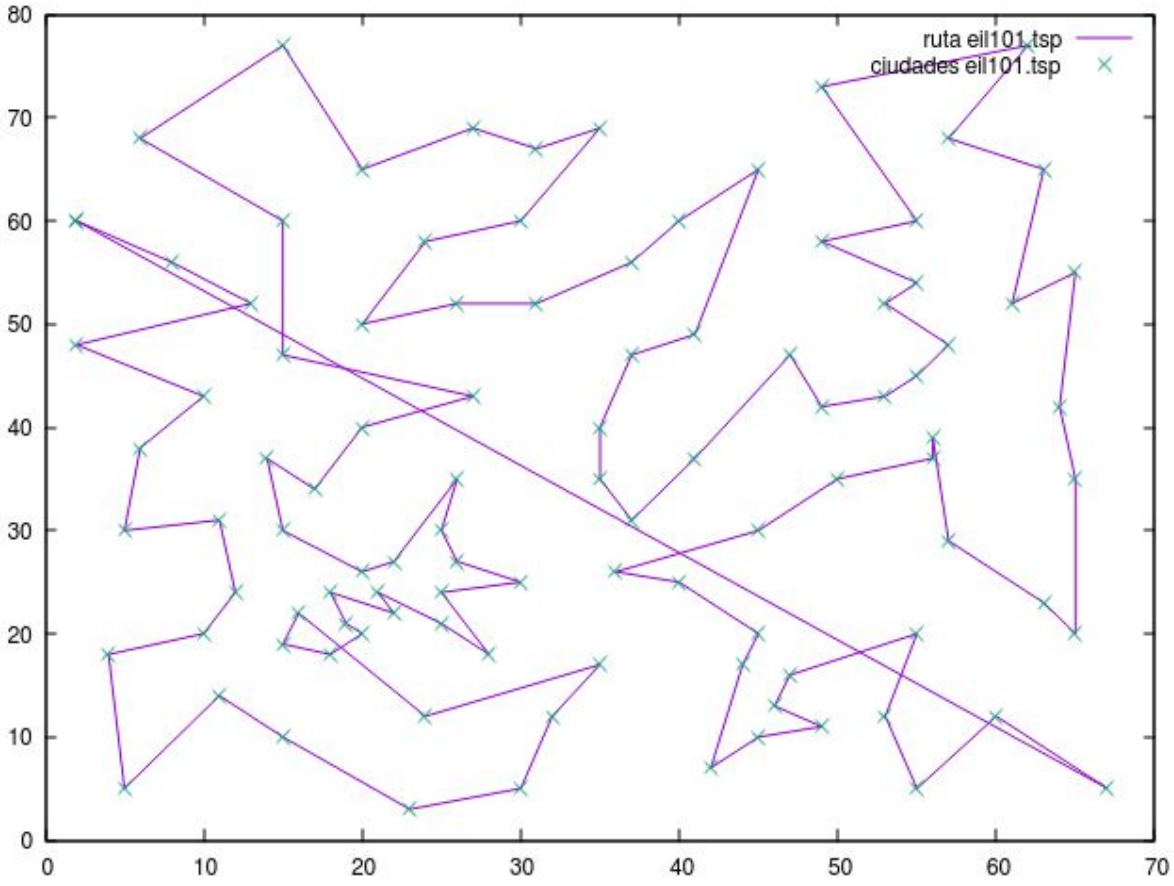
9. eil101



ei101_vmc

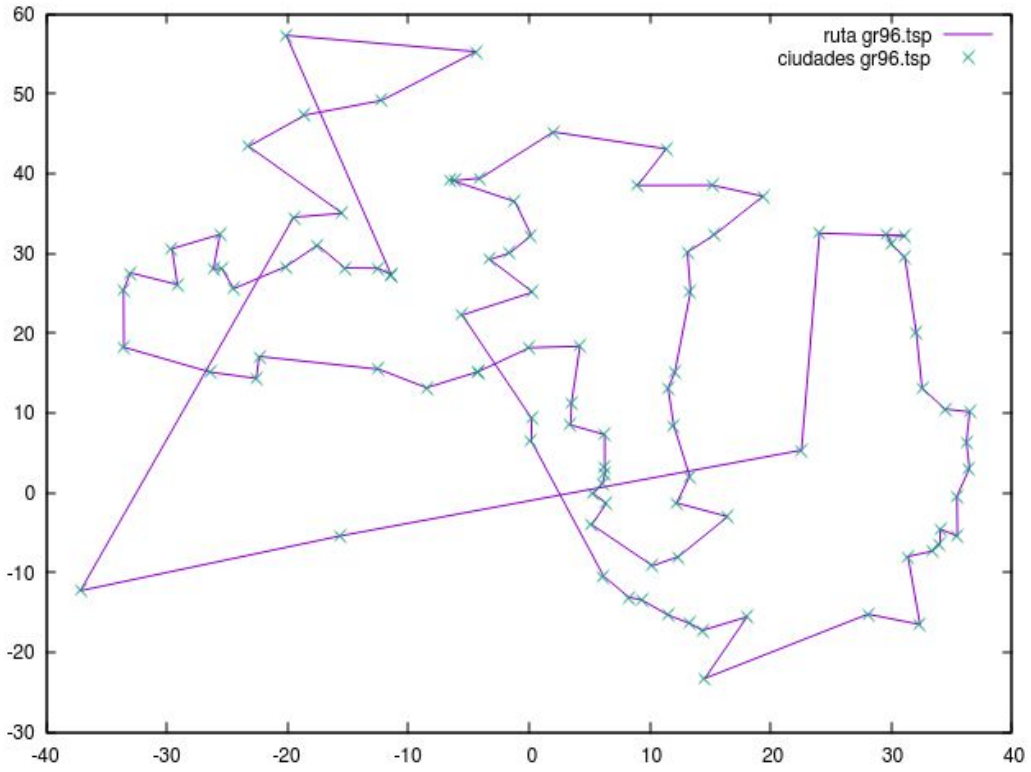


eil101_2-opt

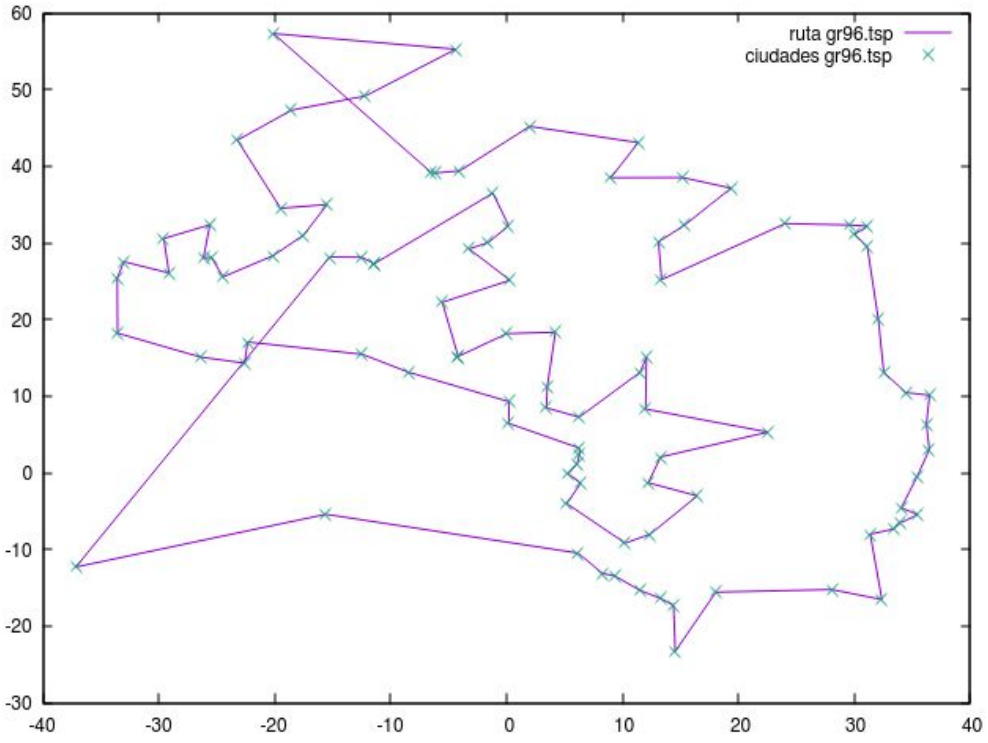


eil101_insercion

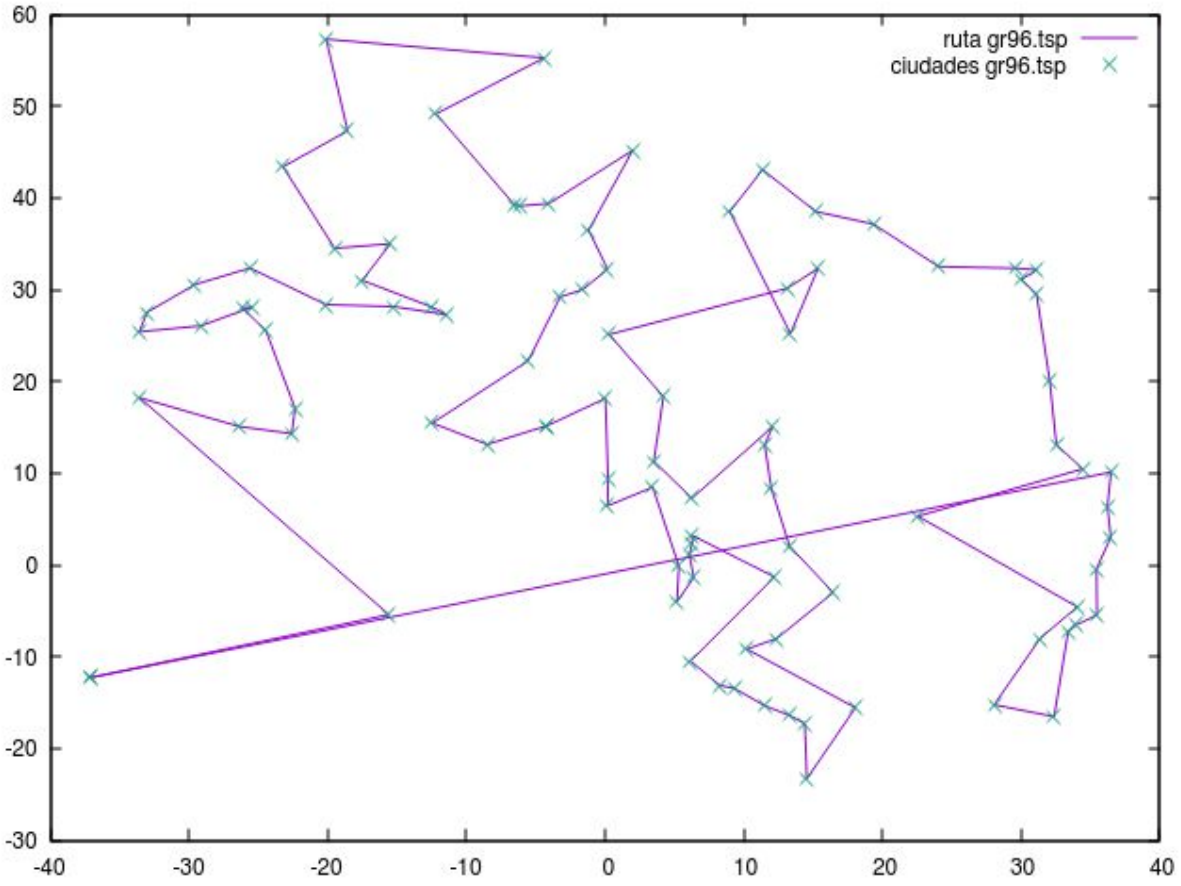
10. gr96



gr96_vmc

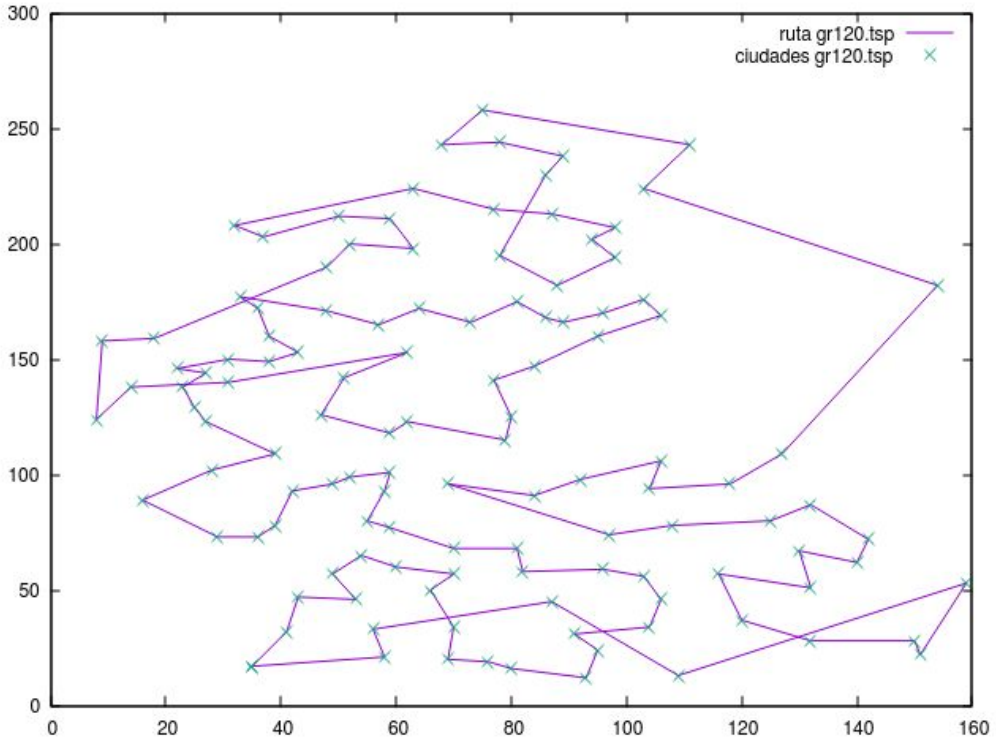


gr96_2-opt

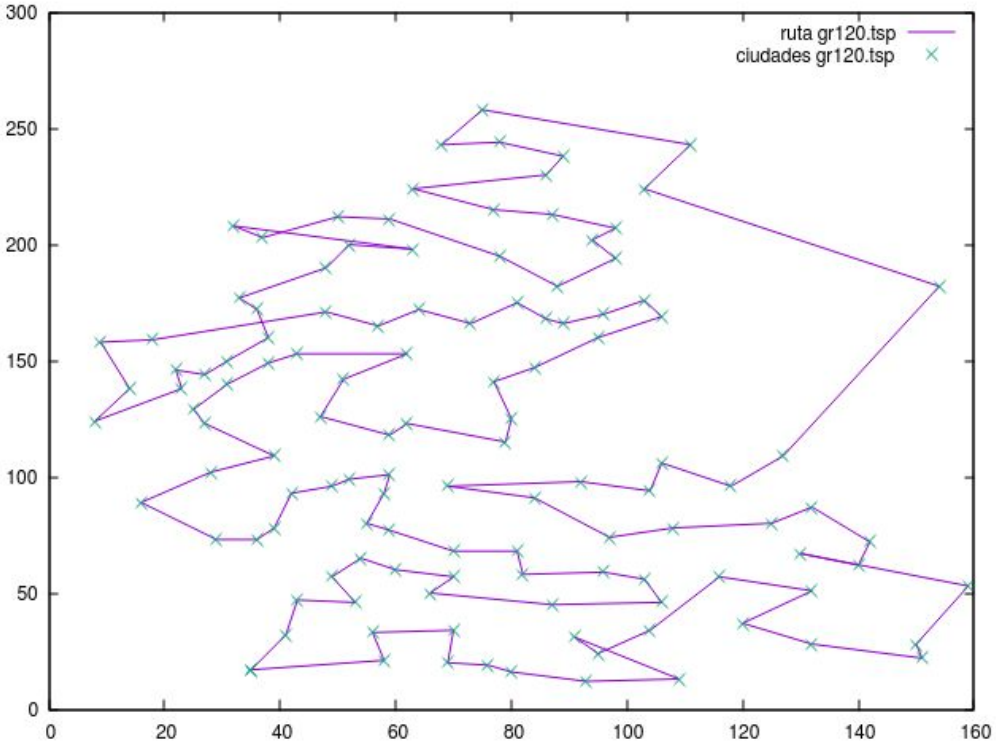


gr96_insercion

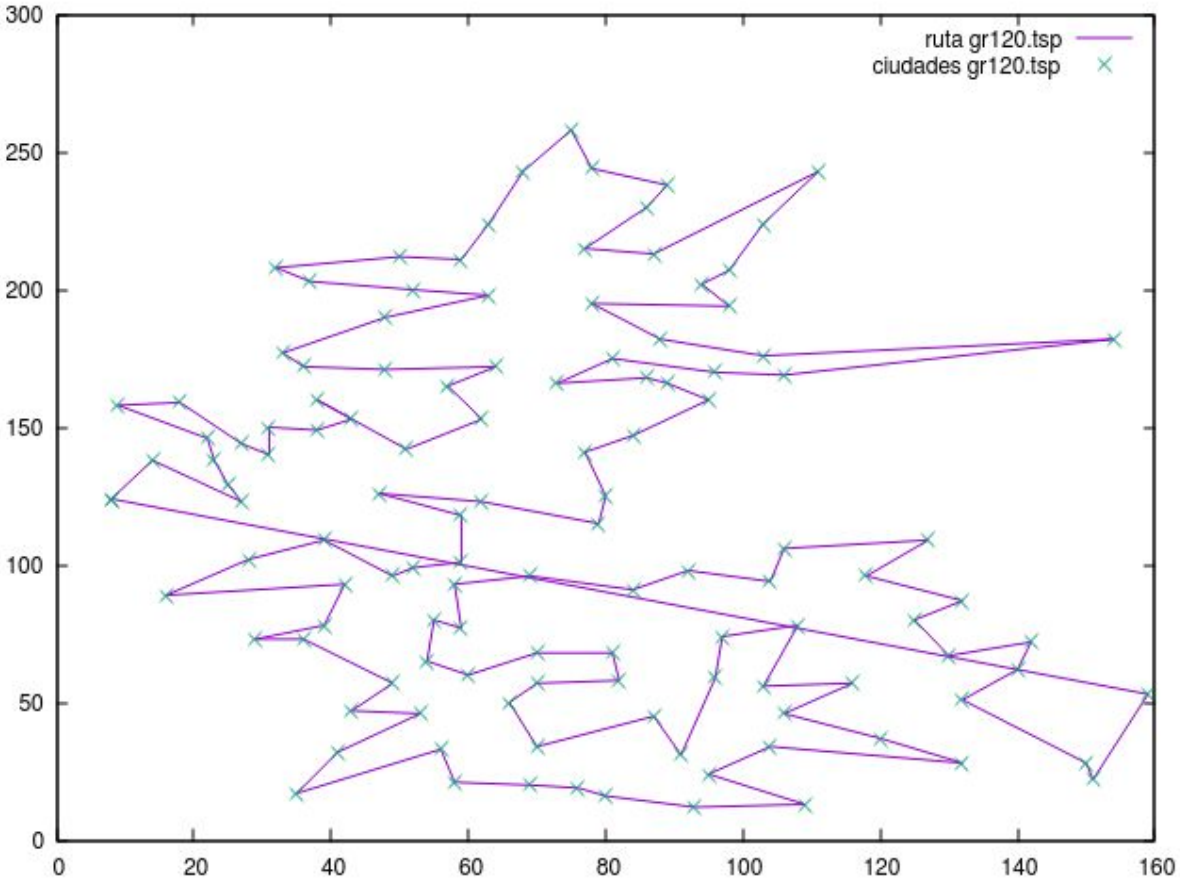
11. gr120



gr120_vmc

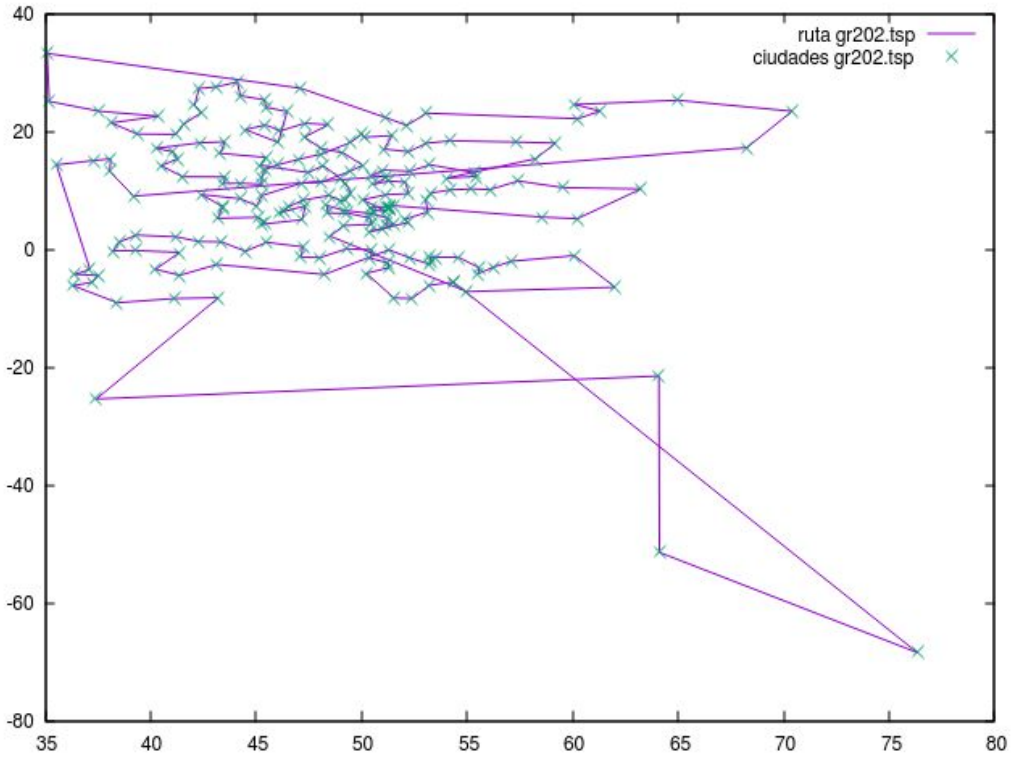


gr120_2-opt

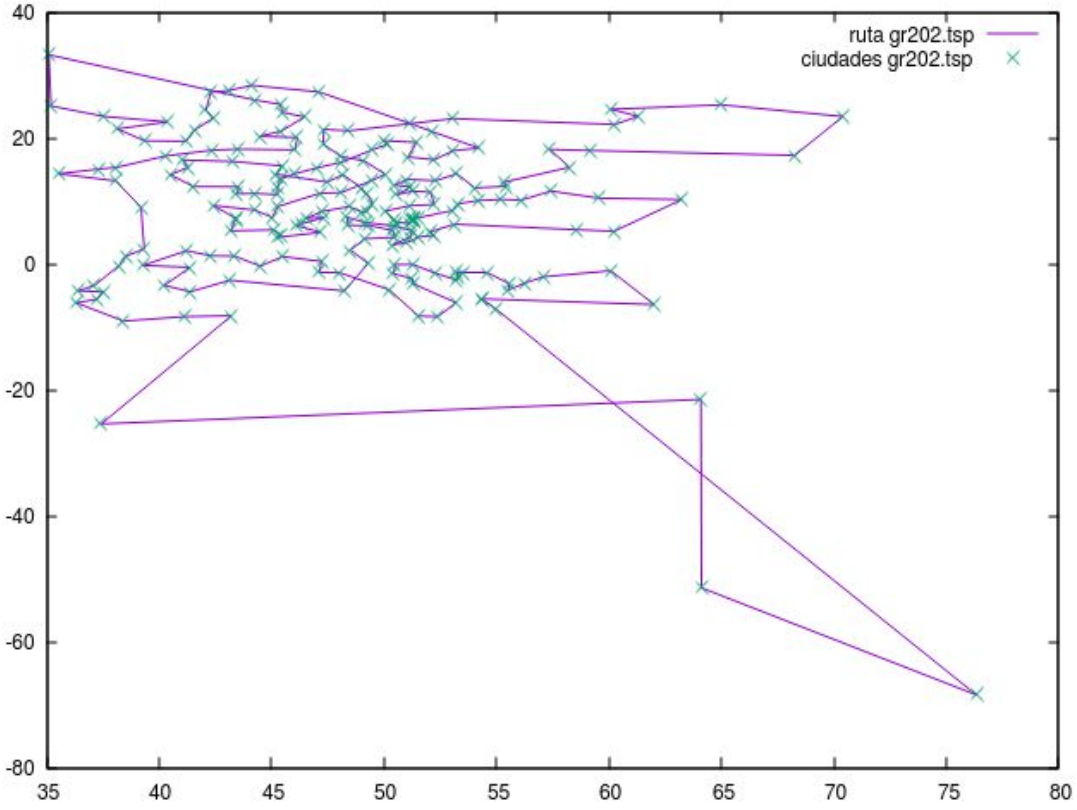


gr120_insercion

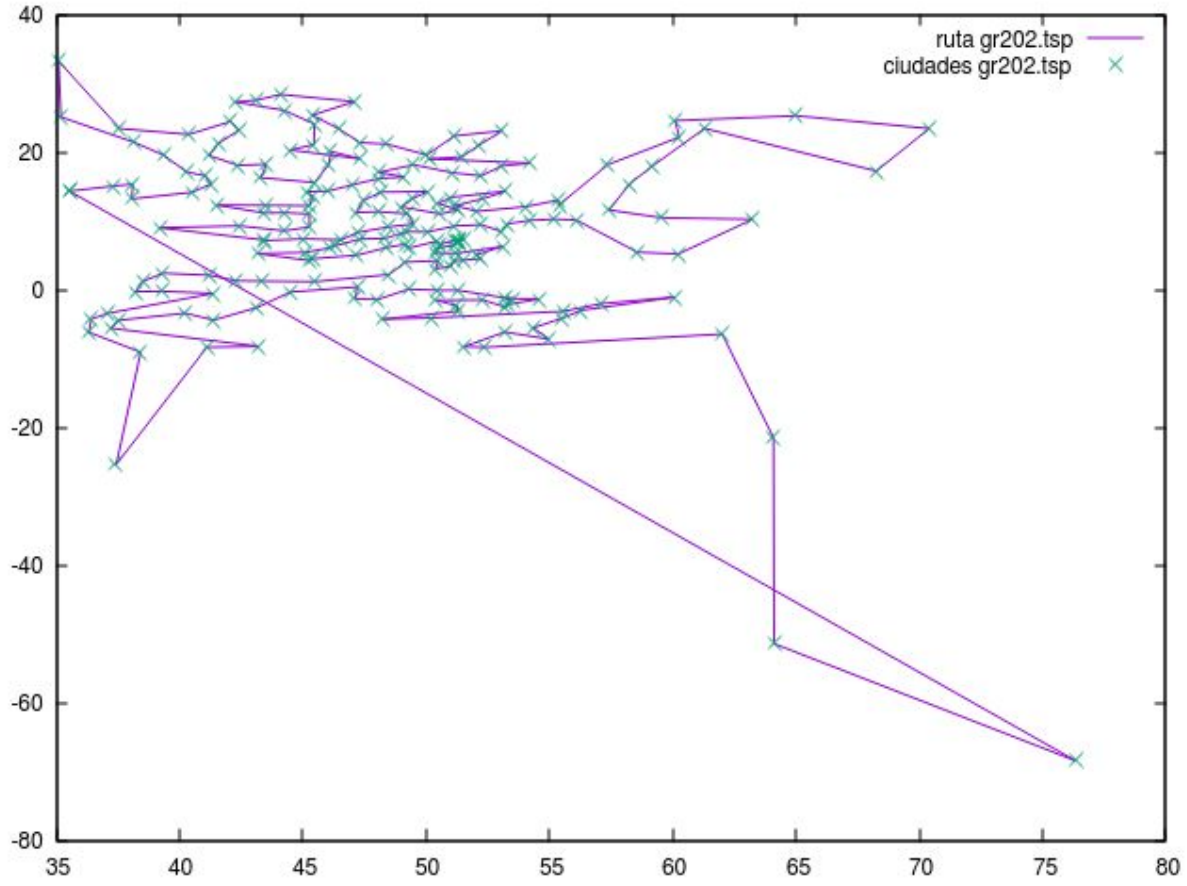
12. gr202



gr202_vmc

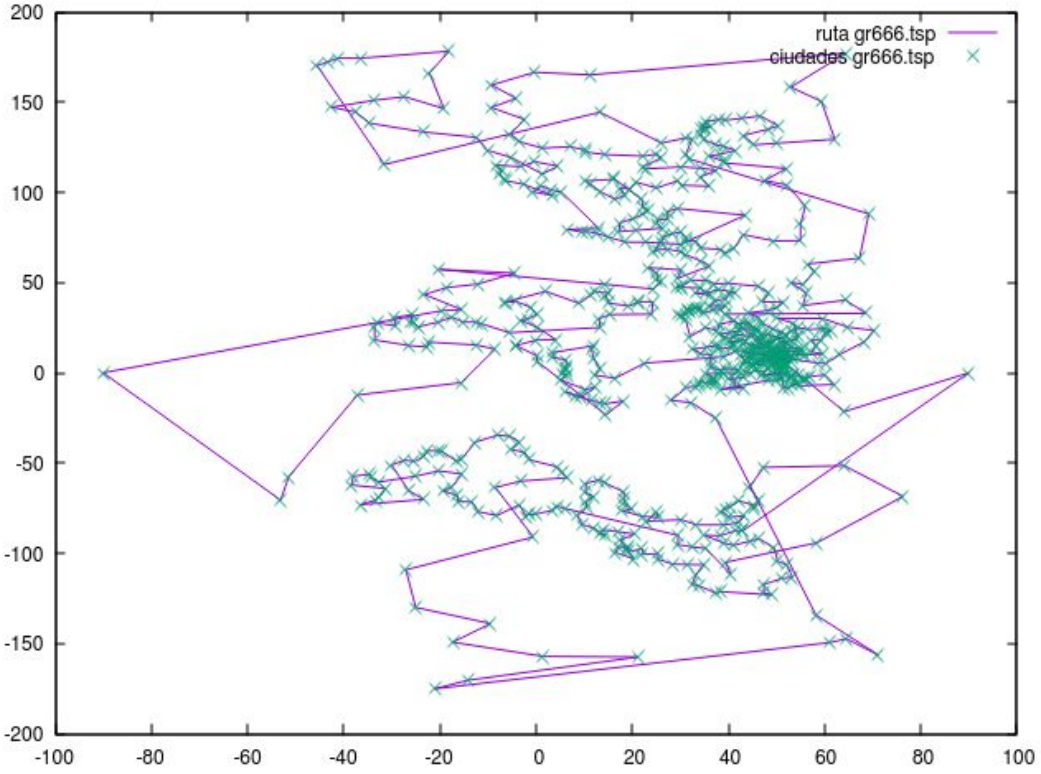


gr202_2-opt

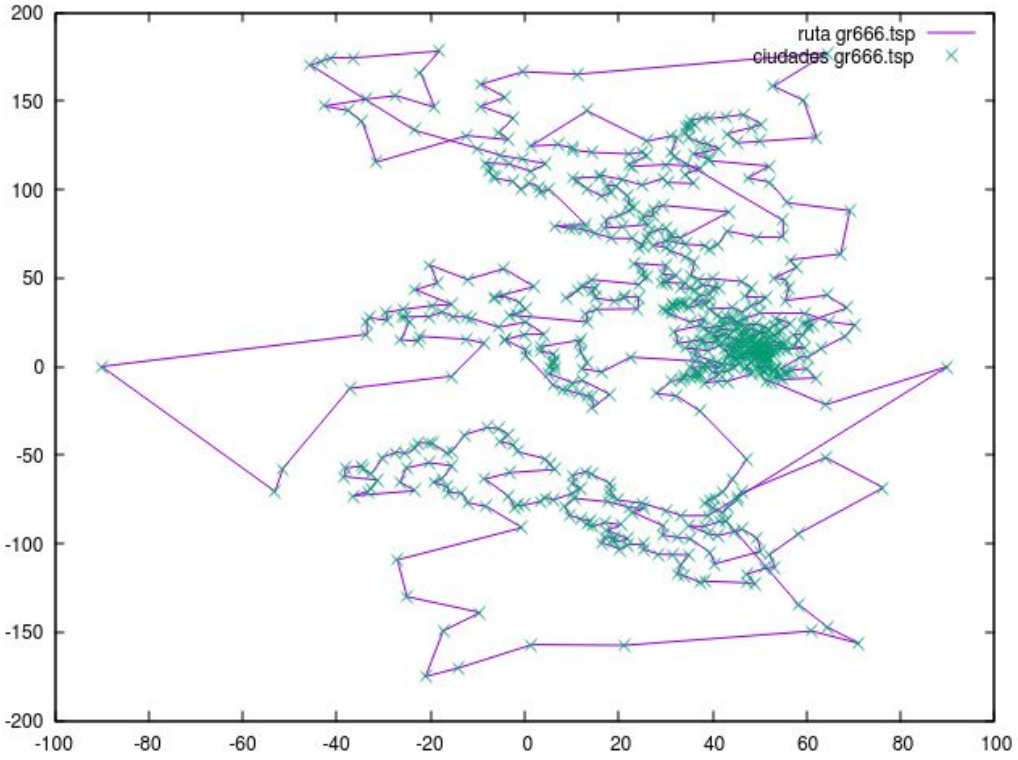


gr202_insercion

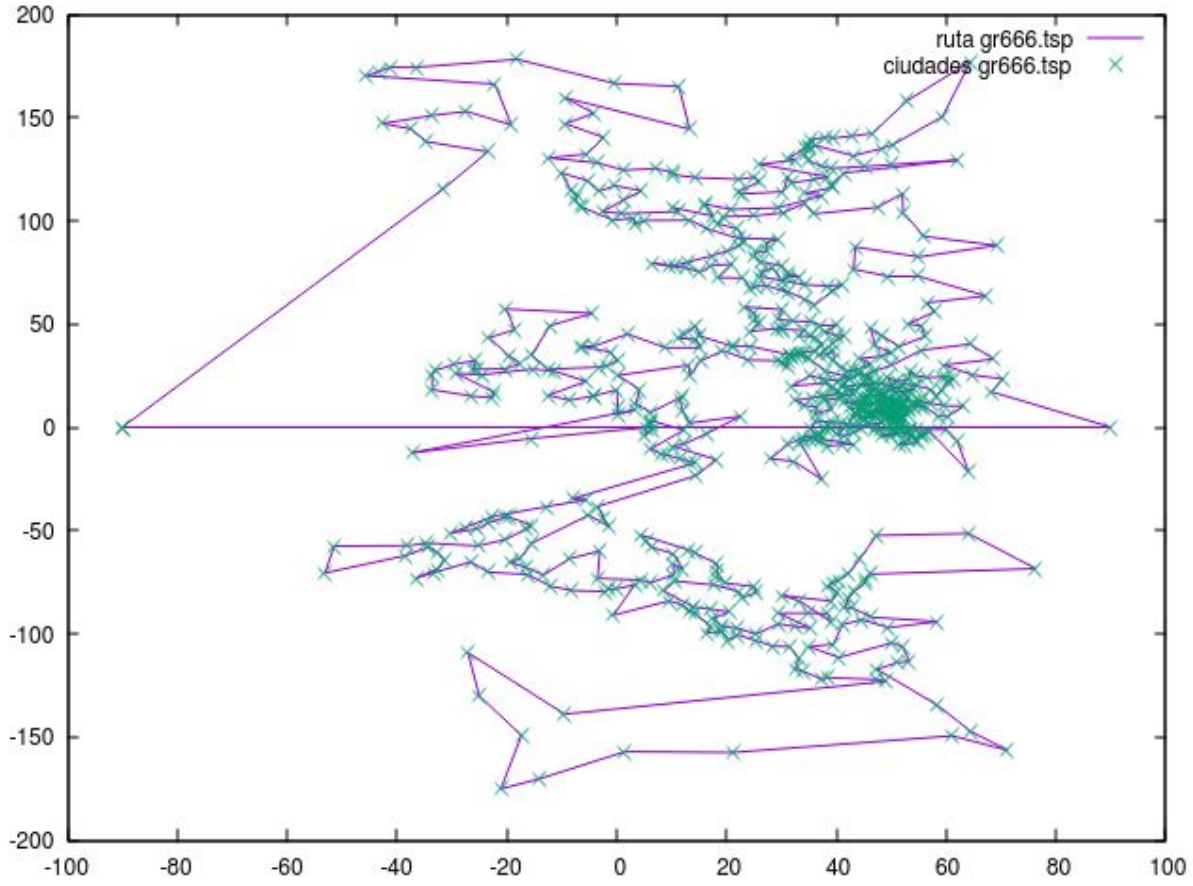
13. gr666



gr666_vmc

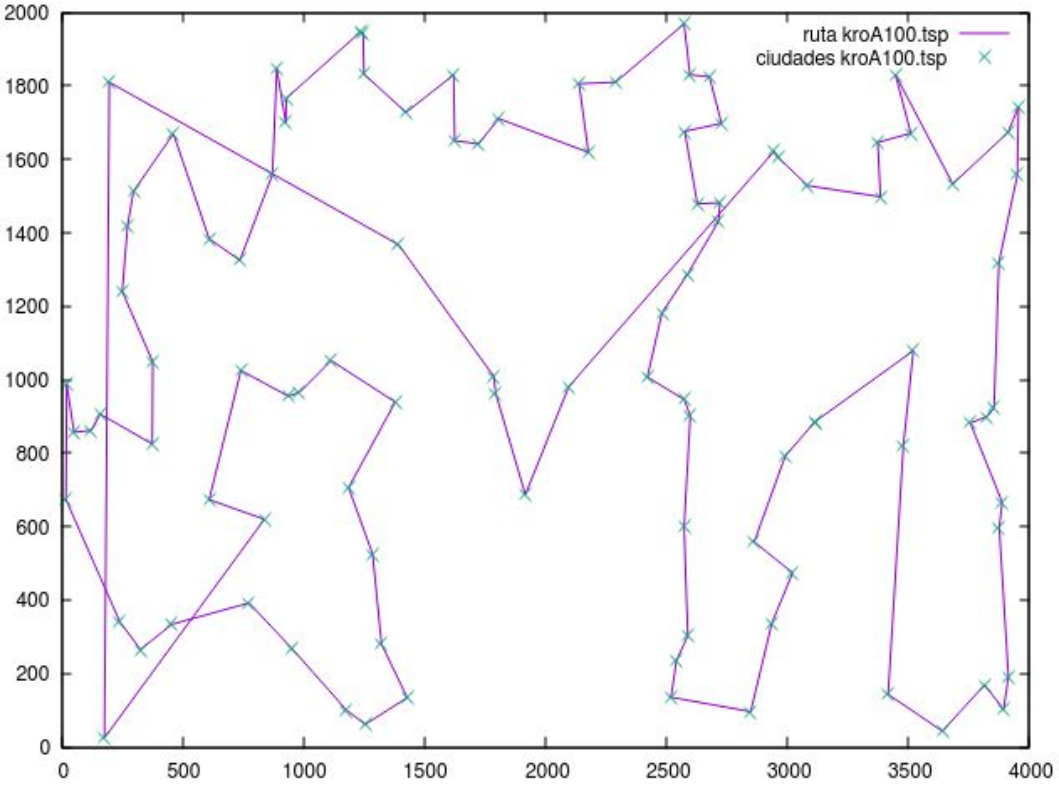


gr666_2-opt

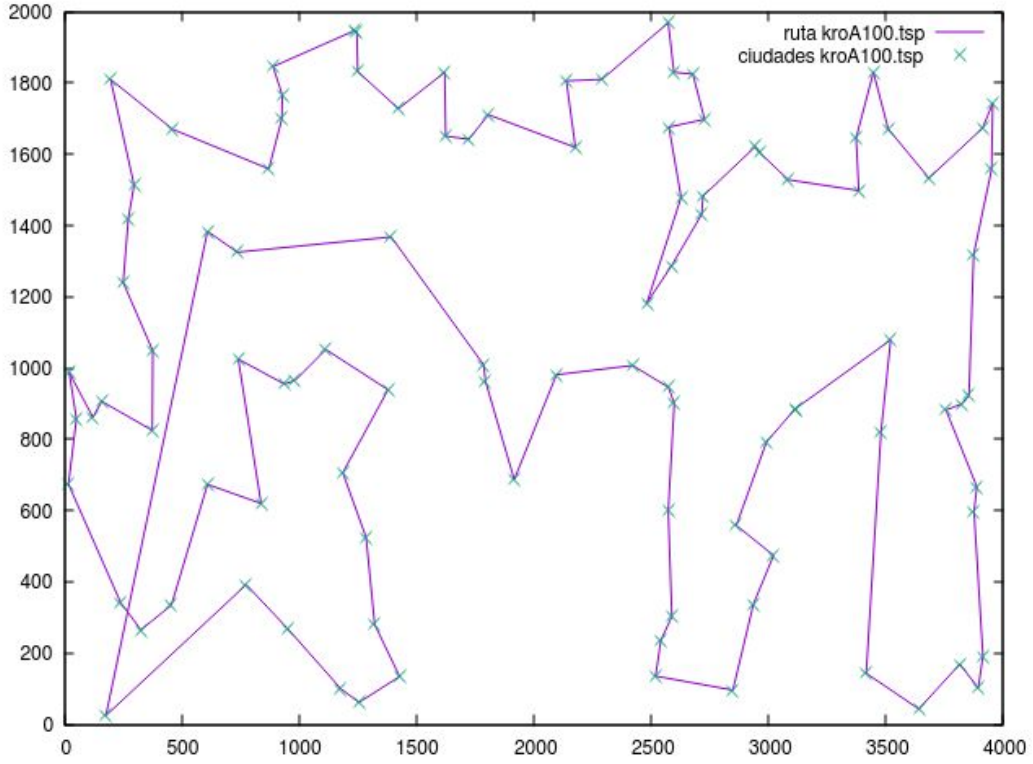


gr666_insercion

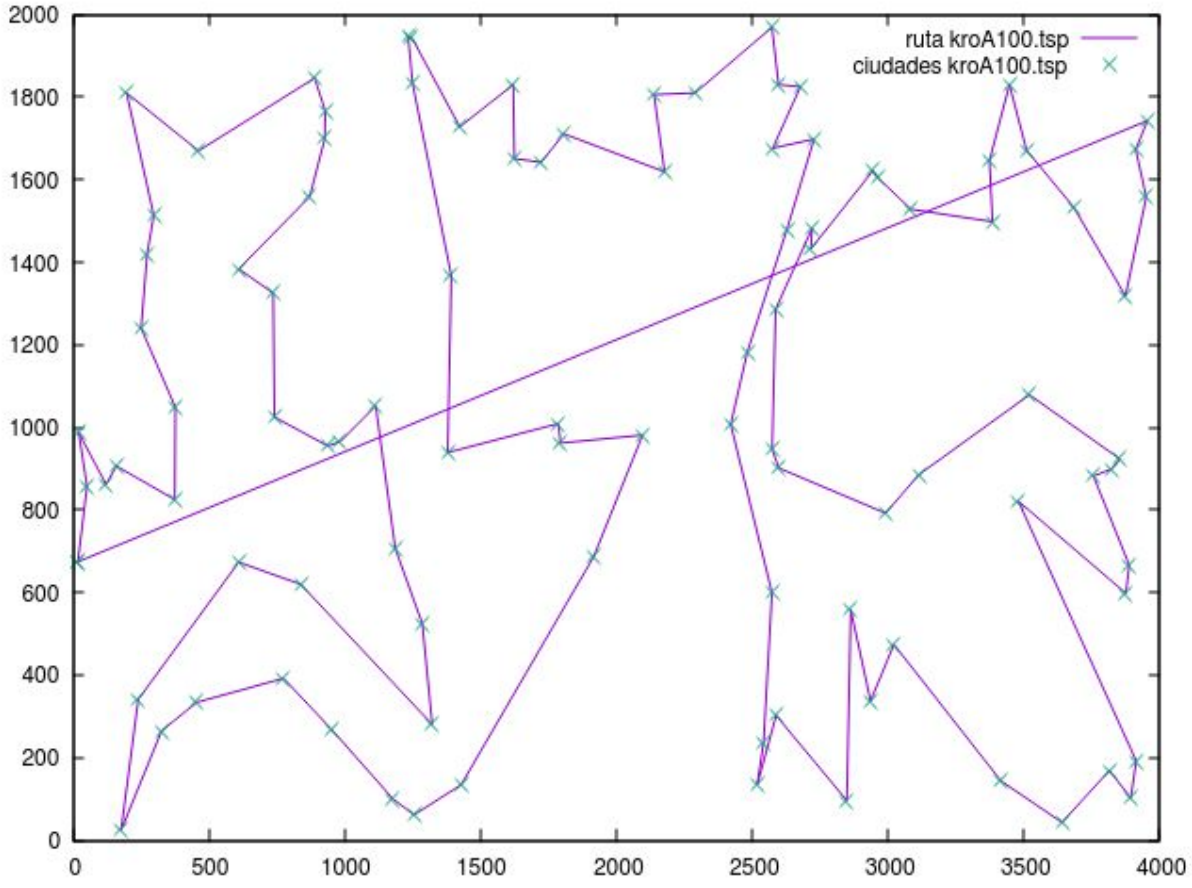
14. kroA100



kroA100_vmc

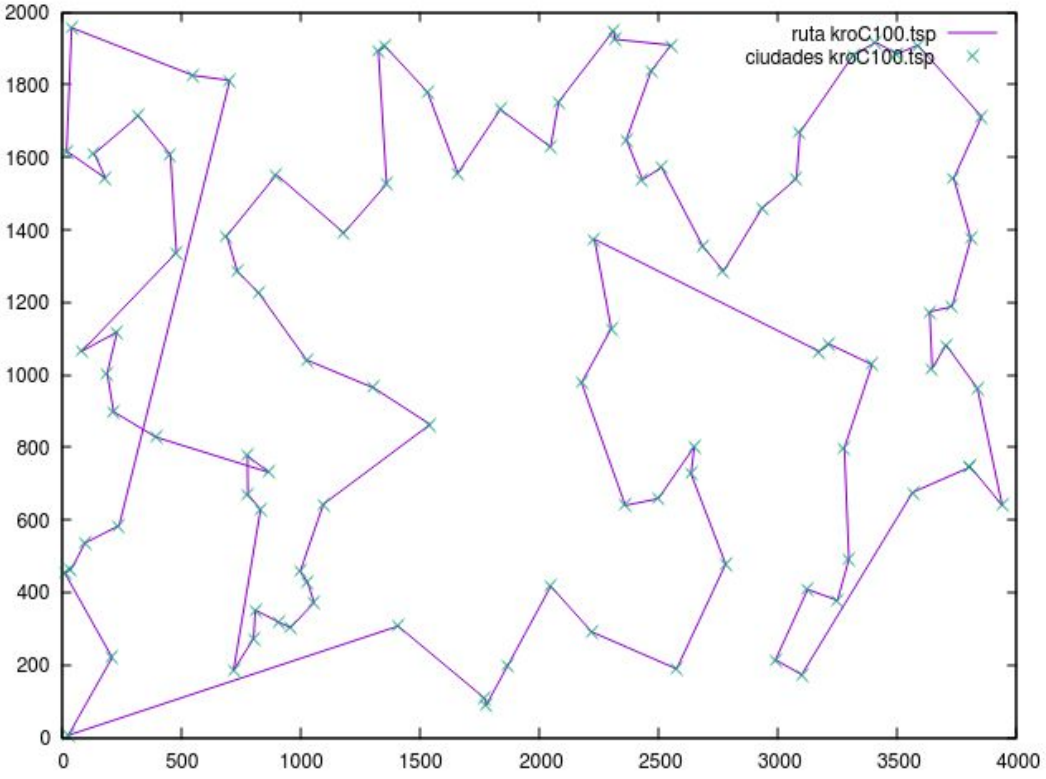


kroA100_2-opt

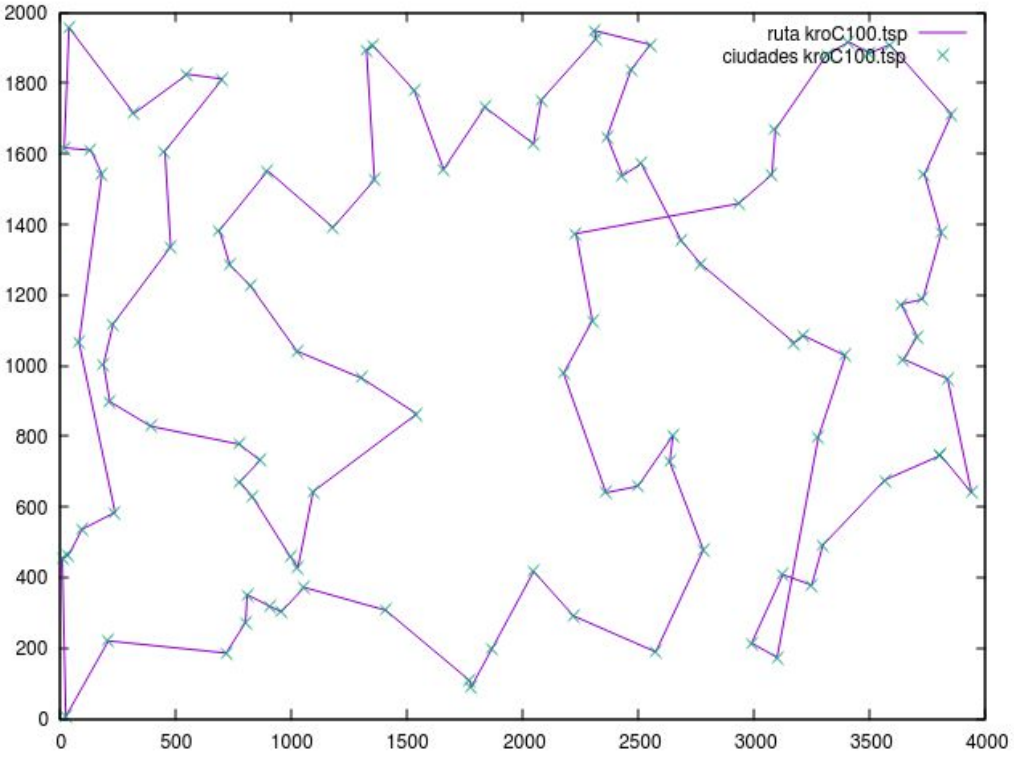


kroA100_insercion

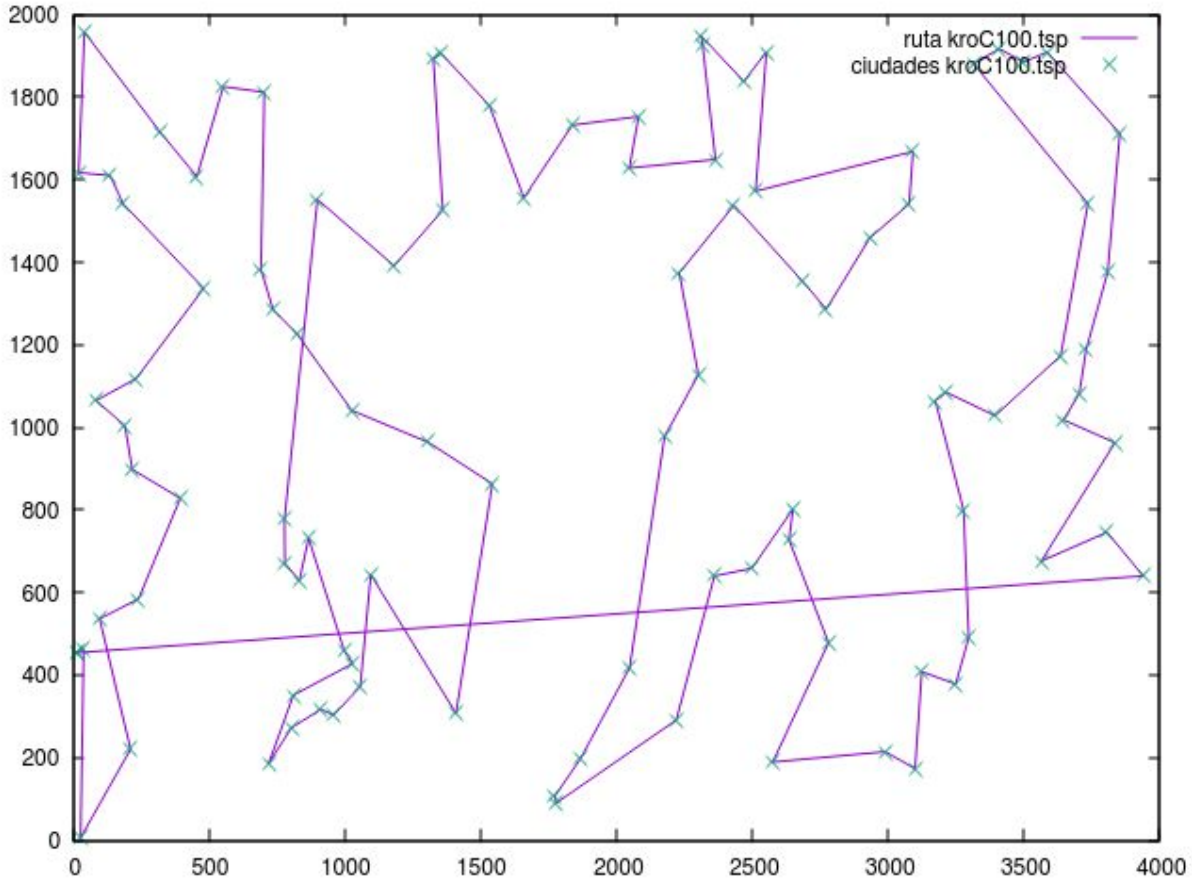
15. kroC100



kroC100_vmc

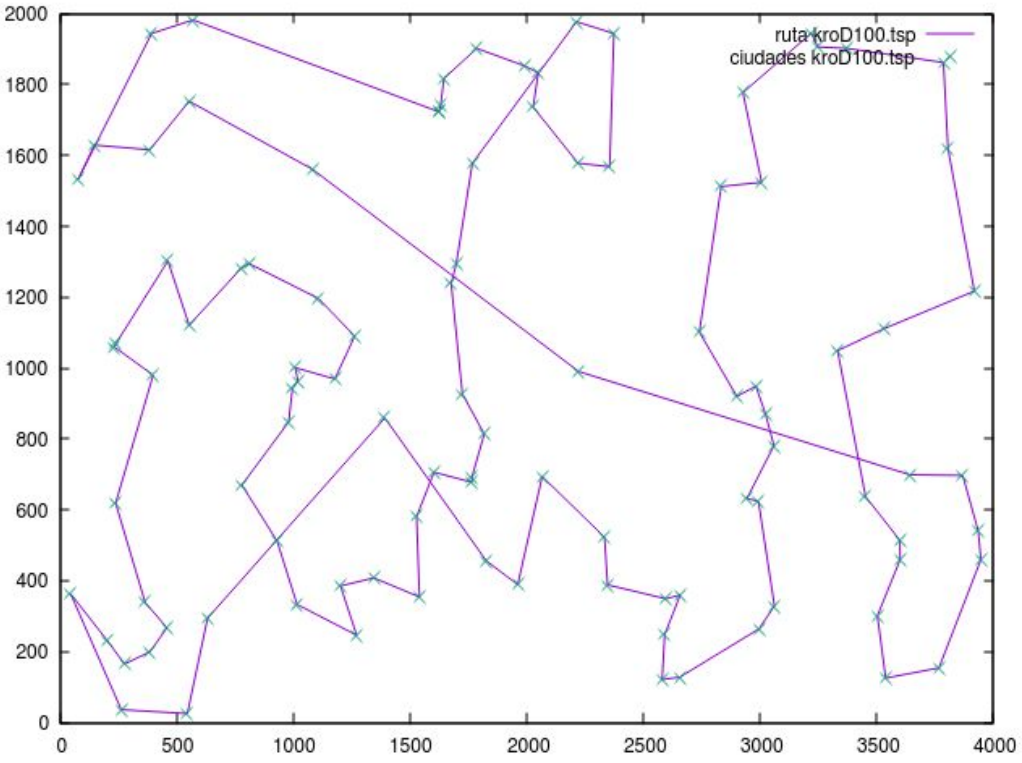


kroC100_2-opt

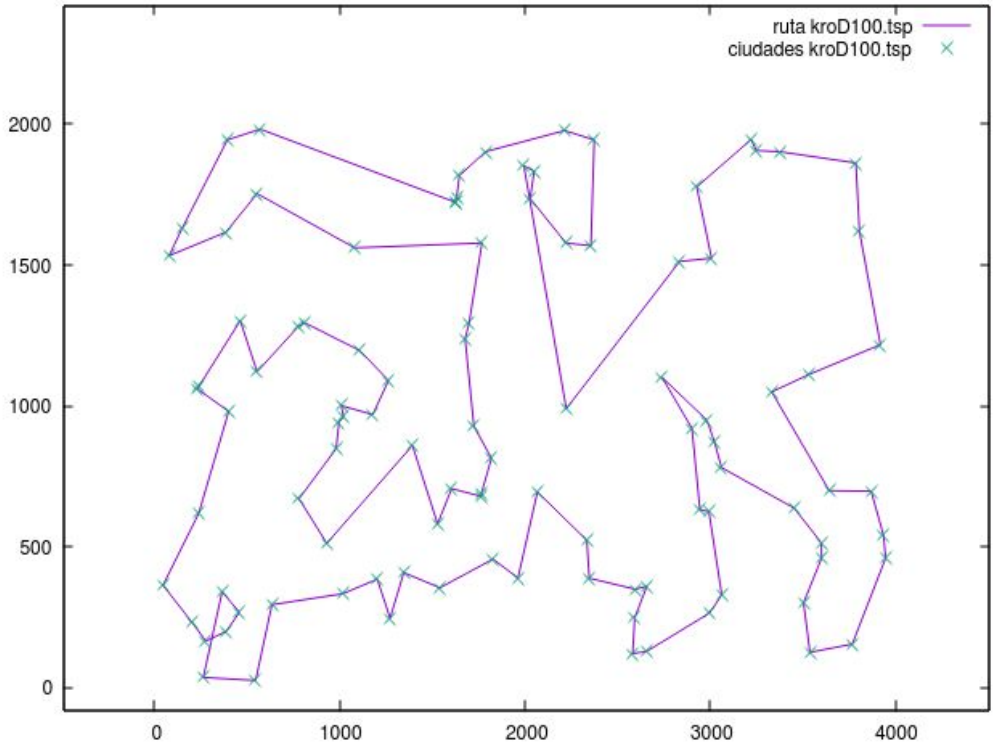


kroC100_insercion

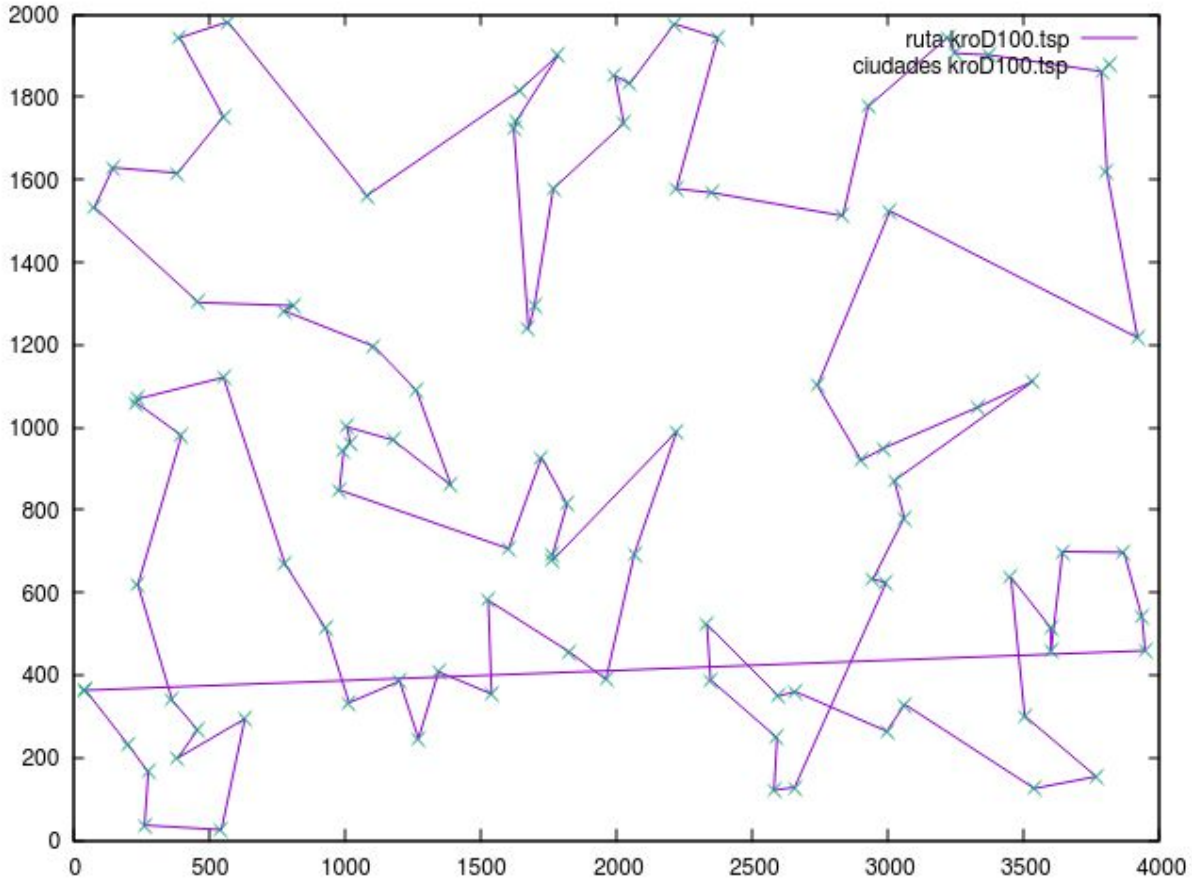
16. kroD100



kroD100_vmc

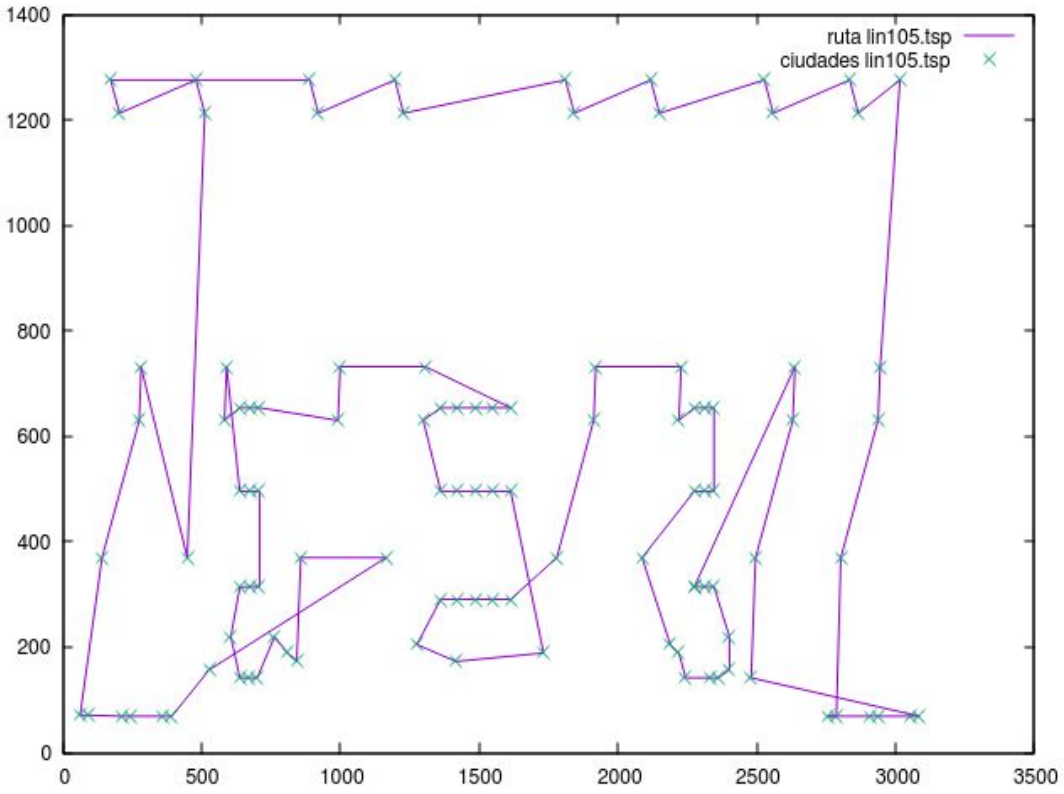


kroD100_2-opt

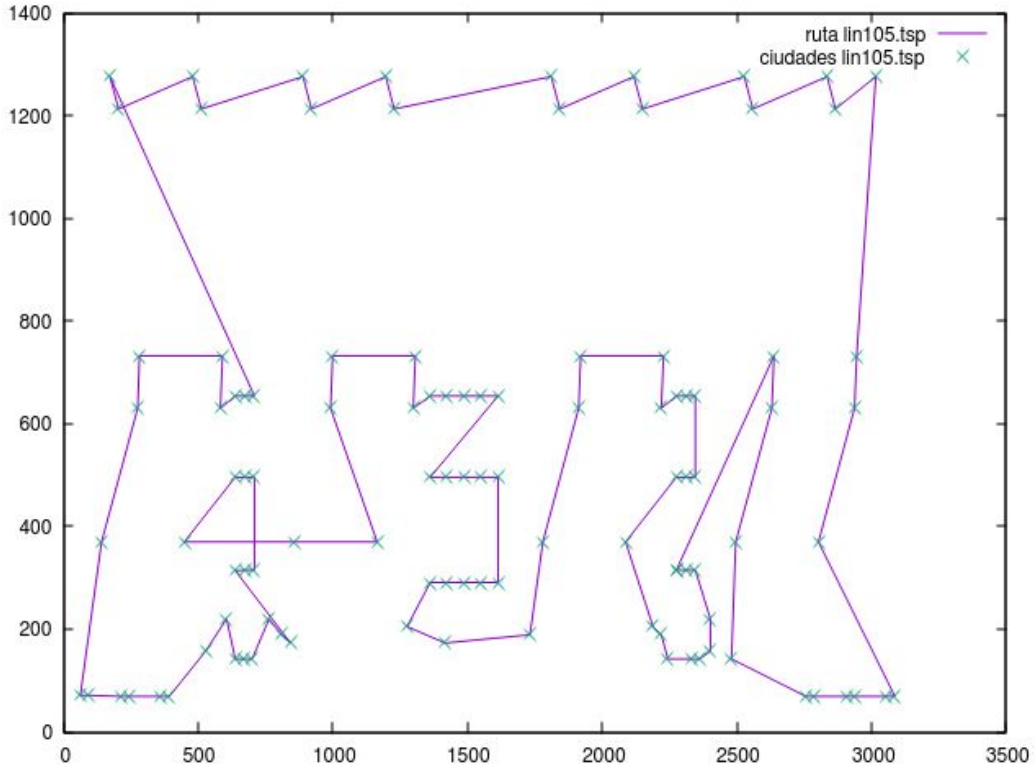


kroD100_insercion

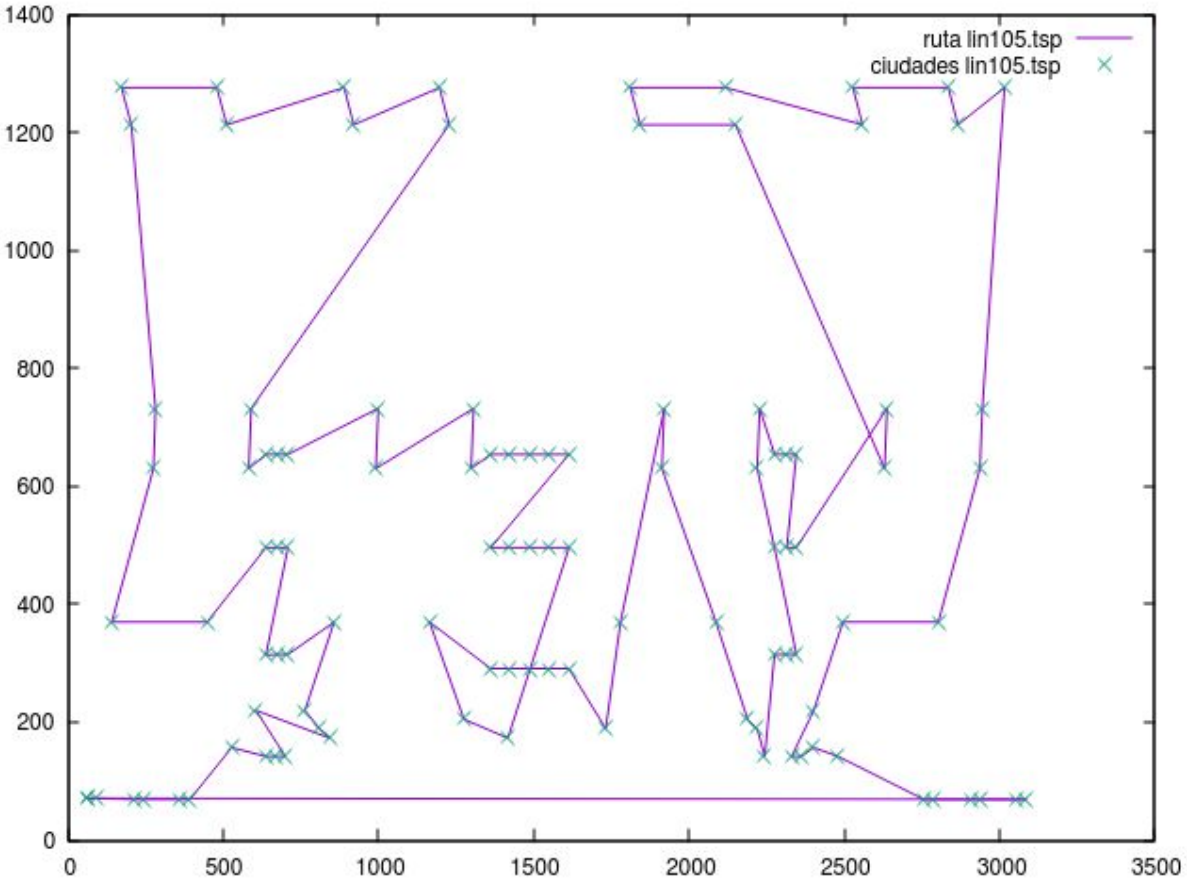
17. lin105



lin105_vmc

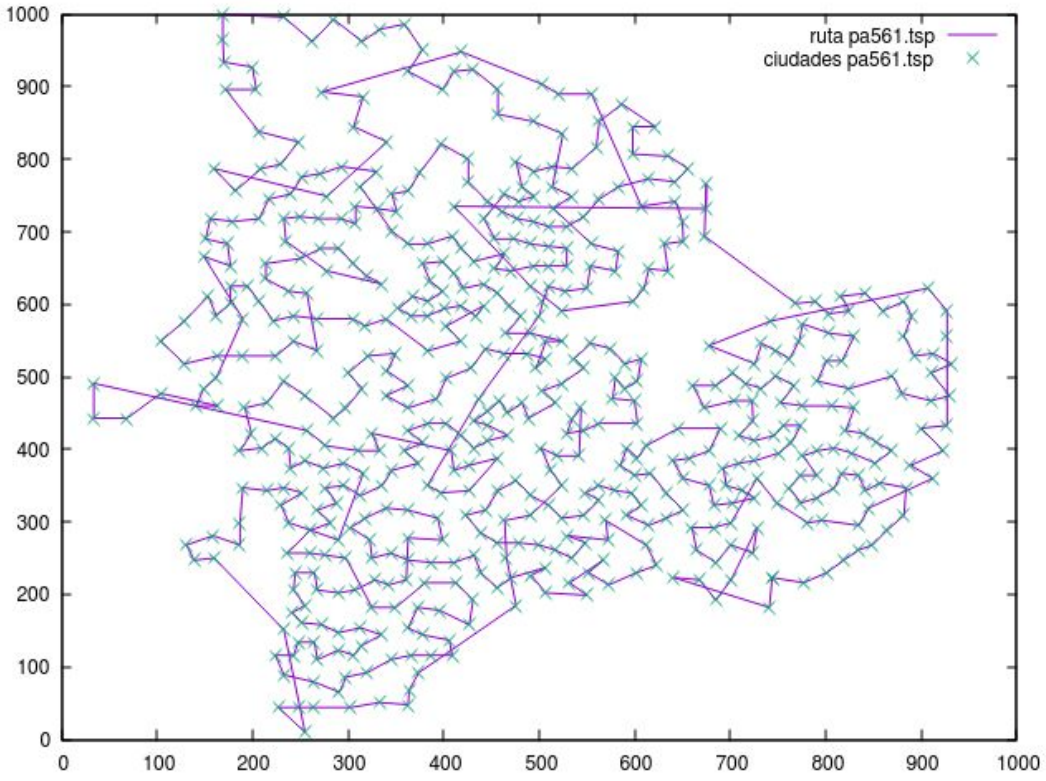


lin105_2-opt

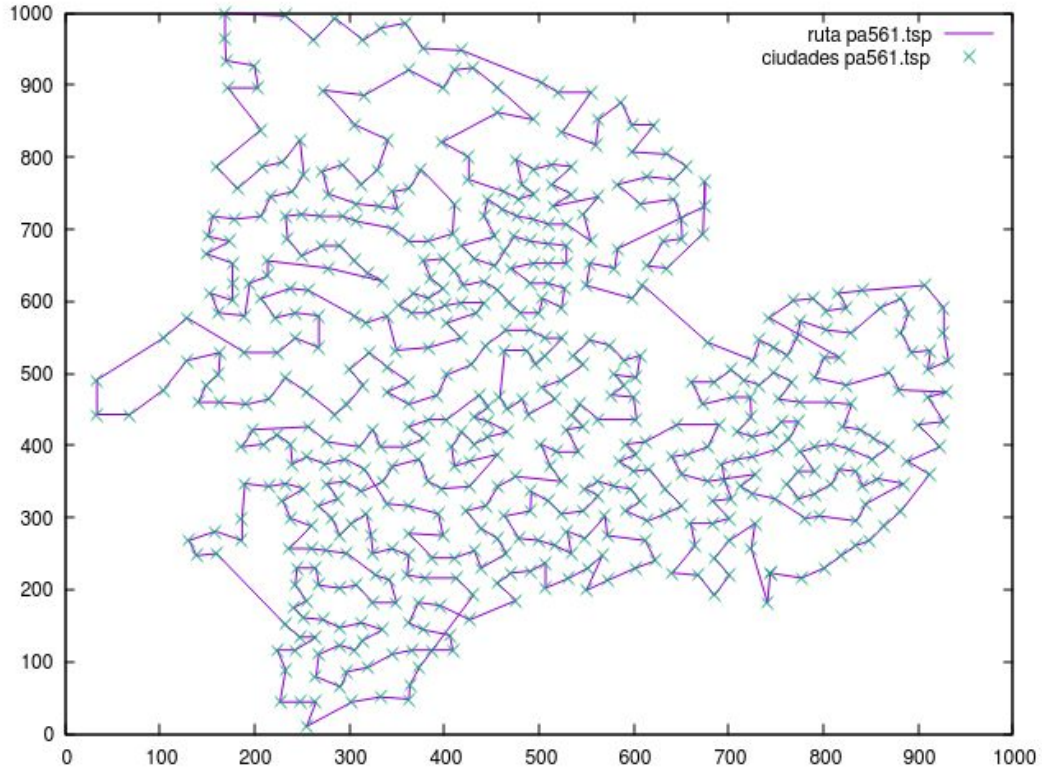


lin105_insercion

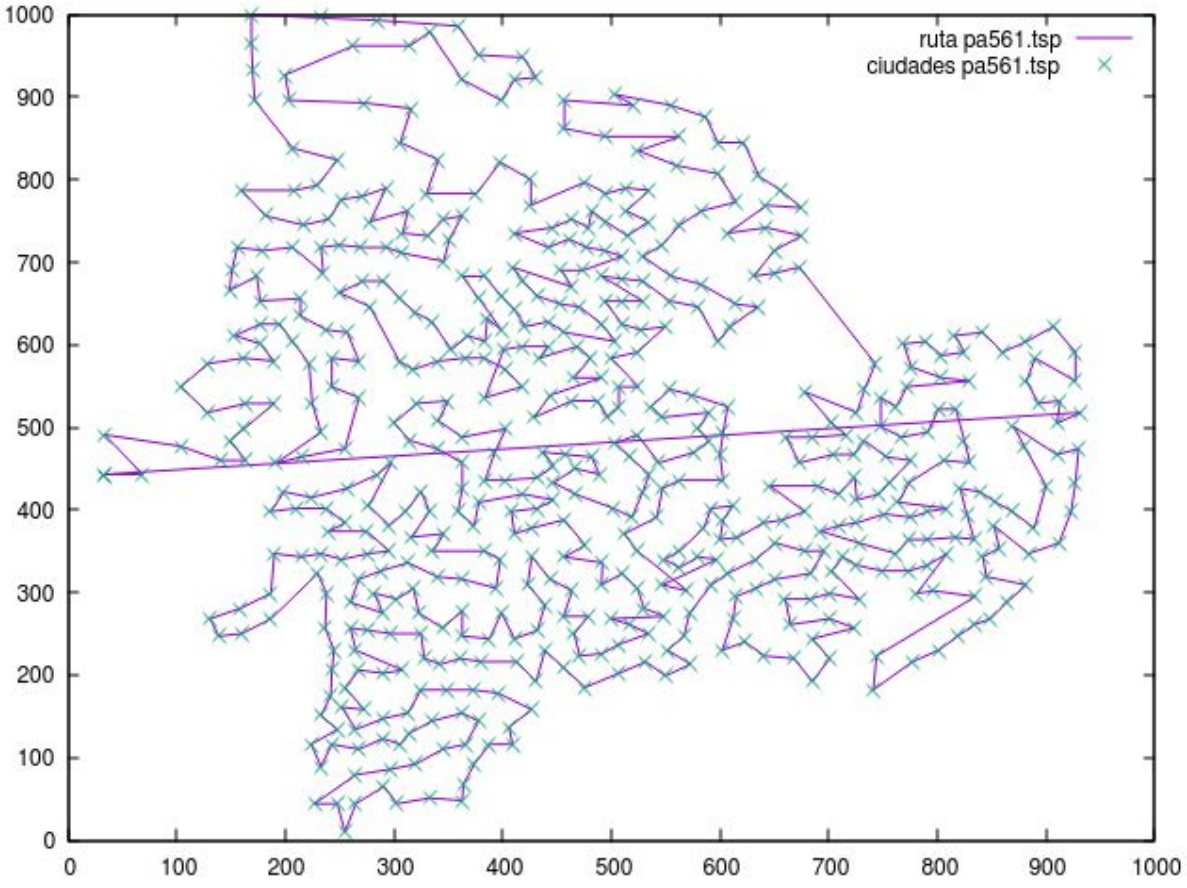
18. pa561



pa561_vmc

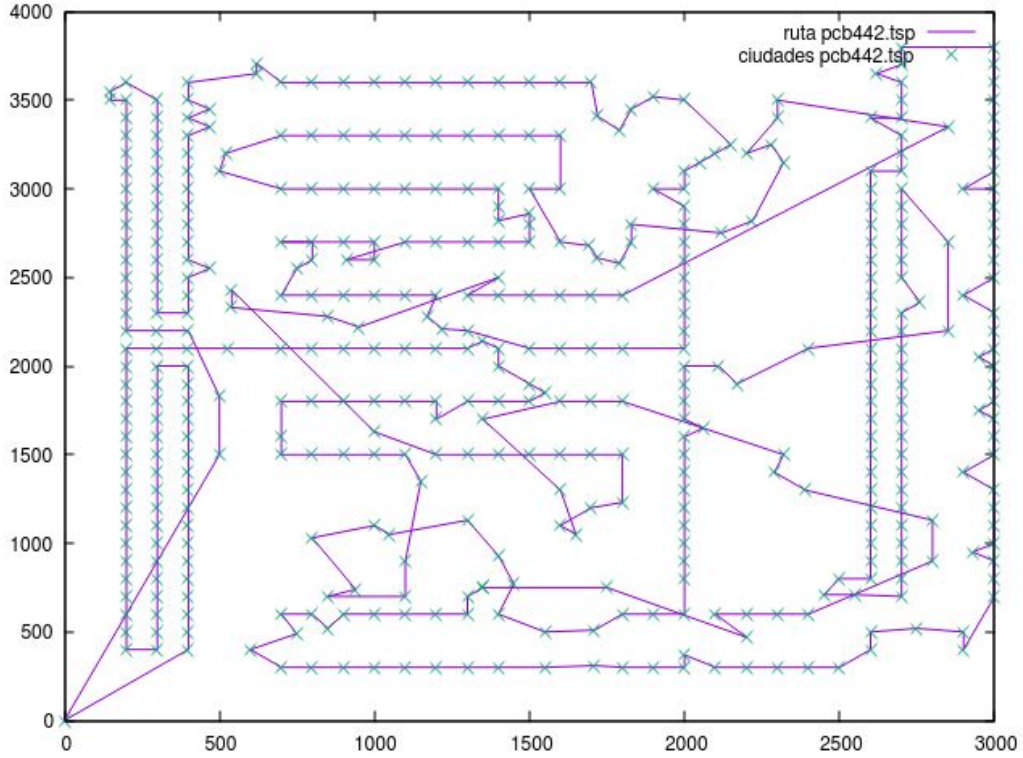


pa561_2-opt

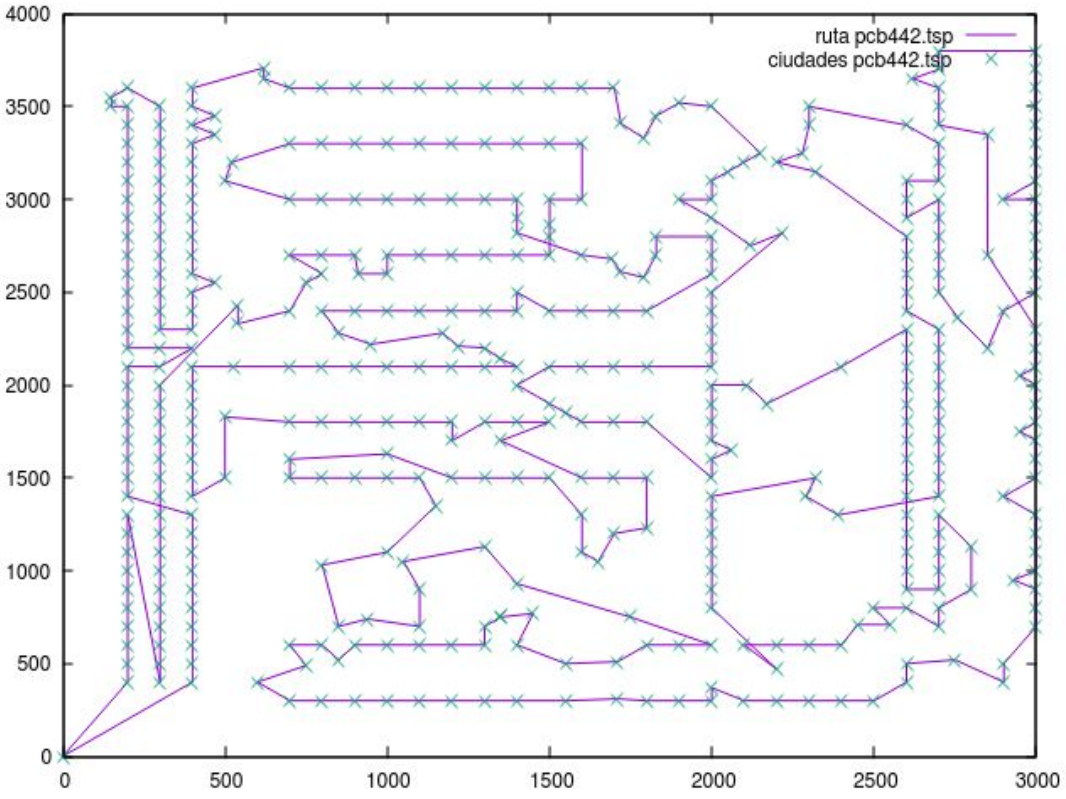


pa561_insercion

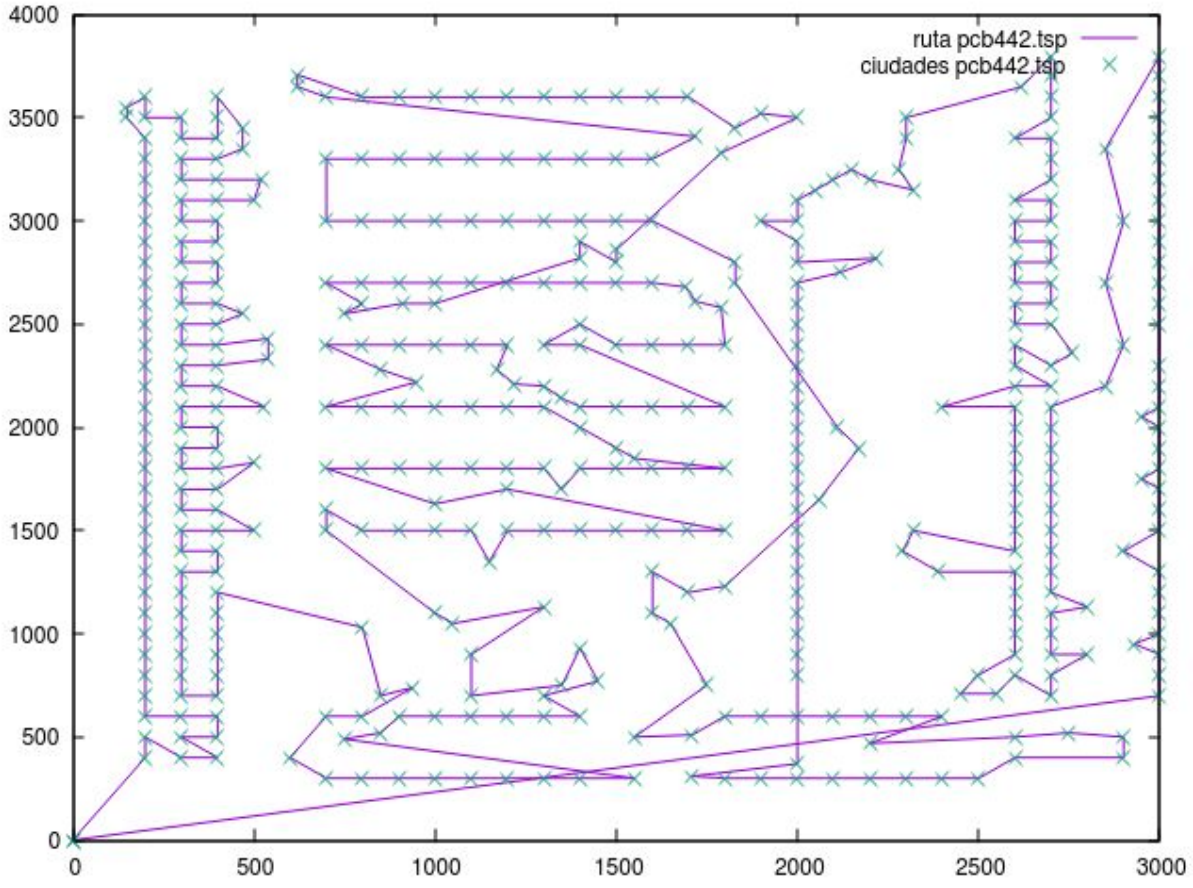
19. pcb442



pcb442_vmc

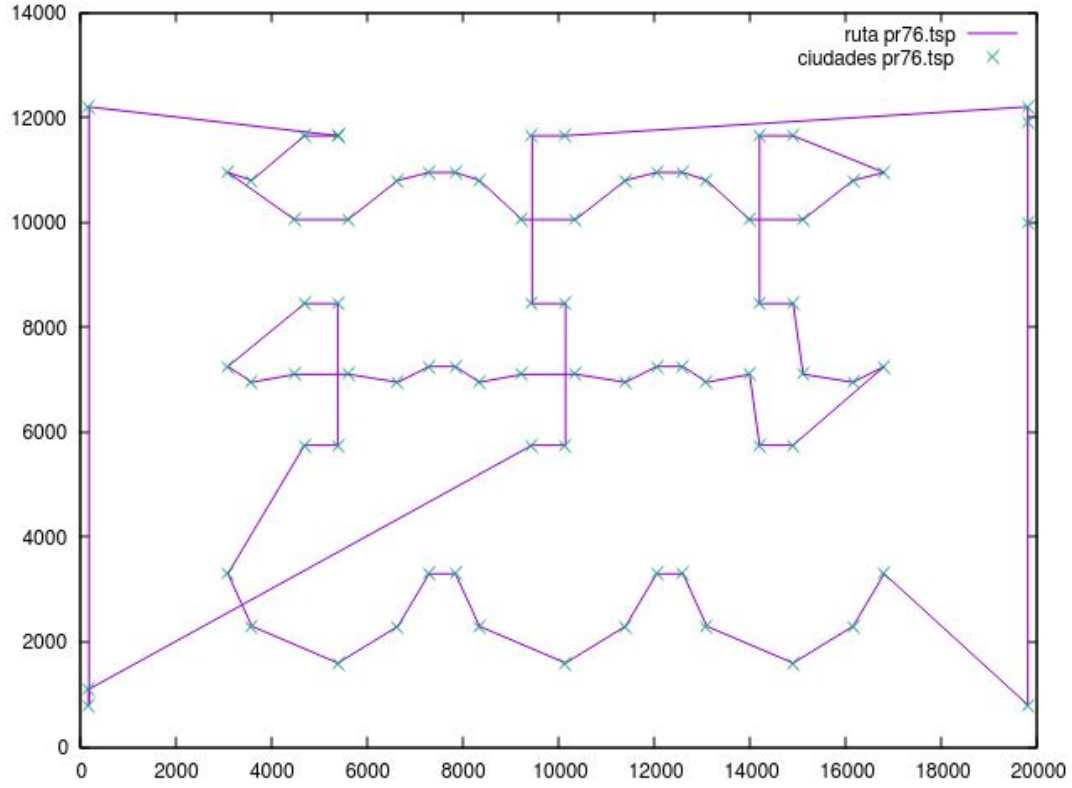


pcb442_2-opt

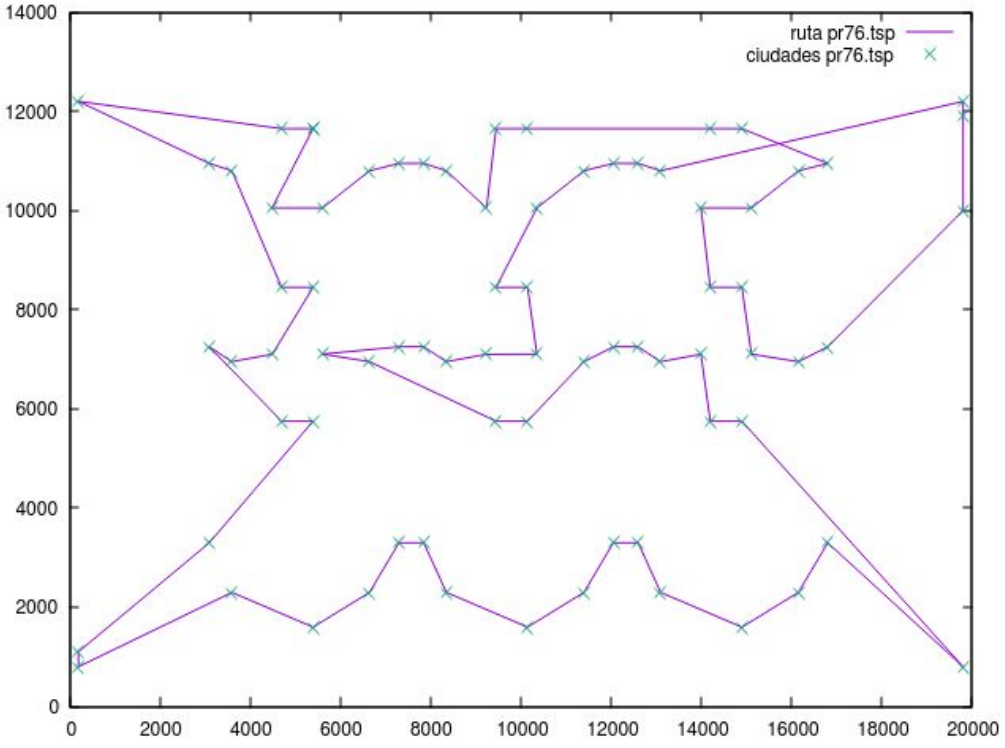


pcb442_insercion

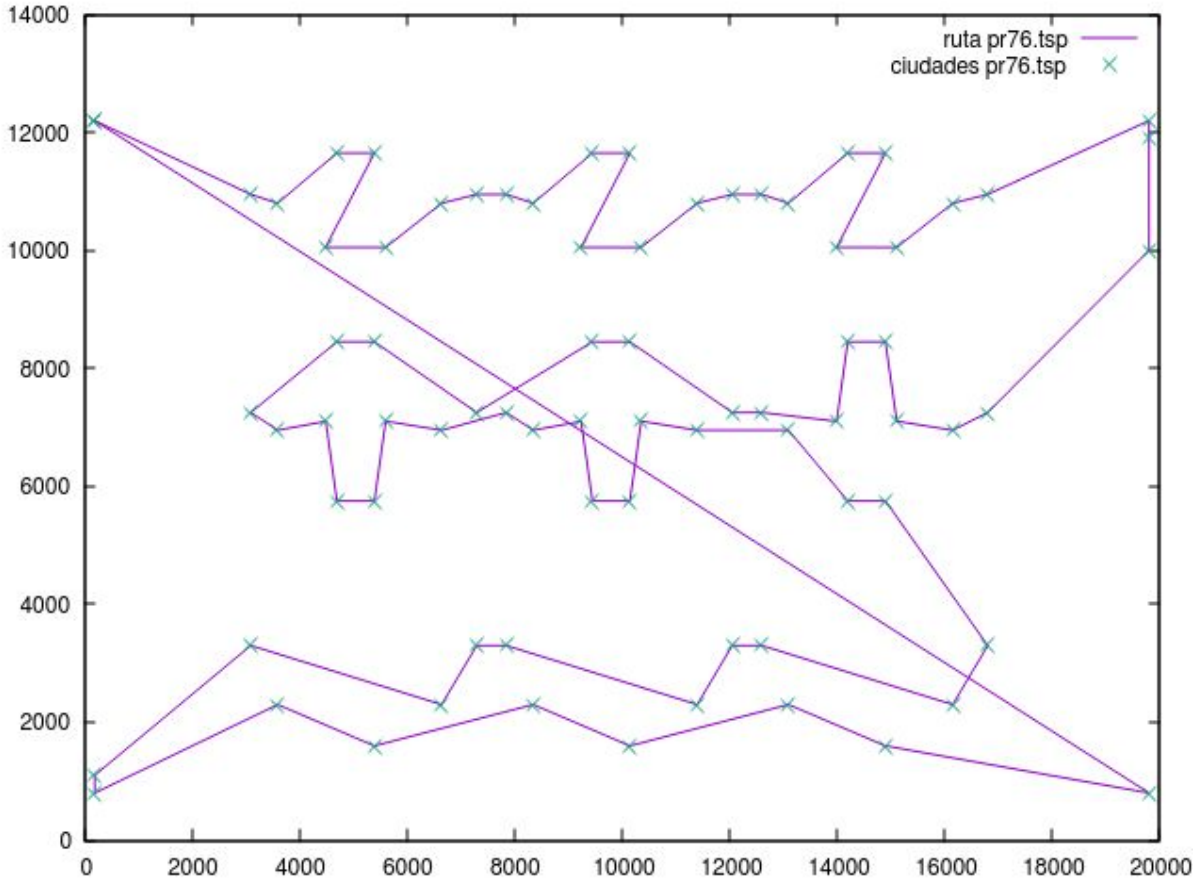
20. pr76



pr76_vmc

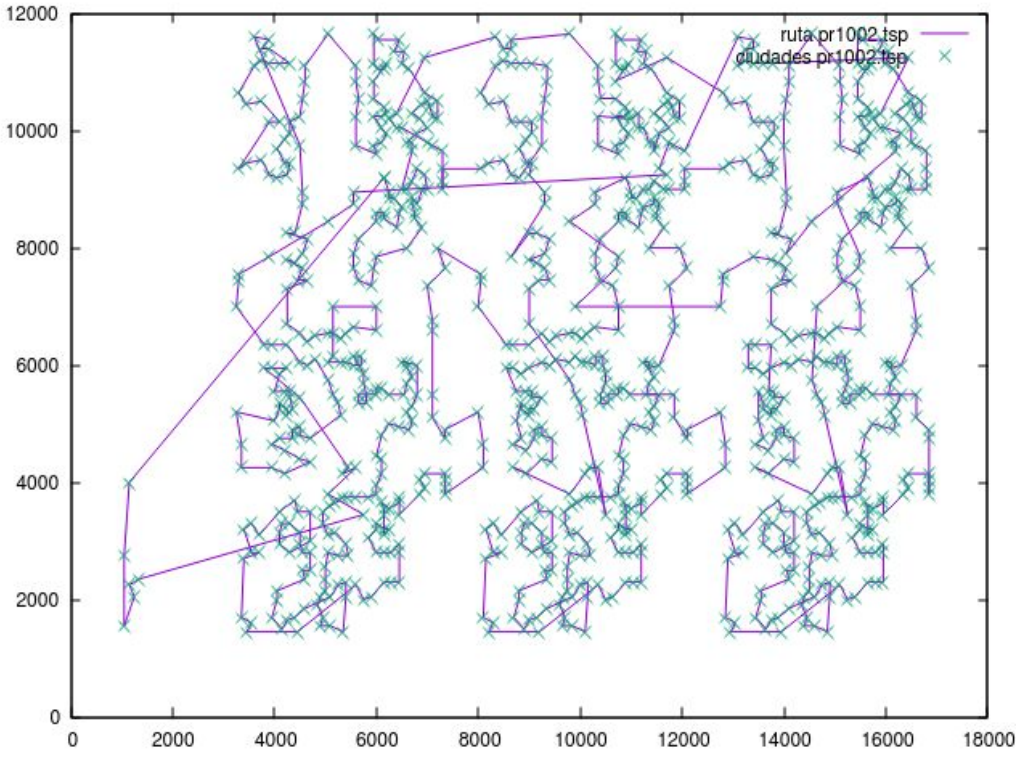


pr76_2-opt

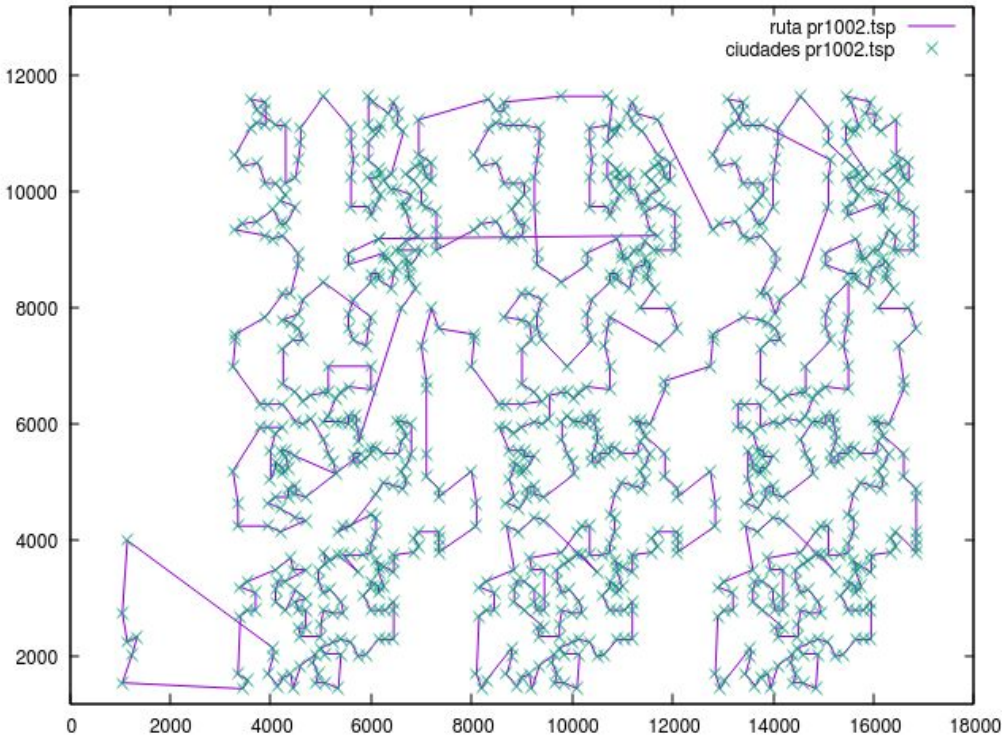


pr76_insercion

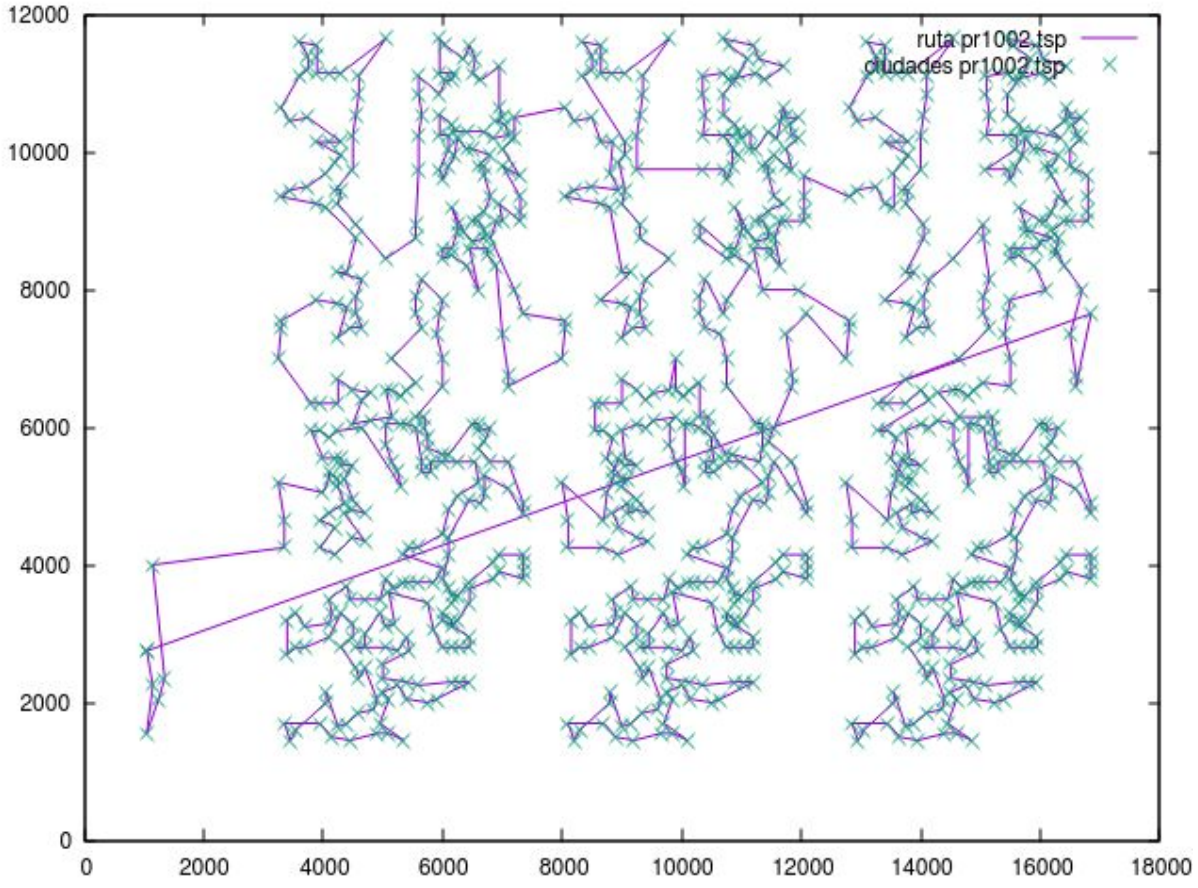
21. pr1002



pr1002_vmc

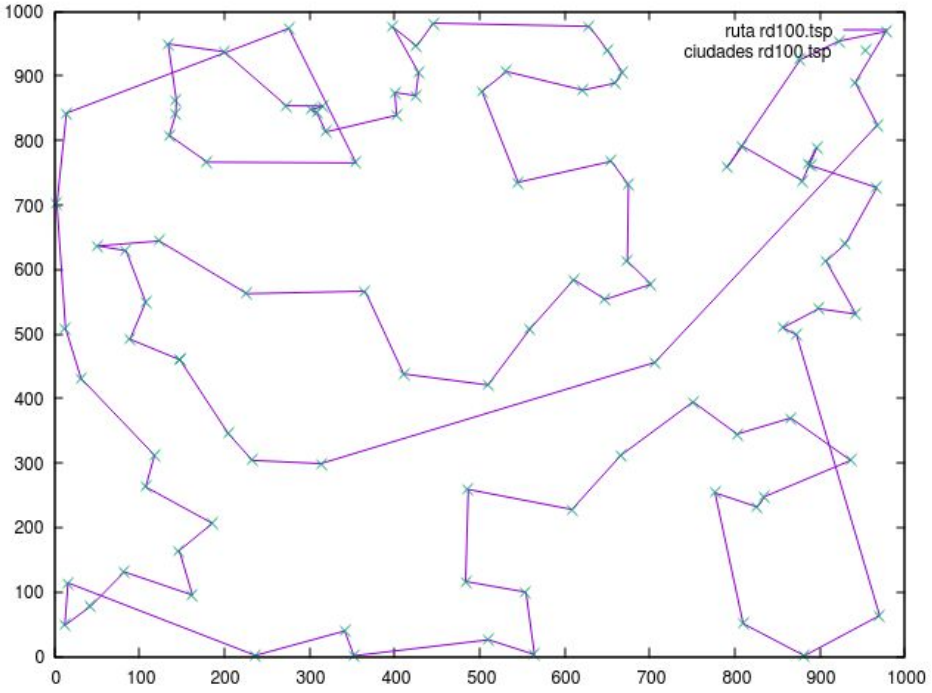


pr1002_2-opt

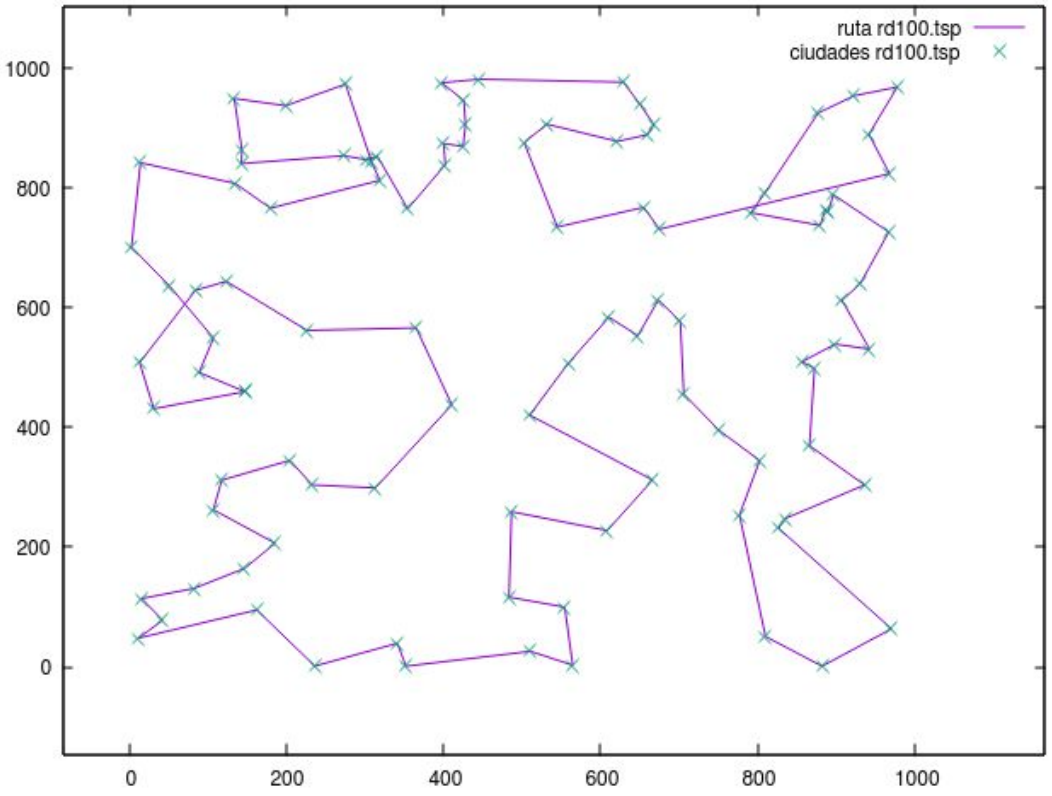


pr1002_insercion

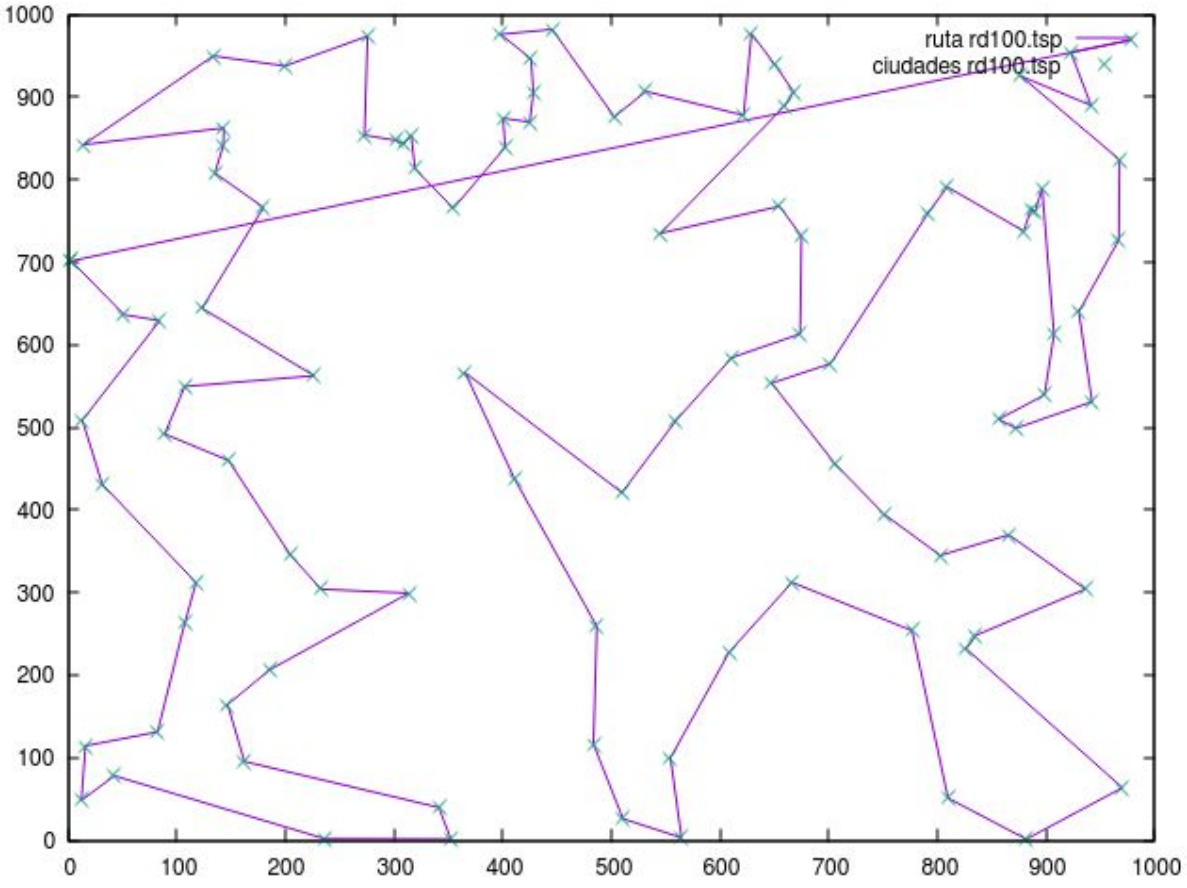
22. rd100



rd100_vmc

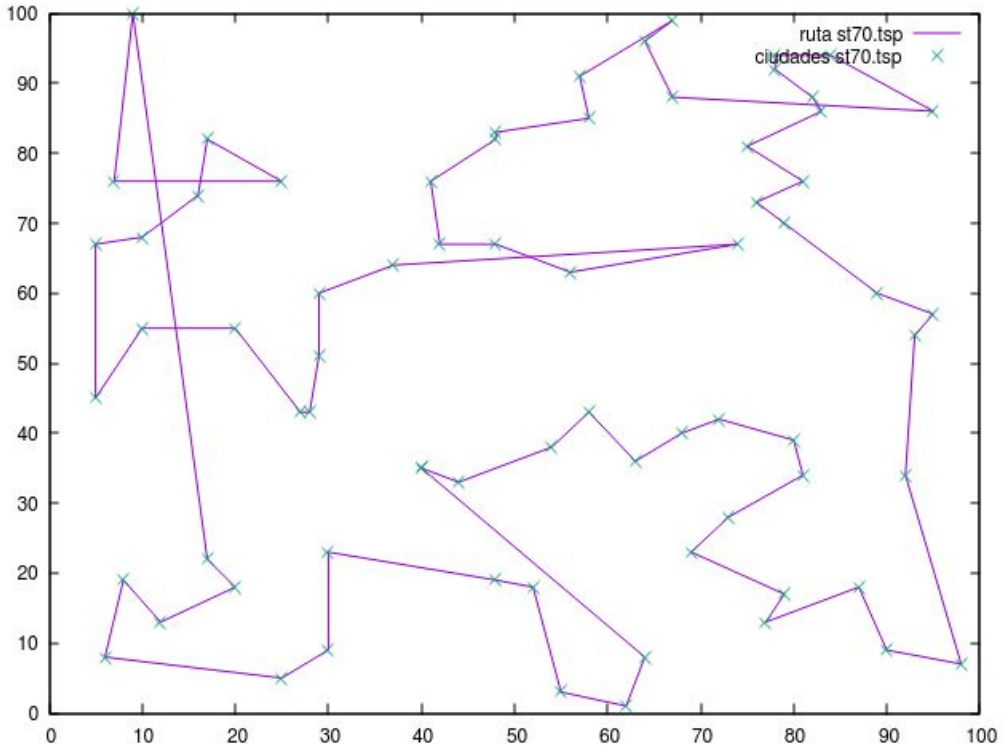


rd100_2-opt

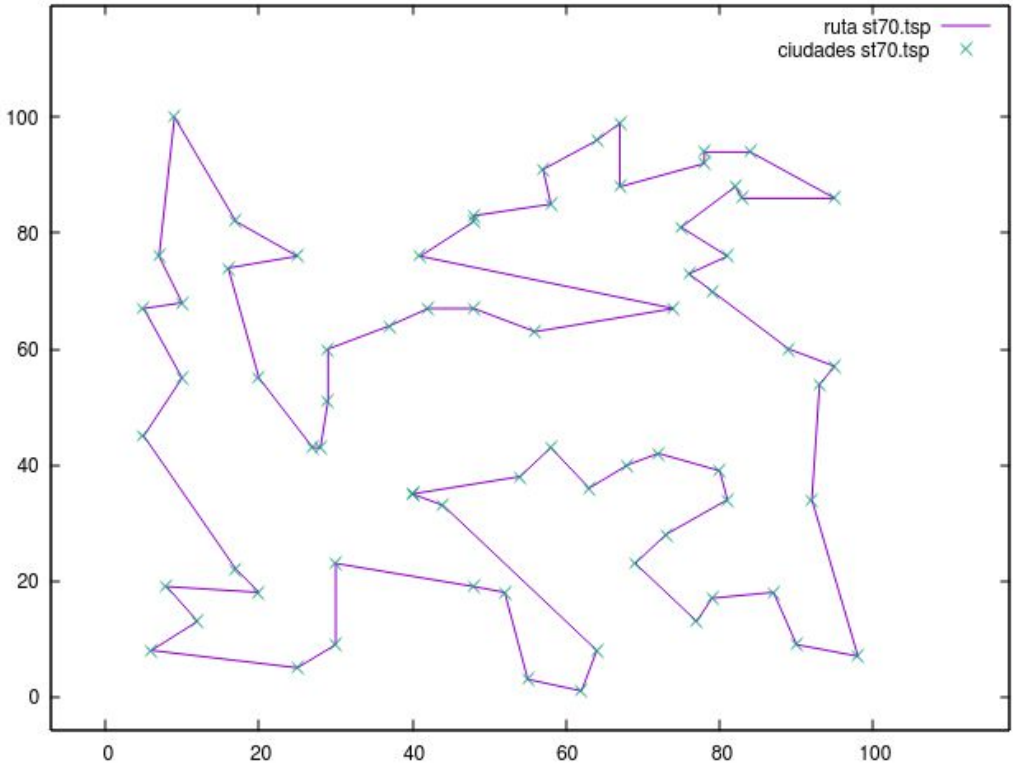


rd100_insercion

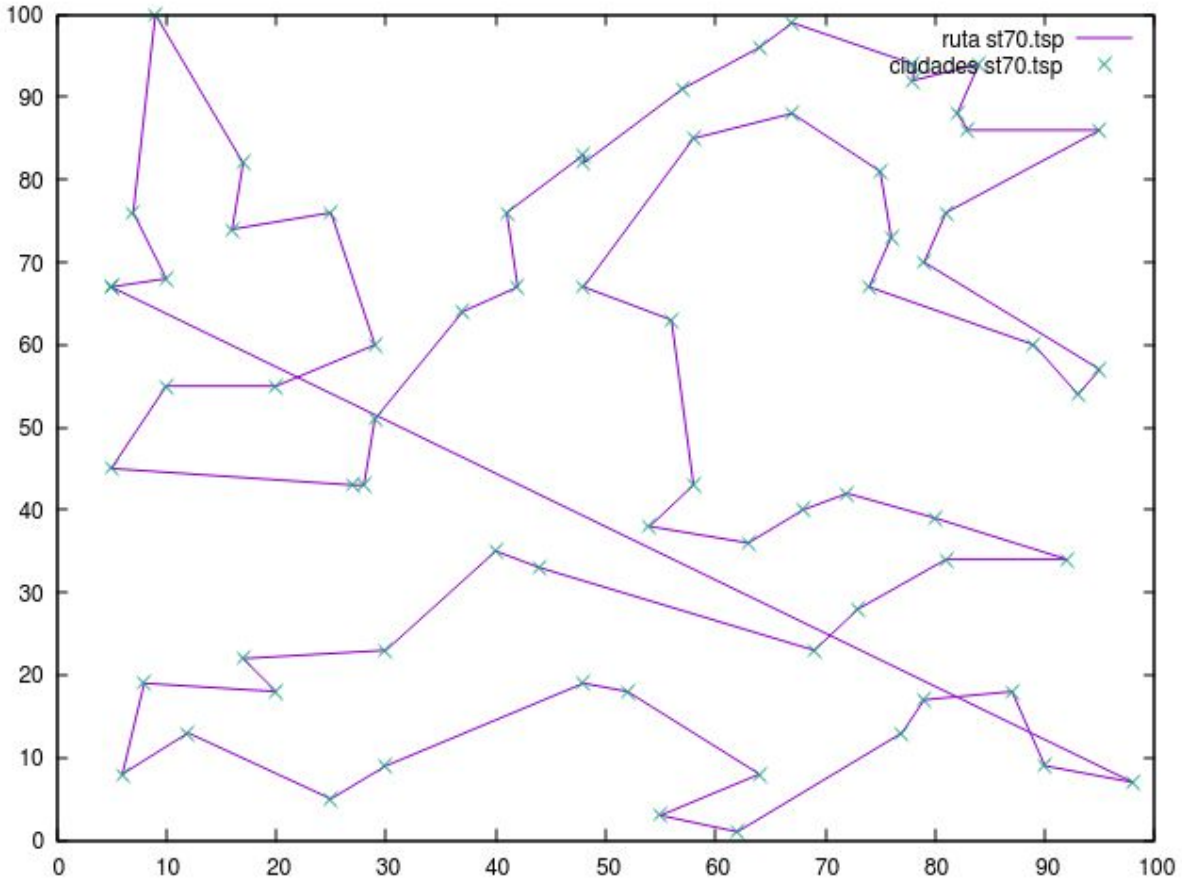
23. st70



st70_vmc

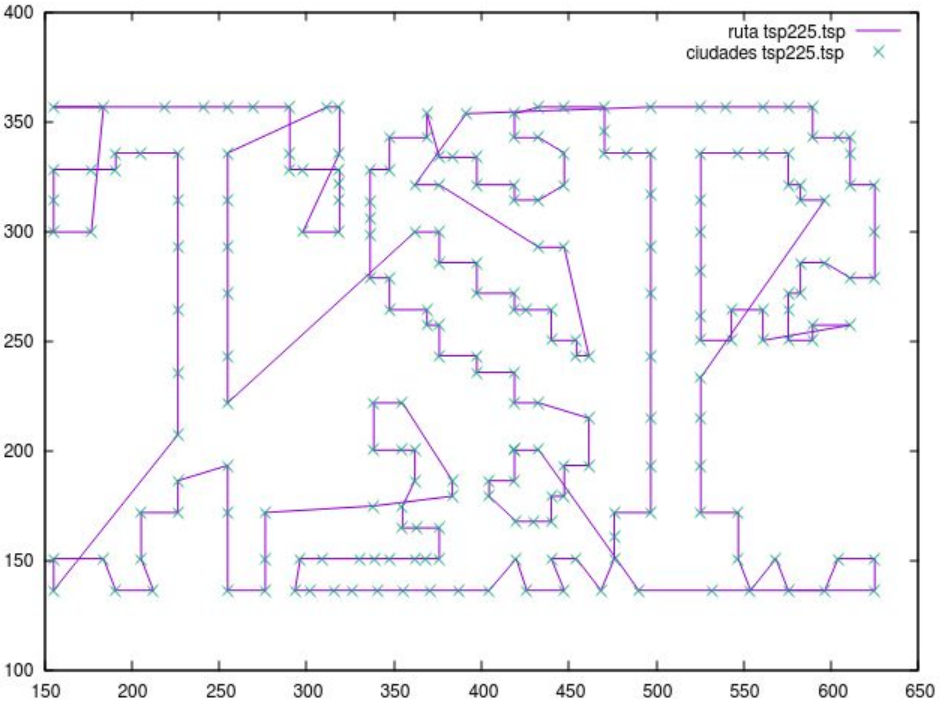


st70_2-opt

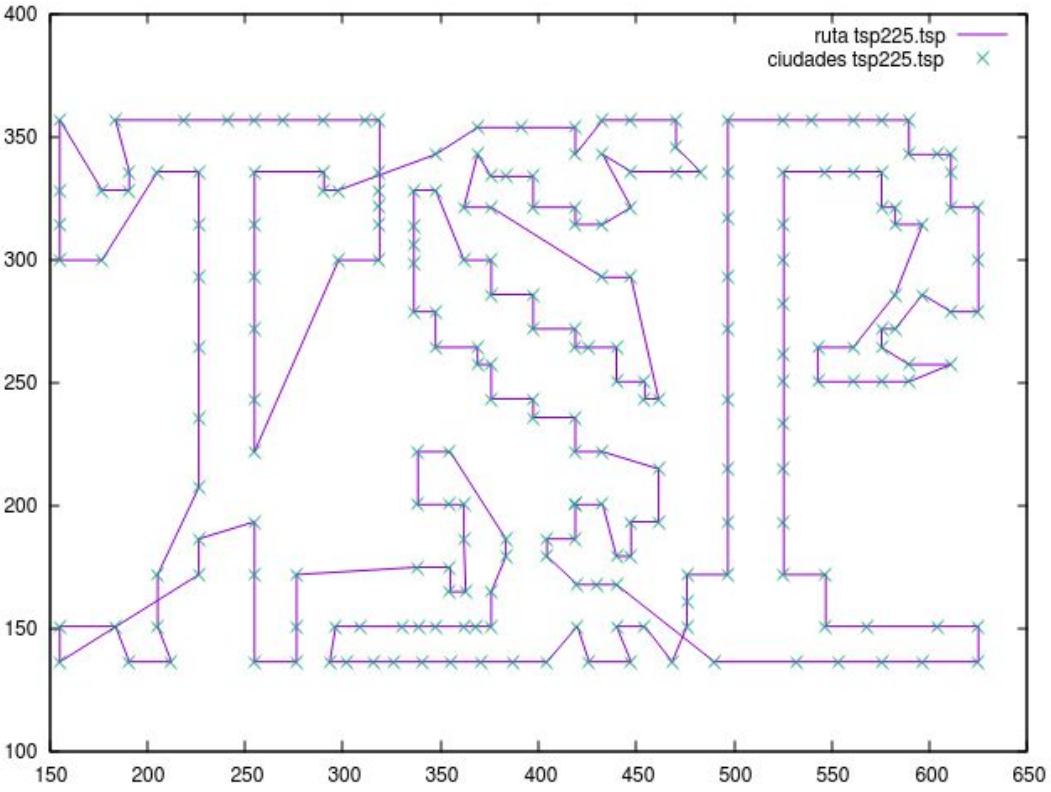


st70_insercion

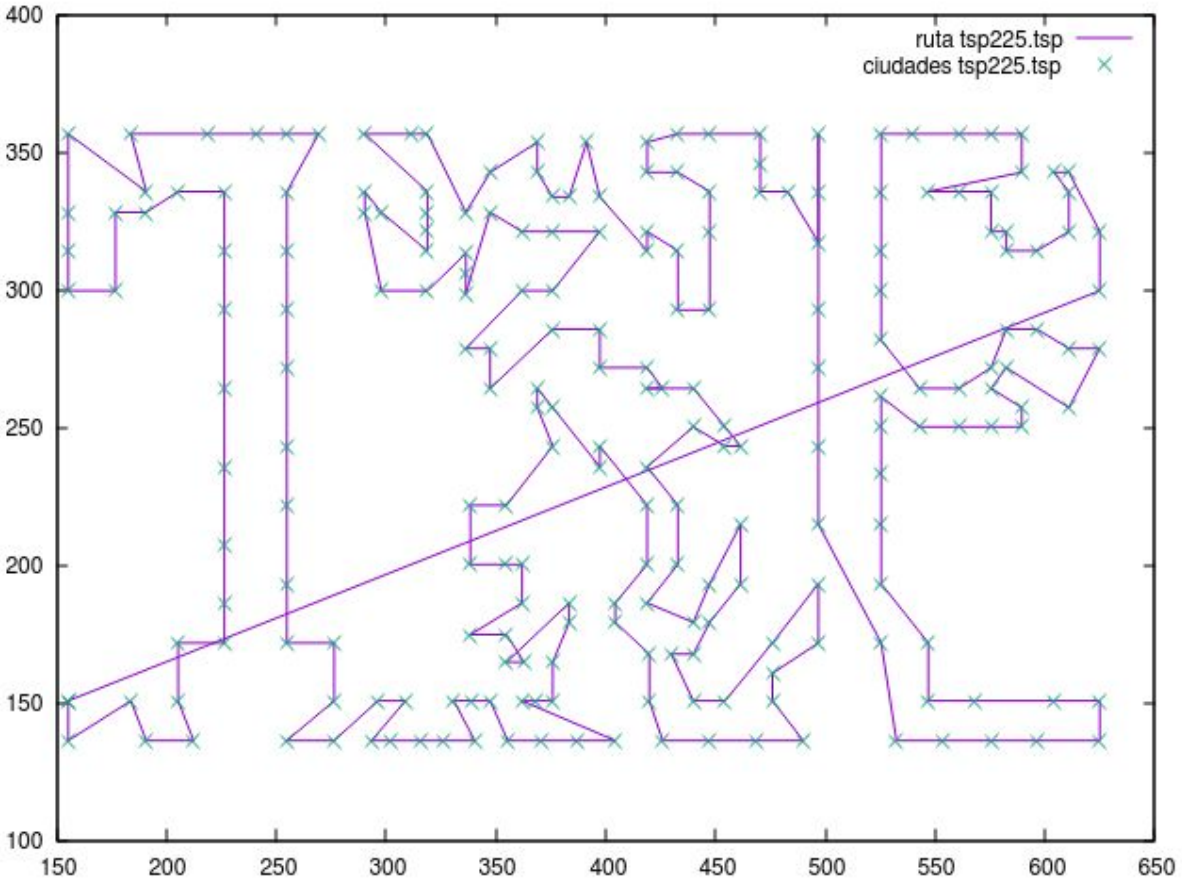
24. tsp225



tsp225_vmc

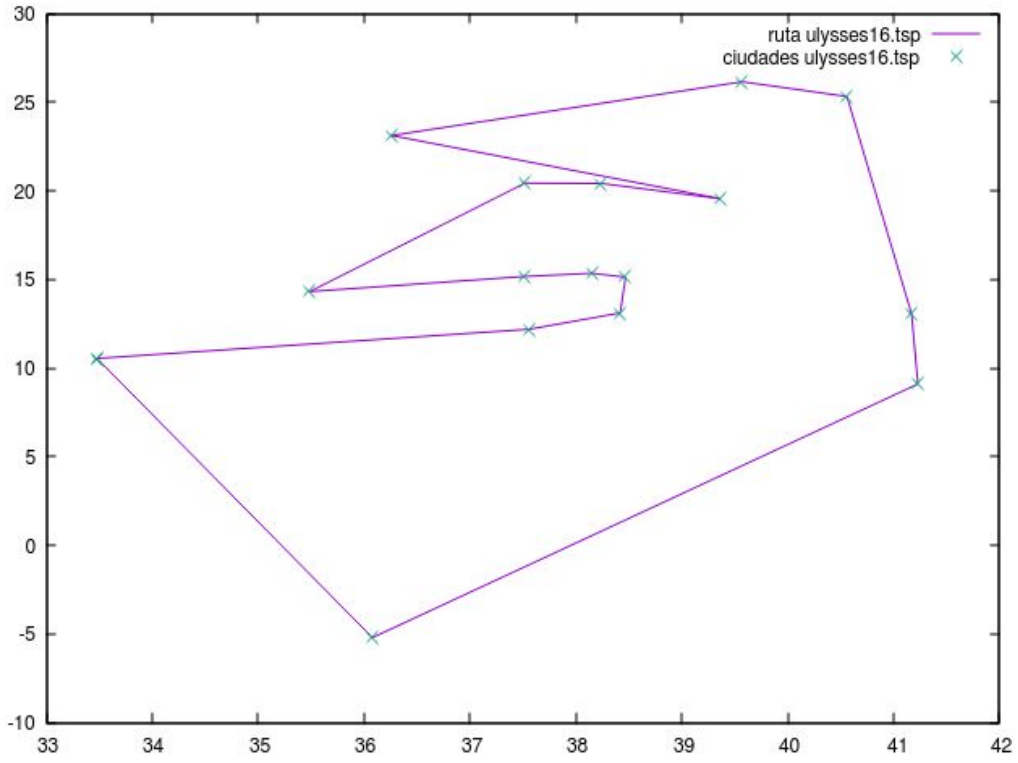


tsp225_2-opt

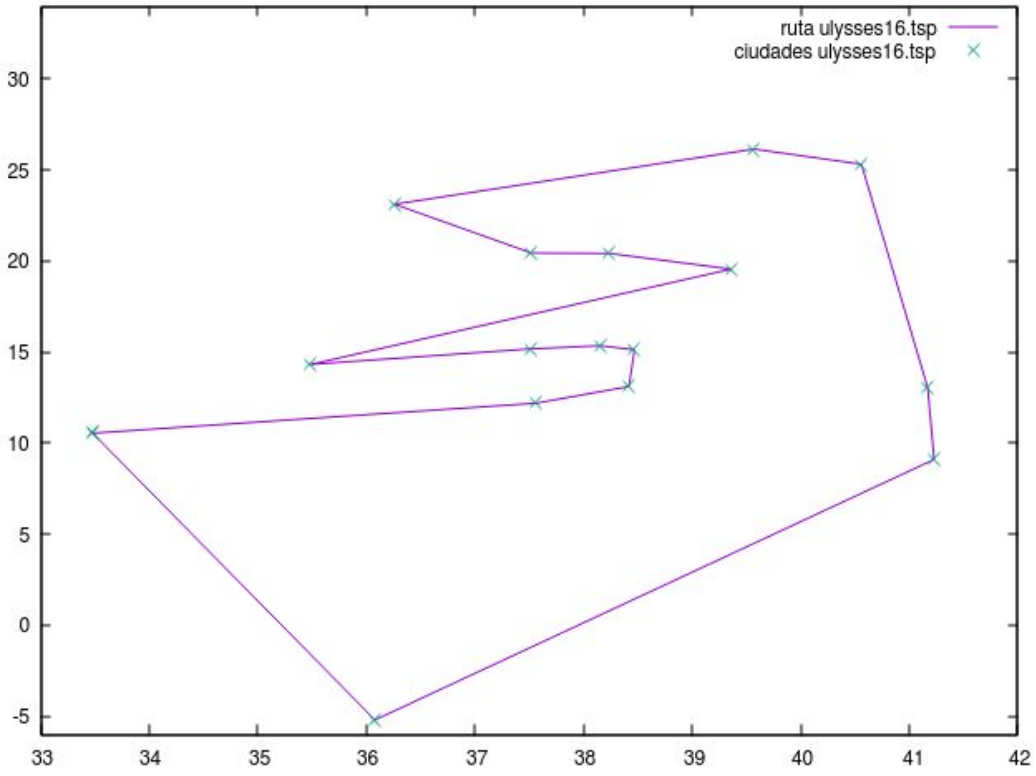


tsp225_insercion

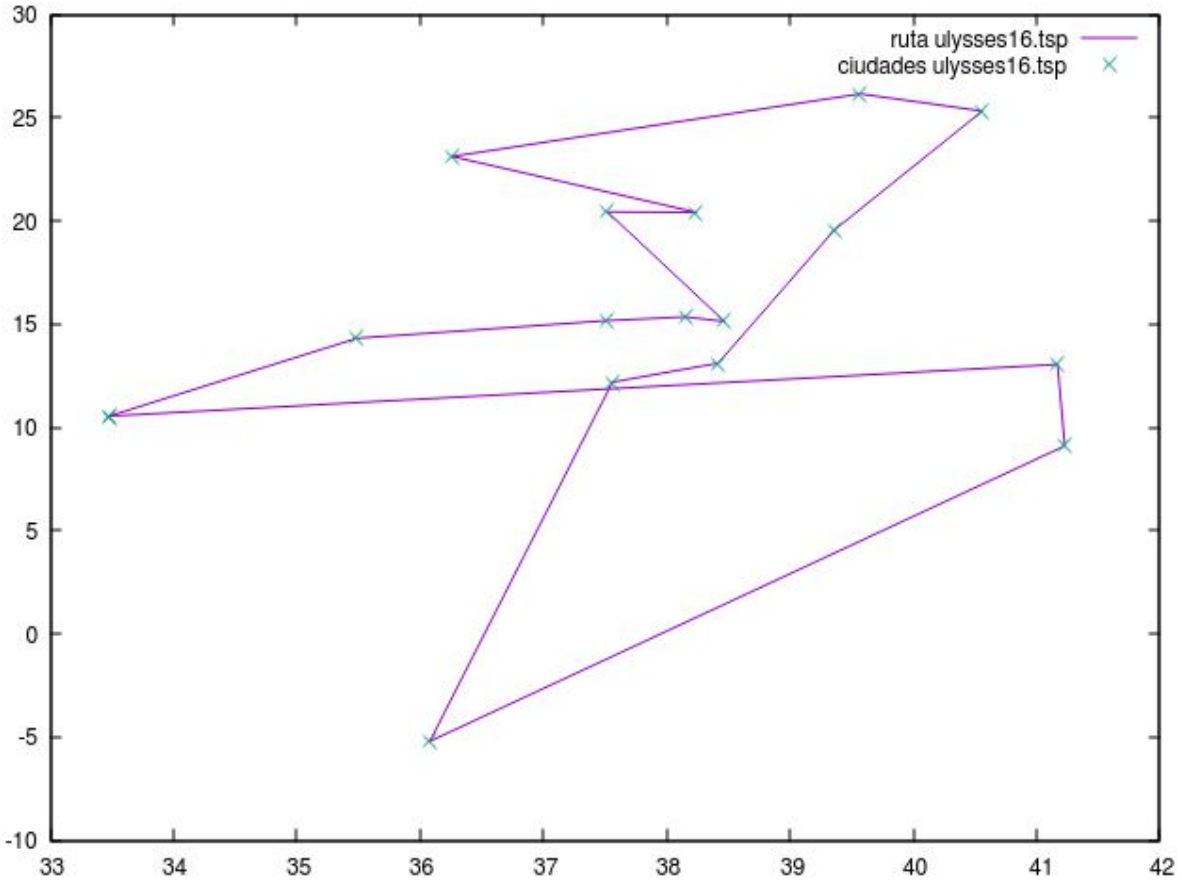
25. ulysses16



ulysses16_vmc

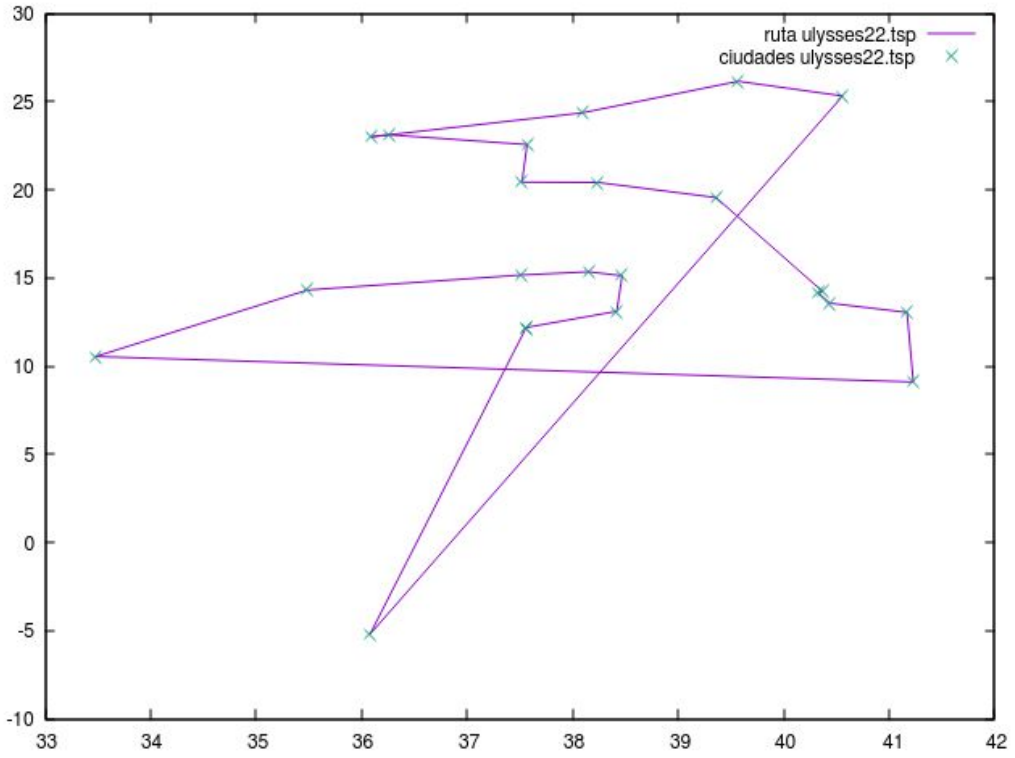


ulysses16_2-opt

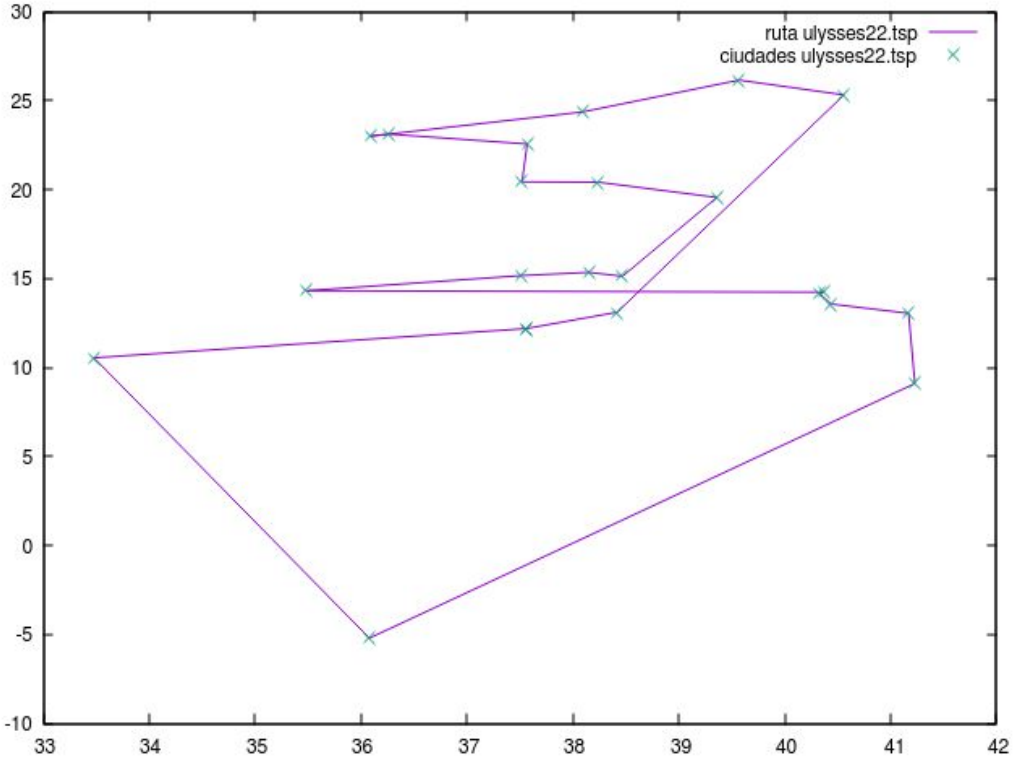


ulysses16_insercion

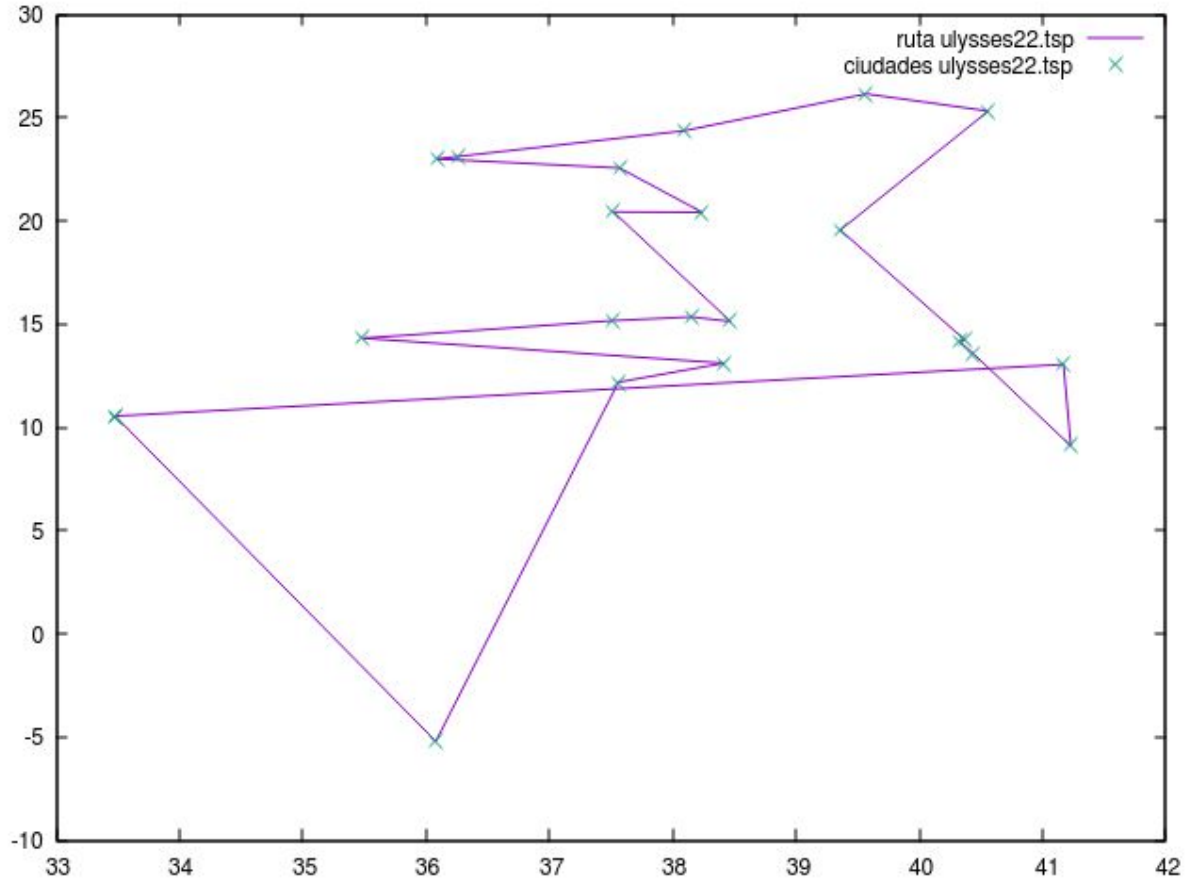
26. ulysses22



ulysses22_vmc



ulysses22_2-opt



ulysses22_insercion

Como se puede ver en todas estas gráficas, el método por inserción es menos eficaz a la hora de buscar una ruta óptima que una heurística por el vecino más cercano.

También se puede observar que con la mejora 2-opt, mejora significativamente la solución propuesta por el vecino más cercano, ya que evita que el viajero pase por el mismo sitio más de una vez, lo que significa una distancia menor (salvo que no haya más remedio que pasar por el mismo sitio otra vez, como por ejemplo, la vuelta a la ciudad de origen)

6. Tabla de distancias totales solución

Archivo	VMC	Inserción	Mejora VMC 2-opt
a280	2984.92	3223.97	2772.64
att48	37900.9	42370.2	35322.3
bayg29	9964.78	11098.7	9490.34
berlin52	8182.19	9622.17	8156.97
ch130	7131.31	7499.39	6647.44
ch150	7118.91	8382.48	6814.53
eil51	483.381	539.222	471.172
eil76	617.186	648.018	585.615
eil101	756.616	800.561	683.226
gr96	608.781	641.986	562.138
gr120	1837.44	1917.11	1764.06
gr202	597.915	589.33	557.339
gr666	3837.37	3792.73	3548.22
kroA100	24698.5	27646.1	23088.4
kroC100	23662.2	27141.4	22140.3
kroD100	24855.8	28499.9	23942.9
lin105	16939.4	19159.3	15865.4
pa561	17950.9	17761.6	16444
pcb442	58953	61217.7	56012.9
pr76	130921	137046	118094
pr1002	313765	316166	289744
rd100	9427.33	9747.94	8593.53
st70	802.53	828.336	732.239
tsp225	4539.54	4768.04	4275
ulysses16	77.4131	81.1402	75.7203
ulysses22	89.8785	85.6016	79.3175