Algoritmo de Kruskal

En el Problema del Agente Viajero - TSP (Travelling Salesman Problem), el objetivo es encontrar un recorrido completo que conecte todos los nodos de una red, visitándolos tan solo una vez y volviendo al punto de partida, y que además minimice la distancia total de la ruta.
Este tipo de problemas tiene gran aplicación en el ámbito de la logística y distribución, así como en la programación de curvas de producción.

La cantidad de rutas posibles en una red está determinada por la ecuación:

(n-1)!

Su popularidad se debe a que es fácil de plantear, pero difícil de resolver.

Se puede describir de la siguiente forma: Dadas *n* ciudades y el costo *Cij* que se tiene al viajar de una ciudad a otra, se debe encontrar la ruta de costo mínimo para visitarlas todas pasando sólo una vez por cada una de ellas, y regresando a la de partida. A cada ruta se le llama *tour o ciclo* hamiltoniano.

El Problema del Agente Viajero puede resolverse de diferentes maneras:

- Enumeración de todas las soluciones factibles. Es decir, enlistar todas las posibles soluciones al problema, calcular sus costos asociados, e identificar, por comparación, cuál es la solución con el costo más conveniente.

- Métodos exactos. También llamados algoritmos óptimos, intentan descartar familias enteras de posibles soluciones, tratando así de acelerar la búsqueda y

llegar a una óptima. Los que más se usan para resolver el TSP son Ramificación y Acotamiento, y Ramificación y Corte.

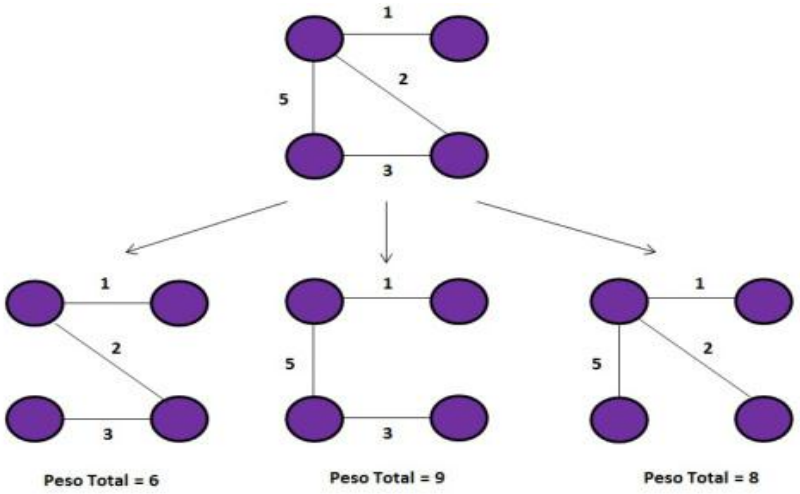
- Heurísticas. Son métodos obtienen buenas soluciones en tiempos de cómputo muy cortos, aunque sin garantizar la optimalidad de la solución.

Porque el problema del agente viajero es difícil

Resulta difícil debido a que actualmente no existe un algoritmo que muestre un resultado exacto para dicho problema el cual tienen que ser extensamente estudiado debido al grado de complejidad y posibles soluciones que son arrojadas cuando esta en consideración la gran cantidad de rutas. Es decir, es un problema para el que no podemos garantizar que se encontrará la mejor solución en un tiempo de cómputo razonable, por esto, cuando una instancia de grandes dimensiones se resuelve con algún método exacto, toma un extenso periodo de tiempo. Con el uso de heurísticas se tienen soluciones de muy buena calidad en tiempos de cómputo mucho más pequeño

Árbol de Expansión Mínima

Dado un grafo conexo, no dirigido y con pesos en las aristas, un árbol de expansión mínima es un árbol compuesto por todos los vértices y cuya suma de sus aristas es la de menor peso. Al ejemplo anterior le agregamos pesos a sus aristas y obtenemos los arboles de expansiones siguientes:



De la imagen anterior el árbol de expansión mínima sería el primer árbol de expansión cuyo peso total es 6.

El problema de hallar el Árbol de Expansión Mínima (MST) puede ser resuelto con varios algoritmos, los más conocidos con Prim y Kruskal ambos usan técnicas voraces

El algoritmo de kruskal

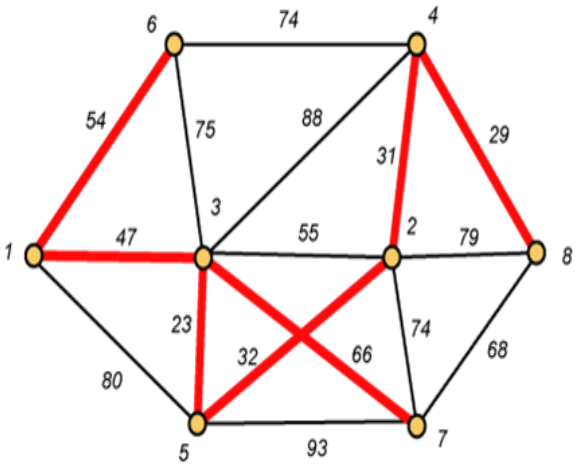
Es un algoritmo que va unir todos los nodos formando un árbol, tomando las aristas que tengan un peso que siempre sea menor. Este árbol (spanning tree) de un grafo es un subgrafo que contiene todos sus vértices o nodos. Un grafo puede tener múltiples árboles. Por ejemplo, un grafo completo de cuatro nodos (todos relacionados con todos) tendría 16 árboles.

El algoritmo se basa en una propiedad clave de los árboles que permite estar seguros de sí un arco debe pertenecer al árbol o no, y usar esta propiedad para seleccionar cada arco. Nótese en el algoritmo, que siempre que se añade un arco (u,v), éste será siempre la conexión más corta (menor coste) alcanzable desde el nodo u al resto del grafo G. Así que por definición éste deberá ser parte del árbol. Este algoritmo es de tipo voraz (también llamado greedy) ya que, a cada paso, éste selecciona el arco más barato y lo añade al subgrafo. Este tipo de algoritmos pueden no funcionar para resolver otro tipo de problemas, por ejemplo, para encontrar la ruta más corta entre los nodos a y b.

Características

- Algoritmo basado en las aristas
- Agregar las aristas, uno a la vez, en orden de peso creciente
- El algoritmo mantiene A – un bosque de árboles. Una arista es aceptada si se conecta vértices de distintos árboles
- Necesitamos una estructura de datos que mantiene una partición, es decir, a colección de conjuntos disjuntos

A continuación, se muestra en la siguiente figura un ejemplo del uso de kruskal donde va seleccionado cuidadosamente cada vecino con la condición de que sea el más corto o menor



Finalmente proyecta cual fue la mejor ruta después de haber estudiado todas las posibles

Código en Python con el algoritmo kruskal

```
def kruskal(self):
    e = deepcopy(self.E)
    arbol = Grafo()
    peso = 0
    comp = dict()
    t = sorted(e.keys(), key = lambda k: e[k], reverse=True)
    nuevo = set()
    while len(t) > 0 and len(nuevo) < len(self.V):
        #print(len(t))
        arista = t.pop()
        w = e[arista]
        del e[arista]
        (u,v) = arista
        c = comp.get(v, {v})
        if u not in c:
            #print('u ',u, 'v ',v ,'c ', c)
            arbol.conecta(u,v,w)
            peso += w
            nuevo = c.union(comp.get(u,{u}))
            for i in nuevo:
                comp[i]= nuevo
    print('MST con peso', peso, ':', nuevo, '\n', arbol.E)
    return arbol
```

Método del vecino mas cercano

El método consiste en una vez establecido el nodo de partida, evaluar y seleccionar su vecino más cercano. Se explicará un ejemplo para mostrar cómo es que trabaja dicho método

En este caso:

Vecinos de A	B	C	D
Distancia	9	7	8

En la siguiente iteración habrá que considerar los vecinos más cercanos al nodo C (se excluye A por ser el nodo de origen):

Vecinos de C	B	D
Distancia	10	4

En la siguiente iteración los vecinos más cercanos de D serán C, con quien ya tiene conexión, A quién es el nodo de origen y B, por esta razón B se debe seleccionar por descarte. Al estar en B todos los nodos se encuentran visitados, por lo que corresponde a cerrar la red uniendo el nodo B con el nodo A, así entonces la ruta solución por medio del vecino más próximo sería A, C, D, B, A = 7, 4, 15, 9 = 35 km.

Este es un caso en el que, a pesar de tener una red compuesta por pocos nodos, el método del vecino más cercano no proporciona la solución óptima, la cual calculamos con el método de fuerza
bruta como 31 km.

Código de Python implementando el método del vecino más cercano

```
def vecinoMasCercano(self):
    lv = list(self.V)
    random.shuffle(lv)
    ni = lv.pop()
    le = dict()
    while len(lv)>0:
        ln = self.v[ni]
        for nv in ln:
            le[nv]=self.E[(ni,nv)]
        menor = min(le.values())
        lv.append(menor)
        del lv[menor]
    return lv
```

Solución exacta.

```
import time
def permutation(lst):
    if len(lst) == 0:
        return []
    if len(lst) == 1:
        return [lst]
    l = [] # empty list that will store current permutation
    for i in range(len(lst)):
       m = lst[i]
       remLst = lst[:i] + lst[i+1:]
       for p in permutation(remLst):
           l.append([m] + p)
    return l
```

Caso.

Un turista ha llegado a N.L y está buscando conocer más sobre las tradiciones, costumbres, vestimenta, gastronomía, etc., todo lo que sea posible sobre algunos municipios del estado que ya selecciono; Esto con la finalidad de terminar su tesis, pero se enfrenta a un problema ya que algunas de sus tarjetas fueron canceladas y no quiere regresar a su país, por lo cual ha decidido continuar con el viaje y reducir los gastos en la mayor cantidad posible.

Medidas.

Se ha diseñado un programa para ayudar al turista a optimizar las rutas de su traslado por lo cual buscaremos la ruta más corta para llegar a los municipios de Salinas Victoria, Mina, Anáhuac, Agua leguas, Abasolo, Marín, Bustamante, Zuazua, Sabinas Hidalgo, El Carmen de modo que sean visitados todos, pero sin la necesidad de repetir una misma ruta.

```
o= Grafo()
o.conecta('Salinas Victoria', 'Sabinas Hidalgo', 85)
o.conecta('Salinas Victoria', 'El Carmen', 9)
o.conecta('Salinas Victoria', 'Mina', 30)
o.conecta('Salinas Victoria', 'Anáhuac', 168)
o.conecta('Salinas Victoria', 'Agualeguas', 120)
o.conecta('Salinas Victoria', 'Abasolo', 13)
o.conecta('Salinas Victoria', 'Marin', 61)
o.conecta('Salinas Victoria', 'Bustamante', 74)
o.conecta('Salinas Victoria', 'Zuazua', 26)

o.conecta('Sabinas Hidalgo','El Carmen', 93)
o.conecta('Sabinas Hidalgo','Mina', 122)
o.conecta('Sabinas Hidalgo','Anáhuac', 137)
o.conecta('Sabinas Hidalgo','Agualeguas', 87)
o.conecta('Sabinas Hidalgo','Abasolo', 92)
o.conecta('Sabinas Hidalgo','Marin', 90)
o.conecta('Sabinas Hidalgo','Bustamante',42)
o.conecta('Sabinas Hidalgo','Zuazua', 80)

o.conecta('El Carmen','Mina',22)
o.conecta('El Carmen','Anáhuac', 178)
o.conecta('El Carmen','Agualeguas', 139)
o.conecta('El Carmen','Abasolo', 5)
o.conecta('El Carmen','Marin', 54)
o.conecta('El Carmen','Bustamante', 82)
o.conecta('El Carmen','Zuazua', 40)
```

```
o.conecta('Mina','Anáhuac', 197)
o.conecta('Mina','Agualeguas', 156)
o.conecta('Mina','Abasolo', 17)
o.conecta('Mina','Marin', 70)
o.conecta('Mina','Bustamante', 102)
o.conecta('Mina','Zuazua', 56)

o.conecta('Anáhuac','Agualeguas', 230)
o.conecta('Anáhuac','Abasolo', 180 )
o.conecta('Anáhuac','Marin', 203)
o.conecta('Anáhuac','Bustamante', 105)
o.conecta('Anáhuac','Zuazua', 194)

o.conecta('Agualeguas','Abasolo', 139)
o.conecta('Agualeguas','Marin', 90)
o.conecta('Agualeguas','Bustamante', 127)
o.conecta('Agualeguas','Zuazua', 98)

o.conecta('Abasolo','Marin', 56)
o.conecta('Abasolo','Bustamante', 88)
o.conecta('Abasolo','Zuazua', 42 )

o.conecta('Marin','Bustamante', 109)
o.conecta('Marin','Zuazua', 11)

o.conecta('Bustamante','Zuazua', 98)
```

En la parte del código anterior se muestran conectados 2 municipios entre sí sin considerar la dirección de las aristas ya que cada uno de ellos conforman nodos de nuestro grafo, previamente se hizo una investigación de la ruta en kilómetros entre cada uno de ellos, por lo que esperamos visitar cada nodo en la menor cantidad de km posibles ya que el turista que está visitando nuestro estado desea gastar poco combustible y seguir disfrutando de los pueblos de la región.

Resultados

La ruta más óptima para visitar todos los municipios utilizando MST:

Municipio inicial	Municipio Final	peso
Abasolo	El Carmen	5
El Carmen	Salinas Victoria	9
Zuazua	Marín	11
Abasolo	Mina	17
Zuazua	Salinas Victoria	26
Bustamante	Sabinas Hidalgo	42
Bustamante	Salinas Victoria	74
Agualeguas	Sabinas Hidalgo	87
Bustamante	Anáhuac	105

Con un peso de 376.

- Utilizando fuerza bruta se llegó podemos visitar los municipios de la siguiente manera:

Municipio de Inicio	Municipio Final	Peso

Anáhuac	Bustamante	105
Bustamante	Salinas Victoria	74
Salinas	Victoria Zuazua	26
Zuazua	Marín	11
Marín	El Carmen	54
El Carmen	Abasolo	5
Abasolo	Mina	17
Mina Sabinas	Hidalgo	122
Sabinas Hidalgo	Agualeguas	87
Agualeguas	Anáhuac	230

Con un costo de 731

- La segunda opción es:

Municipio de Inicio	Municipio Final	Peso
Mina	Abasolo	17
Abasolo Municipio de Inicio	El Carmen	5

El Carmen	Salinas Victoria	9
Salinas Victoria	Bustamante	74
Bustamante	Sabinas Hidalgo	42
Sabinas Hidalgo	Agualeguas	87
Agualeguas	Anáhuac	230
Anáhuac	Zuazua	194
Zuazua	Marín	11
Marín	Mina	70

Con un costo de 739

- Por último, la tercera opción es:

Municipio de Inicio	Municipio Final	Peso
Marín	Zuazua 11	11
Zuazua	Salinas Victoria 26	26
Salinas Victoria	Bustamante 74	74
Bustamante	Sabinas Hidalgo 42	42

Agualeguas	Anáhuac 230	230
Anáhuac	El Carmen 178	178
El Carmen	Abasolo 5	5
Abasolo	Mina 17	17
Mina	Marlín	70

Con un de costo 740

Tiempo de ejecución

Tiempo que tarda el programa en realizar las permutaciones 27.051840260834947

Implementación

- En general, los dos algoritmos constructivos son capaces de llegar al óptimo en instancias pequeñas, todos con un tiempo de cómputo pequeño.

- En instancias grandes, la miopía del vecino más cercano se hace evidente, sin embargo, es el algoritmo con el mejor tiempo de cómputo, esto es, el más rápido.

- La inserción aleatorizada ofrece soluciones de buena calidad aun en instancias grandes, pero es más tardada que el vecino más cercano.

- La búsqueda local unida al vecino más cercano genera una mejora significativa sin comprometer el tiempo. Unida a la inserción aleatorizada, sigue obteniendo muy buenas soluciones, sin embargo, el tiempo aumenta considerablemente.

- Para decidir qué algoritmo utilizar se debe ponderar tanto el tiempo que se tiene para dar una solución, así como qué tan estricta es la exigencia de la calidad de la solución.

Conclusión.

Se tiene bases de que el problema del agente viajero ha sido estudiado por largo tiempo para encontrar la mejor solución posible, pero esto lamentablemente no se ha logrado, lo cual resulta muy triste ya que lo podríamos poner en práctica en cualquier área de estudio, en esta evaluación se tuvo que analizar distintos métodos para encontrar la mejor respuesta , sinceramente considero que la tecnología y la curiosidad humana está sobresaliendo cada día así que no me parecería extraño que en poco tiempo este problema termine por resolverse ya que son muchas las personas que han propuesto soluciones para el TSP.

Por ultimo quiero resaltar que yo en lo particular decidí trabajar con las distancias mas aproximadas a las originales para que en un futuro pueda aplicarla , ya que mi colonia pertenece al municipio de Salinas Victoria y creo que antes de viajar a otro país debemos comenzar por conocer nuestras tierras.

EVALUACION NO.6 Leslie J. Morales Ortiz 1616971 sábado 4/Nov/17