Alicia Magaly Azúa Saucedo

1743217

22 de mayo del 2018

Reporte de problema del viajero

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, o el tiempo total del recorrido.

Este tipo de problemas tiene gran aplicación en el ámbito de la logística y distribución, así como en la programación de curvas de producción.

El problema del agente viajero tiene una variación importante, y esta depende de que las distancias entre un nodo y otro sean simétricas o no, es decir, que la distancia entre A y B sea igual a la distancia entre B y A, puesto que en la práctica es muy poco probable que así sea.

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

$$(n-1)!$$

Es decir que en una red de 5 nodos la cantidad de rutas probables es igual a (5-1)! = 24, y a medida que el número de nodos aumente la cantidad de rutas posibles crece factorialmente. En el caso de que el problema sea simétrico la cantidad de rutas posibles se reduce a la mitad, es decir:

$$((n-1)!) / 2$$

Lo cual significa un ahorro significativo en el tiempo de procesamiento de rutas de gran tamaño.

¿Qué es lo "difícil" en el Problema del Agente Viajero?

En el ámbito de la teoría de complejidad computacional, el PAV pertenece a la clase de problemas NP-completos. Por lo tanto, se supone que no hay ningún algoritmo eficiente para la solución de PAV. En otras palabras, el número de posibles soluciones es tan elevado que si pretendemos que el algoritmo encargado de la búsqueda de la solución óptima deba verificar una a una no tendremos tiempo de cálculo para hallarlo: es probable que en el peor de los casos el tiempo de resolución de cualquier algoritmo para

PAV aumente exponencialmente con el número de ciudades, por lo que incluso en algunos casos de tan sólo cientos de ciudades se tardarán bastantes años de CPU para resolverlos de manera exacta.
Algunos de los métodos que se pueden utilizar para solucionarlo son:

El método de la fuerza bruta:

El método de la fuerza bruta no implica la aplicación de ningún algoritmo sistemático, tan solo consiste en explorar todos los recorridos posibles.

El métodos del vecino más cercano:

El algoritmo del vecino más próximo fue uno de los primeros algoritmos utilizados para determinar una solución para el problema del viajante. Este método genera rápidamente un camino corto, pero generalmente no el ideal. Estos son los pasos del algoritmo:

- elección de un vértice arbitrario respecto al vértice actual.
- descubra la arista de menor peso que ya este conectada al vértice actual y a un vértice no visitado V.
- convierta el vértice actual en V.
- marque V como visitado.
- si todos los vértices del dominio estuvieran visitados, cierre el algoritmo

Algoritmo Kruskal:

El algoritmo de Kruskal es un ejemplo de algoritmo voraz que funciona de la siguiente manera:

- Se crea un bosque B (un conjunto de árboles), donde cada vértice del grafo es un árbol separado
- Se crea un conjunto C que contenga a todas las aristas del grafo
- Mientras C es no vacío: o Eliminar una arista de peso mínimo de C o Si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol o En caso contrario, se desecha la arista Al acabar el algoritmo, el bosque tiene un solo componente, el cual forma un árbol de expansión mínimo del grafo. El algoritmo de Kruskal muestra una complejidad O(m log n), cuando se ejecuta sobre

estructuras de datos simples.

Código:

```
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
```

Para una representación del algoritmo, se seleccionaron 10 países de distintas partes del mundo: México, Brasil, Canadá, Perú, Argentina, Francia, Noruega, Groenlandia, Nigeria y

Sudáfrica. Cuyas distancias (En miles de Kilómetros) son:

México Brasil 6.52

México Canadá 3.32

México Perú 4.62

México Argentina 7

México Francia 9.1

México Noruega 8.65

México Groenlandia 6.32

México Nigeria 11.96

México Sudáfrica 14.86

Brasil Canadá 6.61

Brasil Perú 2.56

Brasil Argentina 1.93

Brasil Francia 8.41

Brasil Noruega 9.74

Brasil Groenlandia 9.13

Brasil Nigeria 7.4

Brasil Sudáfrica 8.4

Canadá Perú 6.03

Canadá Argentina 8

Canadá Francia 5.78

Canadá Noruega 5.43

Canadá Groenlandia 3.33

Canadá Nigeria 9.16

Canadá Sudáfrica 13.06

Perú Argentina 2.38

Perú Francia 9.9

Perú Noruega 10.7

Perú Groenlandia 9.23

Perú Nigeria 9.88

Perú Sudáfrica 10.75

Argentina Francia 10.33

Argentina Noruega 11.66

Argentina Groenlandia 10.87

Argentina Nigeria 8.75

Argentina Sudáfrica 8.56

Francia Noruega 1.85

Francia Groenlandia 3.67

Francia Nigeria 4.41

Francia Sudáfrica 8.63

Noruega Groenlandia 2.43

Noruega Nigeria 6.13

Noruega Sudáfrica 10.19

Groenlandia Nigeria 8.03

Groenlandia Sudáfrica 12.29

Nigeria Sudáfrica 4.27

Al correr el algoritmo de kruskal con la información anterior los resultados obtenidos

fueron los siguientes: {Sud, Nig, Fran, Nor, Groen, Can, Mex, Peru, Arg, Bra, Sud}

Sud Nig 4.27

Nig Fran 4.41

Fran Nor 1.85

Nor Groen 2.43

Groen Can 3.33

Can Mex 3.32

Mex Peru 4.62

Peru Arg 2.38

Arg Bra 1.93

Bra Sud 8.4

costo 36.94