Universidad Autónoma de Nuevo León Facultad de Ciencias Físico Matemáticas

Artemio Guajardo Aparicio Grupo: 001 Matrícula: 1590417 Fecha: 05/11/2017

Soluciones al problema del agente viajero

Resumen: Implementación de un algoritmo de aproximación usando el algoritmo de Kruskal para obtener un árbol de expansión mínima y de él crear una solución aproximada al problema del agente viajero.

Problema del agente viajero (PAV)

En el Problema del Agente Viajero 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. El problema fue formulado por primera vez en 1930 y es uno de los problemas de optimización más estudiados. Es usado como prueba para muchos métodos de optimización.

¿Qué es lo difícil en el problema del agente viajero?

Como problema de la vida real es algo muy complejo pues hay muchas variables que se deben tomar en cuenta al momento de considerar las distancias o costos entre un nodo y otro (tipo de transporte, tráfico, clima, retrasos, inconvenientes, etc.). También se debe tomar en cuenta que la distancia de un nodo ‘a’ a un nodo ‘b’ no siempre es la misma que del nodo ‘b’ al nodo ‘a’ debido a las rutas que se deben tomar, lo cual muchas veces alarga el problema. Se deben tener muy en cuenta todas y cada una de estas variables para poder resolver eficientemente este problema. En este caso omitiremos todas esas variables con el fin de simplificar el proceso.

Algoritmo de aproximación

Un algoritmo de aproximación es un algoritmo usado para encontrar soluciones aproximadas a problemas de optimización. Lo que busca este algoritmo es encontrar soluciones que está demostrado son de calidad y cuyos tiempos de ejecución están acotados

por cotas conocidas. Idealmente, la aproximación mejora su calidad para factores constantes pequeños (por ejemplo, dentro del 5% de la solución óptima).

Implementación de un algoritmo de aproximación en Python:

```
k=g.kruskal()
                print([print(x, k.E[x]) for x in k.E])
                for r in range(10):
                    ni = random.choice(list(k.V))
                    dfs =  k.DFS(ni)
                    c = 0
                    #print(dfs)
                    #print(len(dfs))
                    for f in range(len(dfs) -1):
                            c += g.E[(dfs[f],dfs[f+1])]
                            print(dfs[f], dfs[f+1], g.E[(dfs[f],dfs[f+1])] )

                    c += g.E[(dfs[-1],dfs[0])]
                    print(dfs[-1], dfs[0], g.E[(dfs[-1],dfs[0])])
                    print('costo: c)
```

Este algoritmo se utiliza habiendo declarado antes una función con el algoritmo Kruskal.

¿Qué hace el algoritmo de Kruskal?

Es un algoritmo utilizado en teoría de grafos para encontrar un árbol de expansión mínima en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque de expansión mínima (un árbol de expansión mínima para cada componente conexa). Este algoritmo toma su nombre de Joseph Kruskal, quien lo publicó por primera vez en 1956.

Implementación del algoritmo de Kruskal en Python:

```
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
```

¿Qué es un árbol de expansión mínima?

Es un modelo de optimización de redes que consiste en enlazar todos los nodos de la red de forma directa y/o indirecta con el objetivo de que la longitud total de los arcos o ramas sea mínima (entiéndase por longitud del arco una cantidad variable según el contexto operacional de minimización, y que puede bien representar una distancia o unidad de medida).

Ejemplo:

Una familia está de vacaciones, pero no cuenta con mucho tiempo y desea saber cuál es la ruta más corta para ir a 10 municipios de Nuevo León:

- China

- General Bravo
- Galeana
- Abasolo
- Anáhuac
- García
- Aramberri
- Sabinas Hidalgo
- Parás
- Lampazos

De los cuales tenemos los siguientes datos respecto a las distancias:

De China a General Bravo: 12.3 km

De China a Galeana: 214.5 km

De China a Abasolo: 144 km

De China a Anáhuac: 330.8 km

De China a García: 158.1 km

De China a Aramberri: 312.4 km

De China a Sabinas Hidalgo: 161.6 km

De China a Paras: 124.8 km

De China a Lampazos: 263.8 km

De General Bravo a Galeana: 226.9 km

De General Bravo a Abasolo: 152.2 km

De General Bravo a Anáhuac: 318 km

De General Bravo a García: 166.3 km

De General Bravo a Aramberri: 324.7 km

De General Bravo a Sabinas Hidalgo: 214.7 km

De General Bravo a Paras: 127.7 km

De General Bravo a Lampazos: 271.9 km

De Galeana a Abasolo: 252.1 km

De Galeana a Anáhuac: 422.3 km

De Galeana a García: 221.1 km

De Galeana a Aramberri: 112.9 km

De Galeana a Sabinas Hidalgo: 327.2 km

De Galeana a Paras: 232.8 km

De Galeana a Lampazos: 376.2 km

De Abasolo a Anáhuac: 182 km

De Abasolo a García: 47.3 km

De Abasolo a Aramberri: 339.2 km

De Abasolo a Sabinas Hidalgo: 108.6 km

De Abasolo a Paras: 154.8 km

De Abasolo a Lampazos: 135.9 km

De Anáhuac a García: 217.8 km

De Anáhuac a Aramberri: 509.7 km

De Anáhuac a Sabinas Hidalgo: 136.7 km

De Anáhuac a Paras: 208.7 km

De Anáhuac a Lampazos: 46.4 km

De García a Aramberri: 308.2 km

De García a Sabinas Hidalgo: 124.9 km

De García a Paras: 171.1 km

De García a Lampazos: 172.3 km

De Aramberri a Sabinas Hidalgo: 412.8 km

De Aramberri a Paras: 459 km

De Aramberri a Lampazos: 461.8 km

De Sabinas Hidalgo a Paras: 71.5 km

De Sabinas Hidalgo a Lampazos: 90.4 km

De Paras a Lampazos: 161.4 km

Soluciones:

Con ayuda de nuestro algoritmo de aproximación se obtuvieron las siguientes 5 soluciones:

La mejores rutas aprox. son:	**Distancia Total**	**Tiempo de ejecución**
Aramberri>Galeana>China >Parás>SabinasHidalgo>Abasolo >García>Lampazos>Anáhuac >General Bravo>Aramberri	1541 Kilómetros	2.292 segundos
Abasolo>Sabinas Hidalgo >Lampazos>Anáhuac>Parás >China>Galeana>Aramberri >General Bravo>García >Abasolo	1444.6 Kilómetros	2.75 segundos

*Mejor Ruta

García>Abasolo>SabinasHidalgo >Lampazos>Anáhuac>Parás >China>Galeana>Aramberri >General Bravo>García	1444.7 Kilómetros	2.83 segundos
Galeana>China>Parás >Sabinas Hidalgo>Abasolo >García>Lampazos>Anáhuac >General Bravo>Aramberri >Galeana	1541 Kilómetros	2.26 segundos

*El algoritmo solo lanzó 4 rutas más cortas, no fue posible obtener más.

Heurística del vecino más cercano

El método del vecino más cercano es un algoritmo heurístico diseñado para solucionar el problema del agente viajero, no asegura una solución óptima, sin embargo suele proporcionar buenas soluciones, y tiene un tiempo de cálculo muy eficiente. El método de desarrollo es muy similar al utilizado para resolver problemas de árbol de expansión mínima.

Implementación del método en Python:

```
def vecinoMasCercano(self):
        ni = random.choice(list(self.V))
        result=[ni]
        while len(result) < len(self.V):
            ln = set(self.vecinos[ni])
            le = dict()
            res =(ln-set(result))
            for nv in res:
                le[nv]=self.E[(ni,nv)]
            menor = min(le, key=le.get)
            result.append(menor)
            ni=menor
        return result
```

Soluciones:

Soluciones:	Distancia	Tiempo de ejecución
General Bravo>China>Parás >Sabinas Hidalgo >Lampazos>Anáhuac>Abasolo >García>Galeana>Aramberri >General Bravo	1233.39 Kilómetros	1.04 segundos
Sabinas Hidalgo>Parás>China >General Bravo>Abasolo >García>Lampazos>Anáhuac >Galeana>Aramberri >Sabinas Hidalgo	1574.8 Kilómetros	1.26 segundos
Anáhuac>Lampazos >Sabinas Hidalgo>Parás >China>General Bravo >Abasolo>García>Galeana >Aramberri>Anáhuac	1388.6 Kilómetros	1.67 segundos
Abasolo>García >Sabinas Hidalgo>Parás >China>General Bravo >Galeana>Aramberri>Lampazos >Anáhuac>Abasolo	1410.8 Kilómetros	1.18 segundos
García>Abasolo>Sabinas H. >Parás>China>General Bravo >Galeana>Aramberri>Lampazos >Anáhuac>García	1430.3 Kilómetros	1.55 segundos

*Mejor Ruta

Solución exacta

Este método calcula todas las formas posibles en las que se pueden permutar los nodos de nuestro grafo. De forma implementada obtenemos lo siguiente:

```
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
```

Ruta solución exacta (utilizando el algoritmo):

Aramberri>Galeana

Galeana>China

China>General Bravo

General Bravo>Parás

Parás>Sabinas Hidalgo

Sabinas Hidalgo>Lampazos

Lampazos>Anáhuac

Anáhuac>Abasolo

Abasolo>García

García>Aramberri

Con una distancia de 1213.19 Kilómetros, obtenido en un total de 254.102 segundos

Conclusiones

El algoritmo de aproximación arrojó como mejor ruta una que tenía un total de 1444.6 kilómetros en 2.75 segundos, el algoritmo del vecino más cercano arrojó como mejor ruta una que tenía un total de 1233.39 Kilómetros en 1.04 segundos y el algoritmo de solución exacta arrojo una de 1213.19 Kilómetros en 254.102 segundos.

De estos resultados podemos concluir que si lo que se necesita son soluciones rápidas, el algoritmo del vecino más cercano es el mejor, pues es más rápido que el de aproximación y más exacto. Si hay mucha disponibilidad de tiempo o se busca la menor ruta a toda costa el algoritmo de solución exacta es el ideal puesto que como su nombre lo dice, es el más exacto pero su efectividad viene con una gran diferencia de tiempo de los otros dos algoritmos, desgraciadamente mayor.