Universidad Nacional de San Agustín

Maestría en Ciencias de la Computación

Algoritmos y Estructura de datos

Grafos y algoritmos de búsqueda

Problema del agente viajero con aplicación del algoritmo del vecino más cercano

Presentado por el Grupo 7, cuyo integrantes son:
Ccallo Fernandez, Adolfo
del Carpio Maraza, Heberth Enrique
Vargas Franco, Mauricio Sebastian

1. Introducción

En este presentable de la practica grupal, proponemos resolver el problema del vendedor ambulante con el algoritmo de vecino más próximo. Que según lo investigado fue uno de los primeros algoritmos utilizados para dar respuesta a este problema, que a pesar de generar una solución rápida, pero que no es la ideal. Escogimos el algoritmo del vecino más proximo porque es fácil de implementar y se ejecuta rápidamente, pero que puede darnos rutas no optimas, que nosostros a simple vista podríamos hallarlas rapidamente.

2. Marco teórico

2.1. El problema del vendedor viajero

> "Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?"

El origen del enunciado anterior, al que llamamos, El "Problema del vendedor viajero", no está claro. Una guía para viajeros de 1832 menciona el problema e incluye ejemplos de viajes a través de Alemania y Suiza, pero no contiene un tratamiento matemático del mismo.

El problema del viajante fue definido en los años 1800s por el matemático irlandés W. R. Hamilton y por el matemático británico Thomas Kirkman. El Juego Icosian de Hamilton fue un puzle recreativo basado en encontrar un ciclo de Hamilton. Todo parece indicar que la forma general del problema del vendedor viajero fue estudiada, por primera vez por matemáticos en Viena y Harvard, durante los años 1930s. Destacándose Karl Menger, quien definió los problemas, considerando el obvio algoritmo de fuerza bruta, y observando la no optimalidad de la heurística de vecinos más cercanos.

EL analisis para dar respuesta a este problema (que es considerado como el mas estudiado del mundo), es encontrar, para un conjunto finito de puntos, de los cuales se conocen las distancias entre cada par, el camino más corto entre estos puntos. Por supuesto, el problema es resuelto por un conjunto finito de intentos. La regla que se debe seguir es que desde el punto inicial se va al punto más cercano a este, de ahí a su más cercano y así sucesivamente, en general este algoritmo no retornaba la ruta más corta. Hassler Whitney de la Universidad de Princeton introdujo el nombre "Travelling Salesman Problem" poco después, de alli el término TSP y de aqui en adelante así lo denominaremos en este trabajo.

2.1.1. Características de TSP

TSP se encuentra clasificado como Problema de optimización Combinatoria, es decir, es un problema donde intervienen cierto número de variables donde cada variable puede tener N diferentes valores y cuyo número de combinaciones es de carácter exponencial, lo que da lugar a múltiples soluciones óptimas (soluciones que se calculan en un tiempo finito) para una instancia.

TSP es un problema considerado difícil de resolver, denominándose en lenguaje computacional NP-Completo, es decir, es un problema para el que no podemos garantizar que se encontrará la mejor solución en un tiempo de cómputo razonable. Para dar solución se emplean diferentes métodos, entre los cuales, los principales se denominan heurísticas cuyo objetivo es generar soluciones de buena calidad en tiempos de cómputo mucho más pequeños.

Las variables anteriores han sido empleadas por diferentes investigadores en los siguientes campo en otras varias:

- Tiempo de recorrido entre ciudades: horas, minutos, días, semanas, etc.
- Distancia de recorrido entre ciudades: metros, kilómetros, millas, milímetros, etc.
- Costo de traslado: dinero, desgaste de las piezas, gasto de energía, etc.

TSP a sido adaptada para solucionar problemas similares por ejemplo:

- Circuitos electrónicos: cantidad de soldadura utilizada, menor espacio entre los puntos de soldadura de los circuitos, evitar el cruce entre las líneas de soldadura, tiempo de fabricación, distribución de los circuitos, entre otras.
- Control de semáforos: Número de semáforos (nodos), tiempo de traslado entre semáforos, cantidad de autos que pasan por un punto, entre otras variables.
- Previsión del tránsito terrestre: puntos en una ciudad, cantidad de vehículos, tiempo de traslado, tipos de vehículos, horas pico, correlación entre variables, regresión lineal, etc.
- Entrega de productos: Peso de las entregas, número de entregas, nodos (domicilios) a visitar, recorridos, tiempos de traslado, tipo de vehículo, etc.
- Estaciones de trabajo: secuencia de actividades, lugar de las herramientas (nodos), Tipo de herramientas, tiempo de uso, etc. Edificación: puntos de edificación (construcciones), distancia entre las construcciones y

los insumos, vehículos (grúas, camiones de volteo, etc.), cantidad de combustible que emplean, etc.

2.1.2. Aplicaciónes en el mundo real de TSP

TSP se puede emplear en cualquier situación que requiere seleccionar nodos en cierto orden que reduzca los costos:

- Reparto de productos. Mejorar una ruta de entrega para seguir la más corta.
- Transporte. Mejorar el recorrido de caminos buscando la menor longitud.
- Robótica. Resolver problemas de fabricación para minimizar el número de desplazamientos al realizar una serie de perforaciones en un circuito impreso.
- Turismo y agencias de viajes. Aun cuando los agentes de viajes no tienen un conocimiento explícito del Problema del Agente Viajero, las compañías dedicadas a este giro utilizan un software que hace todo el trabajo.
- Horarios de transportes laborales y/o escolares. Estandarizar los horarios de los transportes es claramente una de sus aplicaciones, tanto que existen empresas que se especializan en ayudar a las escuelas a programarlos para optimizarlos en base a una solución del TSP.
- Inspecciones a sitios remotos. Ordenar los lugares que deberá visitar un inspector en el menor tiempo.
- Secuencias. Se refiere al orden en el cual n trabajos tienen que ser procesados de tal forma que se minimice el costo total de producción.

2.2. Algoritmo del vecino más cercano

El algoritmo de k vecinos más cercanos, también conocido como KNN o k-NN, es un clasificador de aprendizaje supervisado no paramétrico, que utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de un punto de datos individual.

Si bien se puede usar para problemas de regresión o clasificación, generalmente se usa como un algoritmo de clasificación, partiendo de la suposición de que se pueden encontrar puntos similares cerca uno del otro.

Para los problemas de clasificación, se asigna una etiqueta de clase sobre la base de un voto mayoritario, es decir, se utiliza la etiqueta que se representa con más frecuencia alrededor de un punto de datos determinado. Si bien esto técnicamente se considera "voto por mayoría", el término "voto por mayoría"se usa más comúnmente en la literatura. La distinción entre estas terminologías es que "voto mayoritario"técnicamente requiere una mayoría superior al 50 %, lo que funciona principalmente cuando solo hay dos categorías. Cuando tiene varias clases, por ejemplo, cuatro categorías, no necesita necesariamente el 50 % de los votos para llegar a una conclusión sobre una clase; puede asignar una etiqueta de clase con un voto superior al 25 %.

Los problemas de regresión usan un concepto similar al de los problemas de clasificación, pero en este caso, se toma el promedio de los k vecinos más cercanos para hacer una predicción sobre una clasificación. La distinción principal aquí es que la clasificación se usa para valores discretos, mientras que la regresión se usa para valores continuos. Sin embargo, antes de que se pueda hacer una clasificación, se debe definir la distancia. La distancia euclidiana es la más utilizada, y nos profundizaremos más a continuación.

También vale la pena señalar que el algoritmo KNN también forma parte de una familia de modelos de aprendizaje perezoso, lo que significa que solo almacena un conjunto de datos de entrenamiento en lugar de pasar por una etapa de entrenamiento. Esto también significa que todo el cálculo ocurre cuando se realiza una clasificación o predicción. Dado que depende en gran medida de la memoria para almacenar todos sus datos de entrenamiento, también se lo denomina método de aprendizaje basado en instancias o basado en la memoria.

A Evelyn Fix y Joseph Hodges se les atribuyen las ideas iniciales en torno al modelo KNN en este. artículo de 1951 mientras que Thomas Cover amplía su concepto en su investigación, çlasificación de patrones de vecinos más cercanos ". Si bien no es tan popular como lo fue antes, sigue siendo uno de los primeros algoritmos que uno aprende en la ciencia de datos debido a su simplicidad y precisión. Sin embargo, a medida que crece un conjunto de datos, KNN se vuelve cada vez más ineficiente, lo que compromete el rendimiento general del modelo. Se usa comúnmente para sistemas de recomendación simples, reconocimiento de patrones, extracción de datos, predicciones del mercado financiero, detección de intrusos y más.

2.2.1. Cálculo del vecino KNN

En resumen, el objetivo del algoritmo del vecino más cercano es identificar los vecinos más cercanos de un punto de consulta dado, de modo que podamos asignar una etiqueta de clase a ese punto. Para hacer esto, KNN tiene algunos

requisitos:

Determinar sus métricas de distancia Para determinar qué puntos de datos están más cerca de un punto de consulta determinado, será necesario calcular la distancia entre el punto de consulta y los otros puntos de datos. Estas métricas de distancia ayudan a formar límites de decisión, que dividen los puntos de consulta en diferentes regiones. Por lo general, verá límites de decisión visualizados con diagramas de Voronoi.

Si bien hay varias medidas de distancia entre las que puede elegir, este trabajo solo cubrirá lo siguiente y explicará brevemente otras dos:

- **Distancia euclidiana (p=2):** Esta es la medida de distancia más utilizada y está limitada a vectores de valor real. Usando la fórmula a continuación, mide una línea recta entre el punto de consulta y el otro punto que se mide.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

- **Distancia Manhattan (p=1):** Esta es también otra métrica de distancia popular, que mide el valor absoluto entre dos puntos. También se conoce como distancia de taxi o distancia de cuadra de la ciudad, ya que comúnmente se visualiza con una cuadrícula, que ilustra cómo se puede navegar de una dirección a otra a través de las calles de la ciudad.

$$d(x,y) = (\sum_{i=1}^{m}|(x_i - y_i)|)$$

- **Distancia minkowski:** Esta medida de distancia es la forma generalizada de las métricas de distancia Euclidiana y Manhattan. El parámetro, p, en la fórmula a continuación, permite la creación de otras métricas de distancia. La distancia euclidiana se representa mediante esta fórmula cuando p es igual a dos, y la distancia de Manhattan se denota con p igual a uno.

$$d(x,y) = (\sum_{i=1}^{n}|(x_i - y_i)|^{1/p})$$

2.3. Aplicaciones del algoritmo vecinos cercanos

- **Preprocesamiento de datos:** Los conjuntos de datos suelen tener valores faltantes, pero el algoritmo KNN puede estimar esos valores en un proceso conocido como imputación de datos faltantes.

- **Motores de recomendación :** utilizando datos de flujo de clics de sitios web, el algoritmo KNN se ha utilizado para proporcionar recomendaciones automáticas a los usuarios sobre contenido adicional. Muestra que un usuario está asignado a un grupo en particular y, en función del comportamiento del usuario de ese grupo, se le da una recomendación. Sin embargo, dados los problemas de escala con KNN, este enfoque puede no ser óptimo para conjuntos de datos más grandes.

- **Finanzas:** También se ha utilizado en una variedad de casos de uso económico y financiero. Por ejemplo muestra cómo el uso de KNN en datos crediticios puede ayudar a los bancos a evaluar el riesgo de un préstamo para una organización o individuo. Se utiliza para determinar la solvencia crediticia de un solicitante de préstamo. Otro periódico destaca su uso en la previsión del mercado de valores, valores de cambio de divisas, comercio de futuros y análisis de lavado de dinero.

- **Cuidado de la salud:** KNN se ha aplicado dentro de la industria de la salud, haciendo predicciones sobre el riesgo de ataques cardíacos y cáncer de próstata. El algoritmo funciona calculando las expresiones genéticas más probables.

- **Reconocimiento de patrones:** KNN también ha ayudado a identificar patrones, como en texto y clasificación de dígitos. Esto ha sido particularmente útil para identificar números escritos a mano que puede encontrar en formularios o sobres de correo.

2.4. Ventajas y desventajas del algoritmo KNN

Al igual que cualquier algoritmo de machine learning, k-NN tiene sus puntos fuertes y débiles. Dependiendo del proyecto y la aplicación, puede o no ser la elección correcta.

Ventajas

- **Fácil de implementar:** Dada la simplicidad y precisión del algoritmo, es uno de los primeros clasificadores que aprenderá un nuevo científico de datos.

- **Se adapta fácilmente:** A medida que se agregan nuevas muestras de entrenamiento, el algoritmo se ajusta para tener en cuenta cualquier dato nuevo, ya que todos los datos de entrenamiento se almacenan en la memoria.
- **Pocos hiperparámetros:** KNN solo requiere un valor k y una métrica de distancia, que es baja en comparación con otros algoritmos de machine learning.

Desventajas

- **No escala bien:** Dado que KNN es un algoritmo perezoso, ocupa más memoria y almacenamiento de datos en comparación con otros clasificadores. Esto puede ser costoso desde una perspectiva de tiempo y dinero. Más memoria y almacenamiento aumentarán los gastos comerciales y más datos pueden tardar más en procesarse. Si bien se han creado diferentes estructuras de datos, como Ball-Tree, para abordar las ineficiencias computacionales, un clasificador diferente puede ser ideal según el problema comercial.
- **La maldición de la dimensionalidad:** El algoritmo KNN tiende a ser víctima de la maldición de la dimensionalidad, lo que significa que no funciona bien con entradas de datos de alta dimensión. Esto a veces también se conoce como fenómeno de pico (PDF, 340 MB) (enlace externo a ibm.com), donde después de que el algoritmo alcanza la cantidad óptima de funciones, las funciones adicionales aumentan la cantidad de errores de clasificación, especialmente cuando el tamaño de la muestra es más pequeño.
- **Propenso al sobreajuste:** Debido a la "maldición de la dimensionalidad", KNN también es más propenso al sobreajuste. Si bien se aprovechan las técnicas de selección de características y reducción de dimensionalidad para evitar que esto ocurra, el valor de k también puede afectar el comportamiento del modelo. Los valores más bajos de k pueden sobreajustar los datos, mientras que los valores más altos de k tienden a "suavizar"los valores de predicción, ya que están promediando los valores en un área o vecindario más grande. Sin embargo, si el valor de k es demasiado alto, entonces puede ajustarse mal a los datos.

3. Experimento

En nuestra implementacion del algoritmo genético para la solución del TSP, las ciudades se toman como genes, la cadena generada usando estos caracteres se llama cromosoma, mientras que una puntuación de aptitud que es igual a la longitud del camino de todas las ciudades mencionadas, se utiliza para dirigirse a una población.

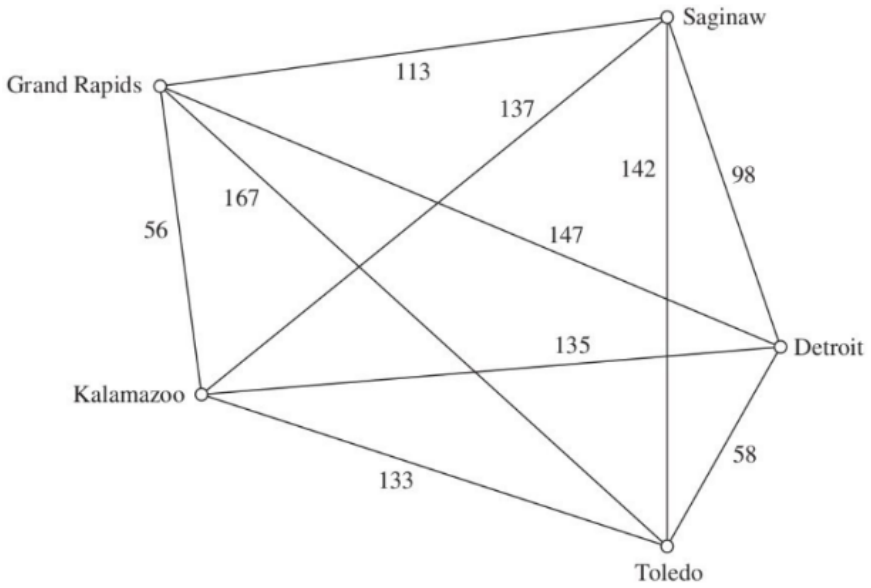


Figura 1: Grafodel problema propuesto.

3.1. Seudocódigo

```
COMIENZO
        Entrada: D = {(x1, c1), . . . ,(xN , cN )}
                 x = (x1, . . . , xn) nuevo caso a clasificar
        PARA todo objeto ya clasificado (xi , ci)
                 calcular di = d(xi , x)
        Ordenar di(i = 1, . . . , N) en orden ascendente
        Quedarnos con los K casos D(x,k) ya clasificados mas
            cercanos a x
        Asignar a x la clase mas frecuente en D(x,k)
FIN
```

3.2. Algoritmo

Link al código: GITHUB.

Función para encontrar la city no visitada mas cercana

```
1         def nearest_neighbor(curr_city, unvisited_cities,
              distance_matrix):
2                 min_distance = float('inf')
3                 nearest_city = None
4                 for city in unvisited_cities:
5                         dist = distance_matrix[curr_city][city]
6                         if dist < min_distance:
7                                 min_distance = dist
8                                 nearest_city = city
9         return nearest_city, min_distance
```

Función con el algoritmo vecinos más cercanos

```
1         def nearest_neighbor_algorithm(distance_matrix):
2                 num_cities = len(distance_matrix)
3                 visited = [False] * num_cities
4                 tour = []
5                 total_distance = 0
6                 current_city = 0
7                 tour.append(current_city)
8                 visited[current_city] = True
9                 for _ in range(num_cities - 1):
10                        next_city, dist = nearest_neighbor(
                              current_city, [city for city in range
                              (num_cities) if not visited[city]],
                              distance_matrix)
11                        total_distance += dist
12                        current_city = next_city
13                        tour.append(current_city)
14                        visited[current_city] = True
15                total_distance += distance_matrix[tour[-1]][tour
                      [0]]
16        return tour, total_distance
```

Función que calcula la distancia Euclidiana

```
1         def generate_distance_matrix(num_cities):
2                 np.random.seed(42)
```

```
3                 distances = np.random.randint(10, 100, size=(
                      num_cities, num_cities))
4                 np.fill_diagonal(distances, 0)
5         return distances
```

3.3. Pruebas

Las pruebas se realizaron con diferentes numero de puntos (cities)

3.3.1. Prueba con el problema propuesto

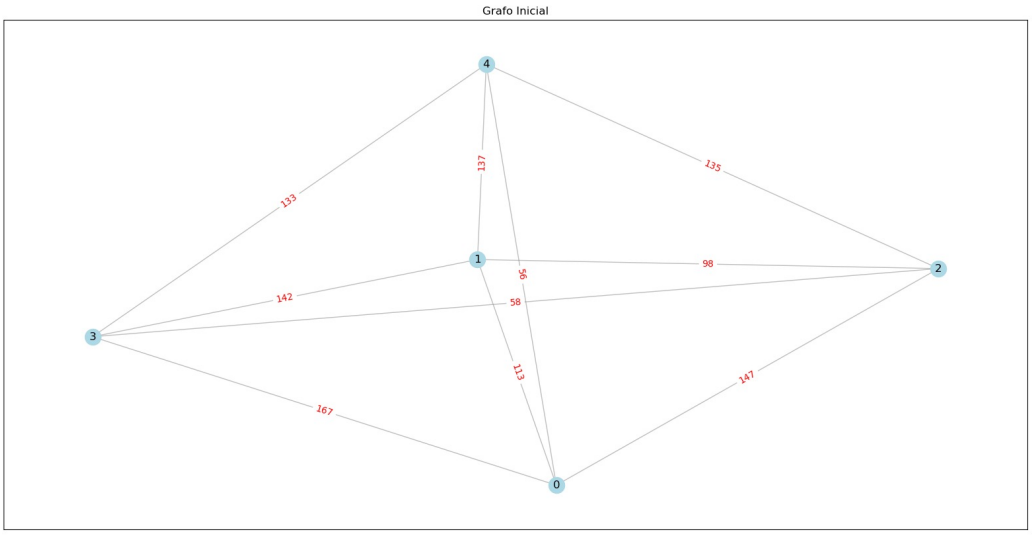


Figura 2: Grafo del problema propuesto con 5 cities.

Tabla de resultados del problema propuesto

Runs	Tiempo	Ruta	Coste
1	0.000132299959659576	[0, 4, 3, 2, 1]	458
2	0.000042900035623461	[0, 4, 3, 2, 1]	458
3	4.33999812230467E-05	[0, 4, 3, 2, 1]	458
4	4.24000318162143E-05	[0, 4, 3, 2, 1]	458
5	4.20000287704169E-05	[0, 4, 3, 2, 1]	458
6	4.07000188715755E-05	[0, 4, 3, 2, 1]	458
7	4.09999629482626E-05	[0, 4, 3, 2, 1]	458
8	4.08999621868133E-05	[0, 4, 3, 2, 1]	458
9	4.11000219173729E-05	[0, 4, 3, 2, 1]	458
10	4.02000150643289E-05	[0, 4, 3, 2, 1]	458
Promedio	5.06900018081068E-05	[0, 4, 3, 2, 1]	458

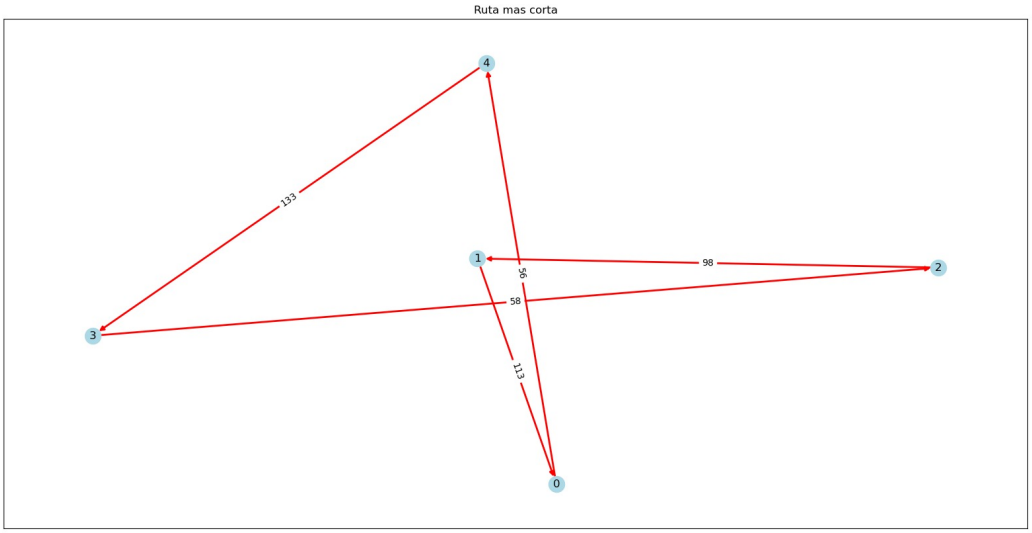


Figura 3: Grafo con la ruta mas corta del problema propuesto con 5 cities.

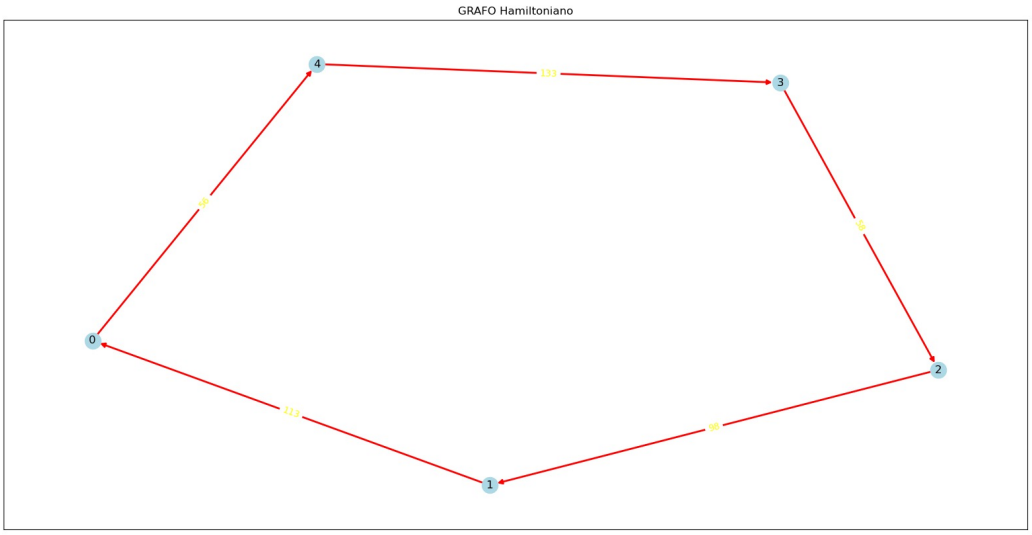
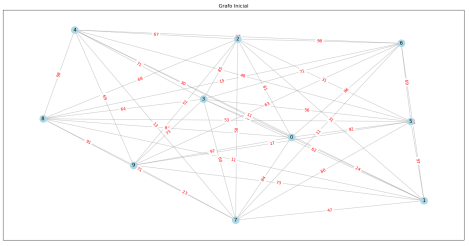


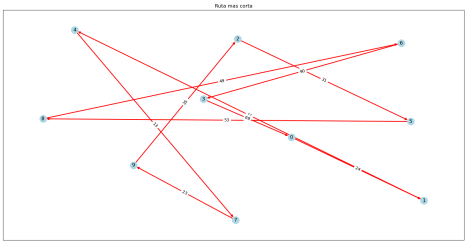
Figura 4: Ciclo Hamiltoniano del problema propuesto para 5 cities.

3.3.2. Otras pruebas realizadsa con diferente numero de cities

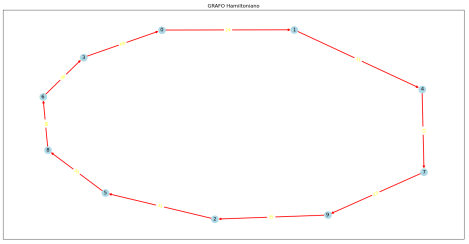
Pruebas con 10 Cities.



(a) Grafo 10 cities

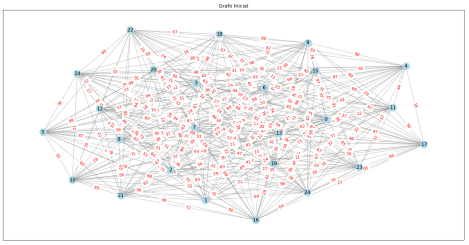


(b) Ruta 10 cities

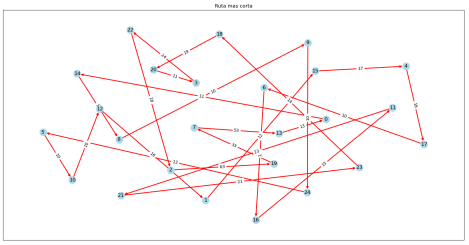


(c) Ciclo 10 cities

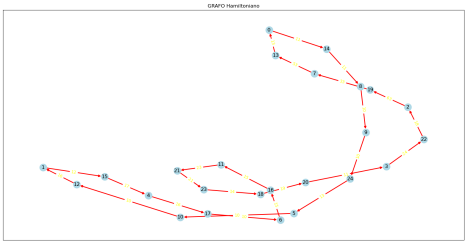
Pruebas con 25 Cities.



(a) Grafo 25 cities

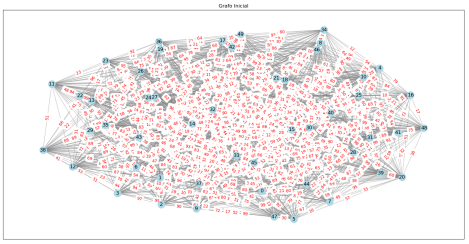


(b) Ruta 25 cities

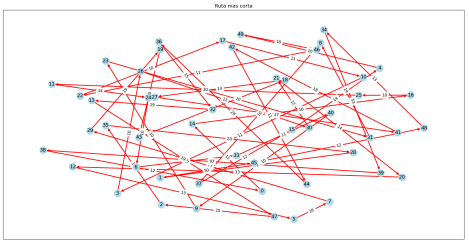


(c) Ciclo 25 cities

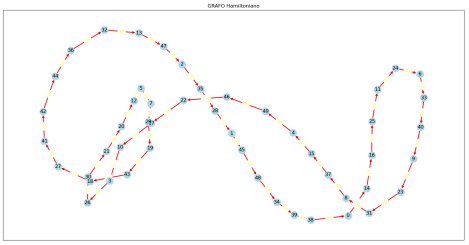
Pruebas con 50 Cities.



(a) Grafo 50 cities

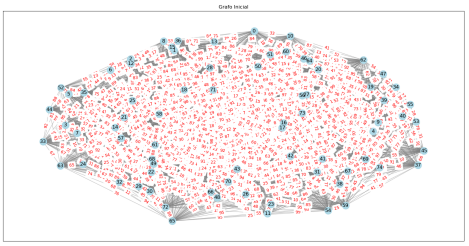


(b) Ruta 50 cities

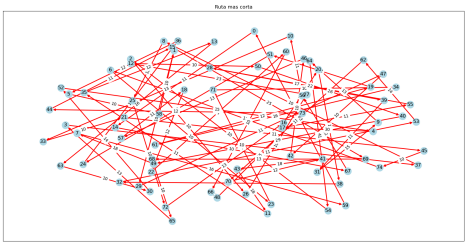


(c) Ciclo 50 cities

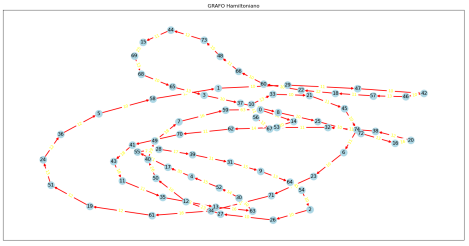
Pruebas con 75 Cities.



(a) Grafo 75 cities



(b) Ruta 75 cities



(c) Ciclo 75 cities

Prueba con 100 puntos

Runs	Tiempo	Ruta	Coste
1	0.00332409999100491	[... 100 ...]	990
2	0.00334450002992525	[... 100 ...]	990
3	0.00272879999829456	[... 100 ...]	990
4	0.00252209999598562	[... 100 ...]	990
5	0.00248580001061782	[... 100 ...]	990
6	0.00252830004319548	[... 100 ...]	990
7	0.00252600002568215	[... 100 ...]	990
8	0.00268410000717267	[... 100 ...]	990
9	0.00252390000969171	[... 100 ...]	990
10	0.00252390000969171	[... 100 ...]	990
Promedio	0.00271915001212619	[... 100 ...]	990

Prueba con 300 puntos

Runs	Tiempo	Ruta	Coste
1	0.0189089999	[... 300 ...]	1395
2	0.01750509	[... 300 ...]	1395
3	0.0176135	[... 300 ...]	1395
4	0.0174149999	[... 300 ...]	1395
5	0.0175668	[... 300 ...]	1395
6	0.01752529	[... 300 ...]	1395
7	0.017733	[... 300 ...]	1395
8	0.0176591	[... 300 ...]	1395
9	0.0174292	[... 300 ...]	1395
10	0.0175542	[... 300 ...]	1395
Promedio	0.01769101798	[... 300 ...]	1395

Prueba con 800 puntos

Runs	Tiempo	Ruta	Coste
1	0.122537	[... 800 ...]	1970
2	0.117908	[... 800 ...]	1970
3	0.117908	[... 800 ...]	1970
4	0.118833	[... 800 ...]	1970
5	0.1215618	[... 800 ...]	1970
6	0.1182243	[... 800 ...]	1970
7	0.120469	[... 800 ...]	1970
8	0.118455	[... 800 ...]	1970
9	0.1219938999	[... 800 ...]	1970
10	0.1184463	[... 800 ...]	1970
Promedio	0.11963362999	[... 800 ...]	1970

Prueba con 1500 puntos

Runs	Tiempo	Ruta	Coste
1	0.4095712	[... 1500 ...]	2569
2	0.4084163	[... 1500 ...]	2569
3	0.406634	[... 1500 ...]	2569
4	0.4060298	[... 1500 ...]	2569
5	0.4100123	[... 1500 ...]	2569
6	0.4086348	[... 1500 ...]	2569
7	0.4056693	[... 1500 ...]	2569
8	0.40923389	[... 1500 ...]	2569
9	0.4073833	[... 1500 ...]	2569
10	0.4079456999	[... 1500 ...]	2569
Promedio	0.40795305899	[... 1500 ...]	2569

Prueba con 3000 puntos

Runs	Tiempo	Ruta	Coste
1	1.6119452	[... 3000 ...]	3956
2	1.6084598	[... 3000 ...]	3956
3	1.617657799	[... 3000 ...]	3956
4	1.6028205	[... 3000 ...]	3956
5	1.61371919	[... 3000 ...]	3956
6	1.6088568	[... 3000 ...]	3956
7	1.6149575	[... 3000 ...]	3956
8	1.608659	[... 3000 ...]	3956
9	1.628781	[... 3000 ...]	3956
10	1.6211127	[... 3000 ...]	3956
Promedio	1.6136969489	[... 3000 ...]	3956

Prueba con 5000 puntos

Runs	Tiempo	Ruta	Coste
1	4.42456509	[... 5000 ...]	6101
2	4.4361885	[... 5000 ...]	6101
3	4.4278524	[... 5000 ...]	6101
4	4.426887	[... 5000 ...]	6101
5	4.44922	[... 5000 ...]	6101
6	4.443473	[... 5000 ...]	6101
7	4.433249	[... 5000 ...]	6101
8	4.44164129	[... 5000 ...]	6101
9	4.436706	[... 5000 ...]	6101
10	4.4397741	[... 5000 ...]	6101
Promedio	4.435955638	[... 5000 ...]	6101

Prueba con 15000 puntos

Runs	Tiempo	Ruta	Coste
1	39.32168	[... 15000 ...]	16035
2	39.484393	[... 15000 ...]	16035
3	39.26685	[... 15000 ...]	16035
4	39.11558	[... 15000 ...]	16035
5	39.07749	[... 15000 ...]	16035
6	39.04824	[... 15000 ...]	16035
7	39.31569519	[... 15000 ...]	16035
8	38.9629	[... 15000 ...]	16035
9	38.923053	[... 15000 ...]	16035
10	38.845817	[... 15000 ...]	16035
Promedio	39.136169819	[... 15000 ...]	16035

Prueba con 25000 puntos

Runs	Tiempo	Ruta	Coste
1	112.053588	[... 25000 ...]	26311
2	113.47157	[... 25000 ...]	26311
3	114.745901	[... 25000 ...]	26311
4	114.698304	[... 25000 ...]	26311
5	112.8187495	[... 25000 ...]	26311
6	114.165464	[... 25000 ...]	26311
7	114.51515	[... 25000 ...]	26311
8	114.5296375	[... 25000 ...]	26311
9	113.08011	[... 25000 ...]	26311
10	113.4107259999	[... 25000 ...]	26311
Promedio	113.74891999999	[... 25000 ...]	26311

4. Conclusiones

- Nuestro aplicativo en Python usando el algoritmo corre relativamente rápido, pero con 30,000 cities crashea nuestro código.
- Entre más ciudades se agregue, más lento es correr el algoritmo, exponencial.
- El Algoritmo Vecinos Cercanos, es un algoritmo sencillo de implementar y relativamente eficiente, se recomienda para casos de rápida implementación, pero si se desea mejores resultados euristicos sería recomendable usar algoritmos genéticos, algoritmos de enjambre de partículas, etc.