Universidad de Santiago de Chile

Facultad de Ingeniería Informática

Problema del Viajero Tarea 2

Diseño de Algortimos

Sección: B-2
Profesor: Jose Fernandez Goycolea
Autor: Nícolas André Farfán Cheneaux

Julio 2021

Contents

Problema del viajero (TSP)

1 Descripción formal del problema

El problema del viajero o en inglés Travelling Salesman Problem(TSP) es un problema de Optimización Combinatoria, es decir que busca la mejor solución o solución menos costosa; entre un conjunto finito de posibles soluciones a un problema.

1.1 Idea General

Un vendedor debe recorrer n ciudades y regresar al punto de partida, cada ciudad está a una determinada distancia, por lo tanto cada camino tiene un costo a recorrer.

1.2 Descripción Formal

Puede ser formalizado cómo un problema de grafos: Se debe encontrar el "tour" o camino de menor costo que recorra todas las ciudades(nodos) y pase por los caminos menos costosos (aristas) y regrese al origen.

Según Wikipedia la formulación de Miller-Tucker-Zemlin es formalizado como un problema de Programación Lineal de Enteros, se define una variable x, si toma el valor 1, se va a ciudad y 0 si no, también se define "c" cómo el costo de camino A-B y sea n el número de ciudades.

Se define i - j el número de la ciudad

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$$

- 1 si se pasa por tal camino de la ciudad i - j y se paga el costo i - j.
- 0 si no se pasa por tal camino de la ciudad i - j y no se paga el costo i - j.

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}:$$
$$x_{ij} \in \{0,1\} \qquad i,j = 1,\ldots,n;$$

- Se busca minimizar la sumatoria de el costo(c) * (Si pasa o no)

Comprobaciones ante una posible solución

$$\sum_{i=0, i \neq j}^{n} x_{ij} = 1 \qquad j = 0,\ldots,n$$

$$\sum_{j=0, j \neq i}^{n} x_{ij} = 1 \qquad i = 0,\ldots,n$$

- La primera sumatoria comprueba que de todas las posibles salidas, se llegue solamente a una

ciudad.
- La segunda sumatoria comprueba que de todas las posibles llegadas, se salga solamente a una ciudad.

Validar la Solución

$$u_i - u_j + nx_{ij} \leq n-1 \qquad 2 \leq i \neq j \leq n;$$
$$1 \leq u_i \leq n-1 \qquad 2 \leq i \leq n.$$

Para validar una solución se verifica que cualquier *tour* escogido recorra todos los nodos:
Representa ui - uj las ciudades visitadas, n el número de ciudades, por lo tanto la primera igualdad valida que los tours o caminos seguidos, sean menor o igual al número de ciudades recorridas -1.

1.3 Esquema del problema(Ejemplo)

1.3.1 Ciclos Hamiltonianos

Según Wikipedia un ciclo Hamiltoniano visita todos los nodos de un grafo y regresa al origen sin pasar dos veces por un mismo nodo, en contexto al problema, esto es exáctamente lo que buscamos visitar todos los nodos, pero encontrando el camino menos costoso.

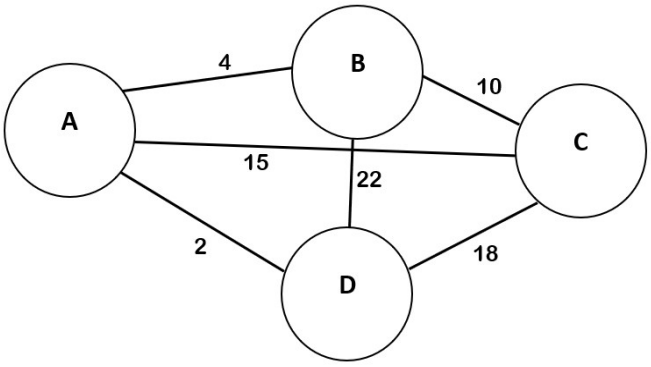

Figura 1: Representación de un grafo no dirigido, cada nodo representa una ciudad [A,B,C,D] y cada arista tiene el valor del costo por recorrer tal camino[2,15,9,18,10,8](Este grafo debe ser conexo y ponderado).

2 Aplicaciones prácticas del problema

Este problema de combinatoria tiene múltiples aplicaciones:

- Cómo su nombre lo indica, una compañía puede utilizar este método para encontrar el camino con menor costo en que un vendedor o repartidor puede ir a "n" ciudades y regresar en el menor costo posible, ejemplos cómo repartición de correos, movilidad escolar.

- Para colocar alambrado de cable de fibra óptica por ejemplo, entre ciudades, se prefiere que sea el menor costo y distancia posible.

- A gran escala, con muchas ciudades(nodos) en un aeropuerto es clave para calcular las distancias y recorrido que debe serguir un vuelo al menor costo.

- En GPS o Google Maps puede ser utilizado para encontrar la ruta más rápida a un número de "n" destinos ingresados por el usuario.

- En términos de ciencias de la computación encontrar la menor distancia o diferencia entre un dato u otro, optimiza en tiempo y espacio de ejecución.

- En algunas areas de la informática o manejo de memoria, un grafo dirigido puede representar un conjunto de instrucciones en las que deben ser ejecutadas en un cierto orden y si no se ejecuta en ese orden, no funciona, por lo tanto para encontrar tal proceso es necesario encontrar el camino ideal o correcto.

3 Estrategia de Solución

3.1 Estrategia 1: BackTracking - Fuerza Bruta

3.1.1 Resolución del Ejemplo(Figura 1)

Cómo su nombre lo indica esta estrategia obtiene todas las posibles soluciones a recorrer, por lo tanto tiene un costo de n! al existir n! posibles caminos.
Desde el punto de vista heurísitco esta solución no es eficiente y es posible encontrar una mejor.

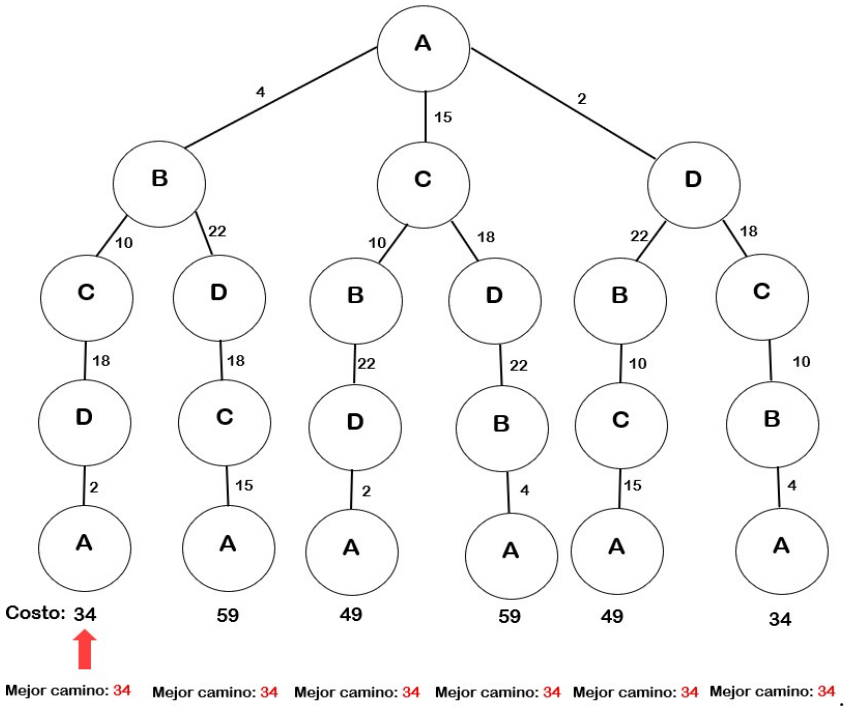

Figura 2: Representación de un arbol, cada nodo representa una ciudad [A,B,C,D] y cada rama tiene el valor del costo por recorrer tal camino [2,15,9,18,10,8] (Punto de partida A)

Caminos Posibles:
Punto de inicio: A

A B C D A
A B D C A
A C B D A
A C D B A
A D C B A
A D B C A

Cuyos costos a recorrer son:
A B C D A = 34
A B D C A = 59
A C B D A = 49 (Adicionando los caminos inversos)

Actualizando cada vez que llega al final, la mejor solución es 34.

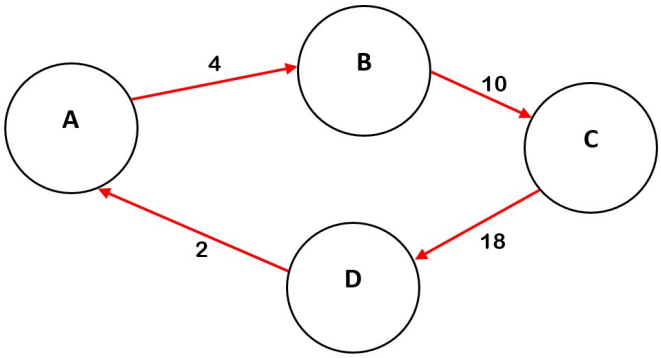

Figura 3: Representación de la solución obtenida por el algortimo de Backtracking(A B C D A), mediante un grafo dirigido

3.1.2 Pseudocódigo BackTracking y encontrar mínimo

```
1. Leer(Grafo,v,a)
2.    Declarar cota = 0, i = 0, lista[n] (Lista de tamanio n)
3.    Generar_Camino(Grafo,v,a)
4.    While (No esten recorridos todos los caminos):
5.        Generar camino(Grafo,v,a)
6.        if (Camino valido*):
7.            Almacencar solucion en lista[i]
8.            i+=1
9. Retornar lista[n]
```

Función buscarMinimo():

```
1. buscarMinimo(A):
2.       if (largo(A) == 1):
3.              devolver A[0]
4.       else:
5.           a = buscarMinimo(A[1:n/2])
6.           b = buscarMinimo(A[n/2:n])
7. if (a > b):
8.       devolver b
9. else:
10.      devolver a
```

**Camino Valido hace referencia a la condición, ubicada en la sección 1.2*

3.2 Estrategia 2: BackTracking con poda

3.2.1 Resolución del Ejemplo(Figura 1)

En este caso se utilizo BackTracking con poda para optimizar el algortimo y no recorrer todas las posibles soluciones utilizando el criterio de si la **cota* **"Mejor Camino"** es mayor a la mejor solución hasta el momento, si es así se poda la rama, se deja de recorrer esta y se pasa a la siguiente.

***cota: El último camino con menor costo obtenido**

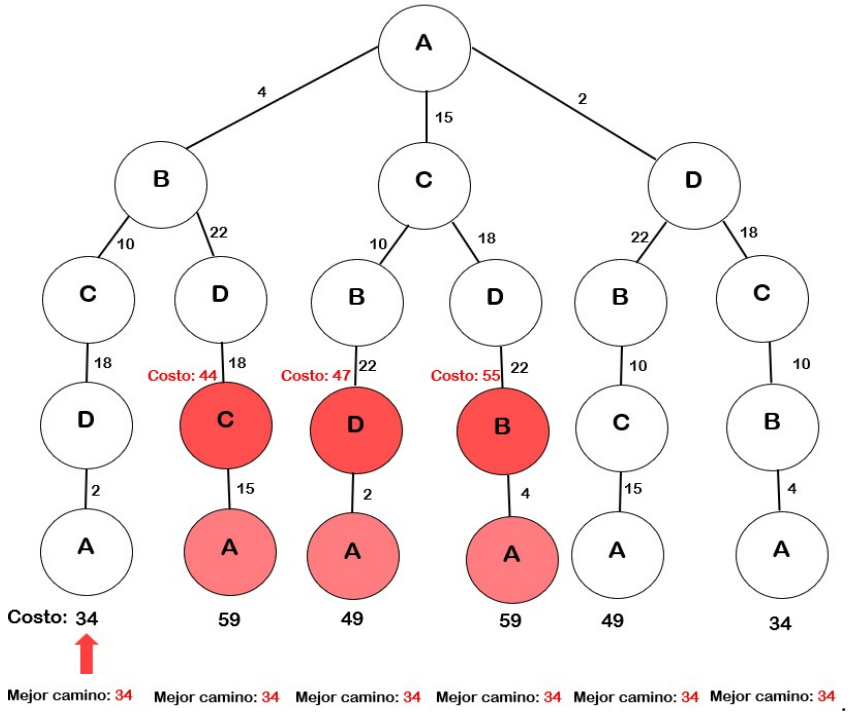

Figura 4: Representación de un arbol con poda, los nodos rojos fueron podados siguiendo la condición que si es menor que la mejor solución hasta el momento ***(Mejor camino = Cota)***

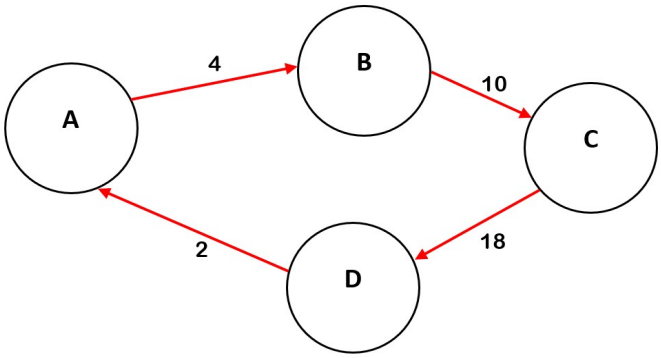

Figura 5: Representación de la solución obtenida por el algortimo de Backtracking(A B C D A), mediante un grafo dirigido

3.2.2 Pseudocódigo BackTracking con poda

```
1. Leer(Grafo,v,a)
2.    Declarar cota = 0
3.    Generar_Camino(Grafo,v,a)
4.    While (No esten recorridos todos los caminos):
5.          Generar_Camino(Grafo,v,a)
6.          if ((Camino valido)* && (camino < cota)):
7.              cota = costo de camino      (Actualizar cota)
8.              mejor_camino = camino
9. Retornar mejor_camino     (Retorna el costo del mejor camino)
```

**Camino Valido hace referencia a la condición, ubicada en la sección 1.2*

4 Complejidad del algortimo y evaluación de la solución escogida

4.1 Estrategia 2: Fuerza Bruta - BackTracking

La estrategia **siempre tiene una complejidad de O(n!+n)**, las ventajas que ofrece son:

- Puede encontrar todas las posibles soluciones

- No necesariamente debe guardar todas las soluciones, sino comparar con la más óptima hasta el momento.

- Encontrará 2 soluciones iguales, por lo tanto se quedará con 1. Desventajas:

- Tiene un costo computacional muy alto al trabajar con muchas ciudades, luego de tener todas las soluciones válidas, debe encontrar la de menor costo, lo cual tiene un costo de O(n) + O(n!)

4.2 Estrategia 2: BackTracking Poda

La estrategia **tiene una complejidad de O(n!) en el peor caso**, al tener que recorrer todas las ramas en busca de una solución óptima, las ventajas que ofrece son:

- Ahorrará memoria y ejecución al *Cortar ramas* y dejando de ejecutar caminos de un mayor costo que el óptimo hasta el momento, por lo tanto no ejecuta ni recorrer todas las posibles soluciones y sin almacenarlas.

- Encontrará 1 solución óptima.

- Existe la posiblidad que encuentre la solución óptima en menos pasos que la complejidad en el **peor caso O(n!).**

5 Otras alternativas de solución

5.1 Estrategia Alternativa 1: Algoritmo Greedy o el vecino más cercano

Cómo su nombre lo indica compara los caminos disponibles y escoge el de menor costo, pero no mira al futuro y la solución que entrega por lo general no es óptima.

5.1.1 Resolución del Ejemplo(Figura 1)

Usamos el Heurístico de el camino más cercano de menor valor no visitado:
El camino A tiene 3 vecinos, el camino más óptimo es de A - D con un costo de 2
D tiene 2 vecinos ya que no puedo retroceder, el más óptimo es D - C con un costo de 18, luego de B - C con un costo de 10 y finalmente A - B con un costo de 4. El costo total es: 34

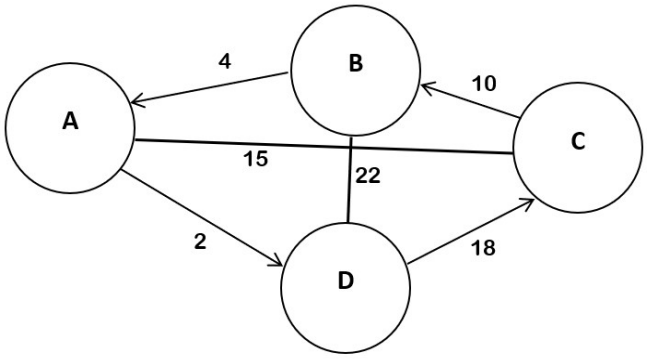

Figura 6: Representación de el grafo de ejemplo, se presentan flechas representando el recorrido que se seguiría usando el heurístico del camino cercano de menor costo.

5.1.2 Pseudocódigo Greedy

```
1. Leer(Grafo,V,A)
2.    Declarar suma = 0
3.    Greedy(v,a)
4.    Encontrar arista (a) de menor costo de vertice(v)
5.    While (No se recorran todos los vertices && arista no recorrida antes):
6.            suma = suma + costo_arista
7.            Greedy(sig_v,a)      (Sig vertice)
8. Retornar suma                   (Retornar el costo total del camino recorrido)
```

5.1.3 Casos Generales

Por lo general esta estrategia al tratarse de casos donde solo el algortimo va por los caminos más cercanos, no siempre encontrará el camino más óptimo, incluso puede encontrar el peor, **en conclusión el algoritmo greedy no es una alternativa de solución para encontrar el camino óptimo.**

5.2 Estrategia Alternativa 2: Algoritmo de Dijkstra

5.2.1 Aplicación Dijkstra

El algortimo de Dijkstra encuentra el camino más cercano entre 2 nodos, en este caso particular (TSP), se hace una modificación, al recorrer todos los nodos debe regresar al nodo inicial. La secuencia quedaría:

- Verifica que el grafo es conexo y ponderado simple.

- Aplica el algortimo de Dijkstra, desde el nodo inicial "X" ingresado por el usuario hasta que todos los nodos estén recorridos.

- Encuentra el camino más corto hacia el nodo inicial "X".

5.2.2 Resolución del Ejemplo(Figura 1)

Group/Vertex	**A**	**B**	**C**	**D**
[A]	∞	4	15	**2**
[A, D]	-	22	**20**	-
[A, D, C]	-	**30**	-	-
[A, D, C, B]	**34**	-	-	-

Tabla 1: Representación de el algortimo Dijkstra ejecutandose sobre el Ejemplo(Figura 1).

5.2.3 Complejidad Algortimo Dijkstra

Según Wikipedia:
Teorema: El algoritmo de Dijkstra realiza $\boldsymbol{O(n^2)}$ *operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo y no dirigido con n vértices.*
Luego, para hallar la complejidad completa, hace falta encontrar la complejidad de extraer el mínimo y disminuir la clave.

Tiempo de ejecución = $O(|A|.\boldsymbol{T\_dk}+|v|.\boldsymbol{T\_dm})$

|A|: Número de aristas

$\boldsymbol{T\_dk}$: Complejidad de disminuir clave

|V|: Número de vértices

$\boldsymbol{T\_dm}$: Complejidad de extraer mínimo

.

6 Aprendizaje y Conclusiones

En algún momento de nuestra vida, nos enfrentaremos a este problema y curiosamente la primera solución que a una persona común se le ocurriría sería recorrer el camino más cercano a su punto de destino o el de menor costo primero, antes de evaluar un posible recorrido, y curiosamente, este método pertenece a un recorrido por el algoritmo **Greedy**, el cual no encuentra siempre el camino óptimo, por lo tanto es importante recalcar, que es conveniente conocer los métodos que se podrían utilizar antes de realizar un recorrido y el diseño de algortimos con otros métodos más eficientes cómo el **BackTracking con o sin poda**, podemos encontrar soluciones óptimas. Finalmente concluyo que este problema tiene mejores alternativas de solución ya que para un mayor números de nodos el costo o complejidad **Big O notation** es mayor y puede ser ineficiente, pero para los conocimientos abarcados en este curso, logramos mostrar alternativas de solución válidas al presente problema.

7 Fuentes

Definición Formal del problema:

https://en.wikipedia.org/wiki/Travelling_salesman_problem
https://www.gestiondeoperaciones.net/programacion-entera/solucion-del-problema-del-vendedor-viaj
https://es.wikipedia.org/wiki/Problema_del_camino_m%C3%A1s_corto

Solución con BackTracking, Poda y Greedy:

https://www.youtube.com/watch?v=W16E5qzzCxU
https://www.youtube.com/watch?v=EutHYzkSo5Y
https://www.youtube.com/watch?v=XaXsJJh-Q5Y
https://bit.ly/3xqEI3f/

Conceptos importantes:

https://es.wikipedia.org/wiki/Camino_hamiltoniano
https://es.wikipedia.org/wiki/Optimizaci%C3%B3n_combinatoria
https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra
https://es.wikipedia.org/wiki/Grafo_ponderado

Código:
Encontrar el mínimo por descompocisión de una ayudantía:
https://bit.ly/3rz5nZe