UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE CIENCIAS FÍSICO – MATEMÁTICAS

MATEMÁTICAS COMPUTACIONALES

“ALGORITMO DE DIJKSTRA”

Nombre: Diego Rubio Romero

Matrícula: 1738544

Carrera: Lic. En Matemáticas

Profesor: Lic. José Anastacio Hernández Saldaña

Grupo: 002

Fecha: Domingo 5 de Noviembre del 2017

Lugar: Ciudad Universitaria

Problema del Agente Viajero (PAV)

Definición

En el Problema del Agente Viajero (PAV), o también es conocido por su nombre en inglés Travelling Salesman Problem (TSP), 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.

¿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.

¿Qué es un algoritmo de aproximación?

Un algoritmo de aproximación es un algoritmo usado para encontrar soluciones aproximadas a problemas de optimización. A diferencia de las heurísticas, que usualmente sólo encuentran soluciones razonablemente buenas en tiempos razonablemente rápidos, lo que se busca aquí es encontrar soluciones que está demostrado son de calidad y cuyos tiempos de ejecución están acotadas por cotas conocidas. Idealmente, la aproximación mejora su calidad para factores constantes pequeños.

Árbol de Expansión Mímima: Algoritmo de Kruskal

Definiciones

Árbol: Un árbol es una gráfica en la cual no existen ciclos.

Árbol de expansión: Un árbol de expansión es aquel árbol que enlaza todos los nodos de la red, de igual manera no permite la existencia de ciclos.

El algoritmo del **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 ramales sea mínima.

Primeramente ordenaremos las aristas del grafo por su peso de menor a mayor. Mediante la técnica greedy, **Kruskal** intentara unir cada arista siempre. Como hemos ordenado las aristas por peso comenzaremos con la arista de menor peso, si los vértices que contienen dicha arista no están en la misma componente conexa entonces los unimos para formar una sola componente mediante Union(x , y), para revisar si están o no en la misma componente conexa usamos la función SameComponent(x , y) al hacer esto estamos evitando que se creen ciclos y que la arista que une dos vértices siempre sea la mínima posible.

Descripción

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:*
    - Eliminar una arista de peso mínimo de *C*
    - Si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol
    - 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.

Pseudocodigo

```
función Kruskal(G)
   Para cada v en V[G] hacer
     Nuevo conjunto C(v) ← {v}.
   Nuevo heap Q que contiene todas las aristas de G, ordenando por su peso.
   Defino un arbol T ← Ø
   // n es el número total de vértices
   Mientras T tenga menos de n-1 aristas y !Q.vacío() hacer
     (u,v) ← Q.sacarMin()
     // previene ciclos en T. agrega (u,v) si u y v están diferentes componentes en el conjunto.
     // Nótese que C(u) devuelve la componente a la que pertenece u.
     Si C(v) ≠ C(u) hacer
       Agregar arista (v,u) a  T.
       Merge C(v) y C(u) en el conjunto
   Responder arbol T
```

Descripción del ejemplo

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

Heurística 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.

Al correr el algoritmo 5 veces con la información anterior los resultados obtenidos fueron los siguientes:

1- 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 Fran 8.41
Fran Nig 4.41
Nig Sud 4.27
Sud Nor 10.19
costo 45.28999999999999

2- Groen Can 3.33
Can Mex 3.32
Mex Peru 4.62

Peru Arg 2.38

Arg Bra 1.93

Bra Nor 9.74

Nor Fran 1.85

Fran Nig 4.41

Nig Sud 4.27

Sud Groen 12.29

costo 48.14

3- Fran Nig 4.41

Nig Sud 4.27

Sud Nor 10.19

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 Fran 8.41

costo 45.28999999999999

4- Can Groen 3.33

Groen Nor 2.43

Nor Fran 1.85

Fran Nig 4.41

Nig Sud 4.27

Sud Mex 14.86

Mex Peru 4.62

Peru Arg 2.38

Arg Bra 1.93

Bra Can 6.61

costo 46.69

5- Can Mex 3.32

Mex Peru 4.62

Peru Arg 2.38

Arg Bra 1.93

Bra Nig 7.4

Nig Sud 4.27

Sud Fran 8.63

Fran Nor 1.85

Nor Groen 2.43

Groen Can 3.33

costo 40.16

Podemos observer que el mejor camino fue:

{Can, Mex, Peru, Arg, Bra, Nig, Sud, Fran, Nor, Groen} con un peso de 40.16

Solución Exacta

Trata de estar corriendo toda la combinación de ciclos posibles, uno a uno, hasta terminar, se obtiene la solución más óptima de todas.

Solución exacta con peso 28.54 : {'Peru', 'Arg', 'Fran', 'Nor', 'Nig', 'Mex', 'Bra', 'Can', 'Sud', 'Groen'}

{('Nor', 'Fran'): 1.85,

('Bra', 'Arg'): 1.93,

('Peru', 'Arg'): 2.38,

('Nor', 'Groen'): 2.43,

('Mex', 'Can'): 3.32,

('Can', 'Groen'): 3.33,

('Nig', 'Sud'): 4.27,

('Fran', 'Nig'): 4.41,

('Mex', 'Peru'): 4.62}

Tiempo de ejecución: 122.97079663756148

Conclusión

Pudimos observar que el problema del agente viajero, no es muy simple que digamos, ya que no existe un algoritmo que determine la ruta más eficiente por una cantidad significativa de tiempo. Es decir, para tener la solución más óptima, se necesita correr todas las combinaciones entre ciudades.

Por otro lado existen alternativas de algoritmos, que a pesar de no ser exactos, se pueden usar, para que en cierta cantidad de veces que se corra se encuentre una solución "buena" para el problema, ya sea el algoritmo de kruskal, o el del vecino más cercano.