Problema del Agente Viajero

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

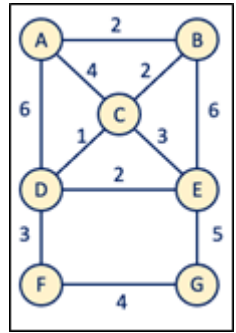
pperezm@tec.mx

08-2022

Contenido

El problema del agente viajero es que, dado un grafo ponderado y no dirigido, se deberá encontrar el ciclo Hamiltoniano con el menor costo, es decir, salir de un vértice, visitar todos los vértices restantes una sola vez y regresar al mismo punto de partida, con el menor costo.

- A-B-C-E-G-F-D-A con un costo de 25.
- A-B-E-G-F-D-C-A con un costo de 25.
- A-C-B-E-G-F-D-A con un costo de 30.
- A-C-D-F-G-E-B-A con un costo de 25.
- A-D-F-E-G-B-C-A con un costo de 30.
- A-D-F-E-G-C-B-A con un costo de 25.



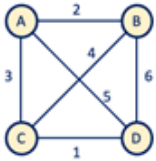
El problema de la Viajero al ser un problema de Selección con Optimización cumple con los requisitos para poder ser resuelto con la técnica de Branch & Bound. El árbol de búsqueda de soluciones tendrá n niveles, y cada nivel se trabajará un nodo a conectar, y cada nodo tendrá $n - i - 1$ hijos donde i es el nivel en que va, diciendo los posibles nodos que falten en esa ruta. Los nodos almacenarán el costo acumulado hasta el momento, así como su costo posible, el último nodo visitado, y el nodo actual. La idea es trabajar en anchura con Best-First, esto es se maneja una fila priorizada con prioridad menor costo posible y se saca al mejor y se generan sus hijos.

El cálculo del costo posible se realiza de la siguiente forma:

- El costo acumulado, más ...
- Del último vértice alcanzado, escoger el menor costo con los vértices faltantes, más ...
- Por cada vértice faltante, escoger el menor costo entre ellos y llegar al vértice inicial.

El criterio de selección del nodo será:

- Si el vértice en el nivel i del árbol, debe ser adyacente al vértice en el nivel $i-1$ del camino correspondiente en el árbol.
- Si el costo posible a acumular al expandir el nodo i, es menor al mejor costo acumulado hasta ese momento.



	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

1

A
CA = 0
Cpos = 6

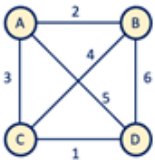
Costo pósible.
A -> min {B, C, D} = 2
B -> min {A, C, D} = 2
C -> min {A, B, D} = 1
D -> min {A, B, C} = 1
Total = 6
Adyacencia && Cpos > Costo óptimo

2

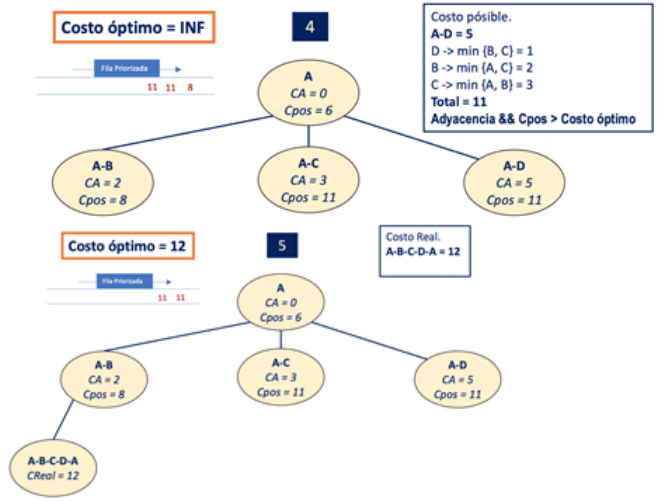
A
CA = 0
Cpos = 6

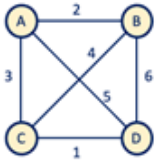
A-B
CA = 2
Cpos = 8

Costo pósible.
A-B = 2
B -> min {C, D} = 4
C -> min {A, D} = 1
D -> min {A, C} = 1
Total = 8
Adyacencia && Cpos > Costo óptimo

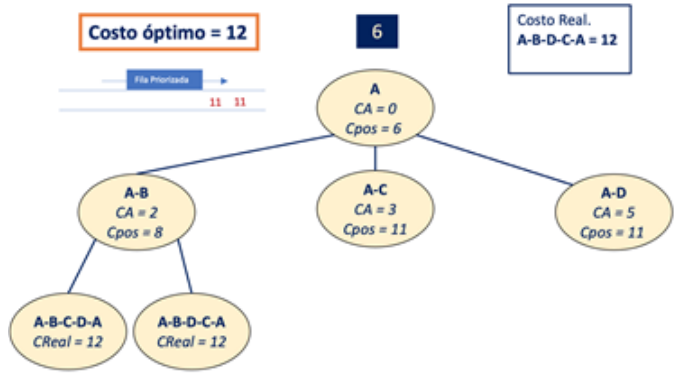


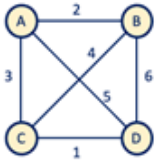
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



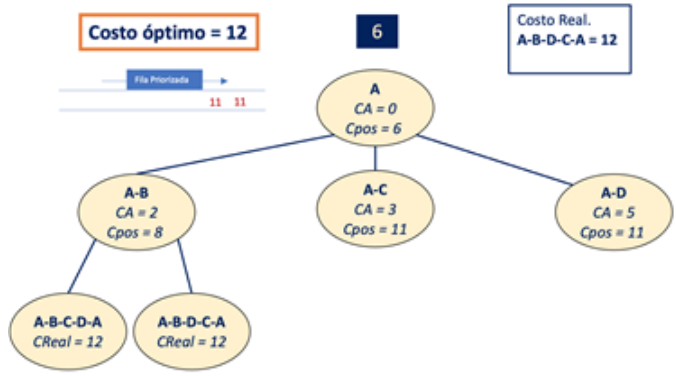


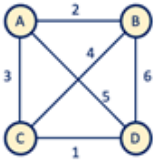
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



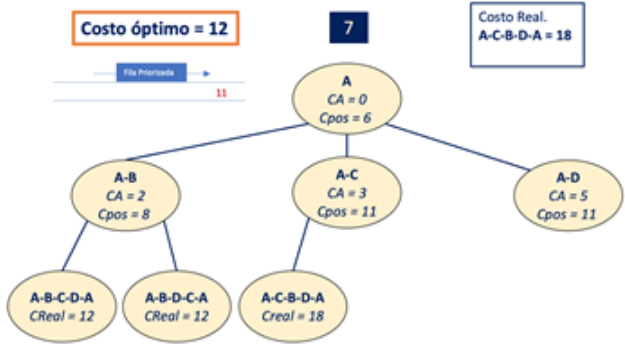


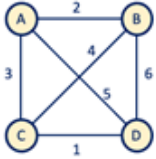
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



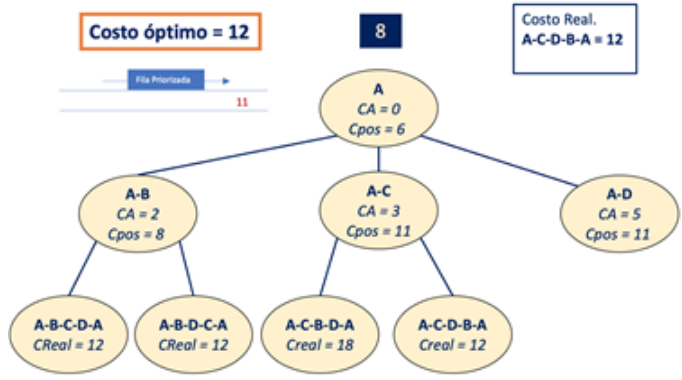


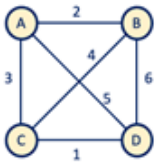
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



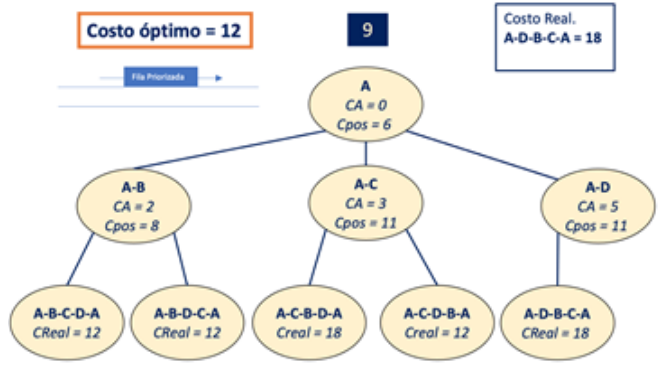


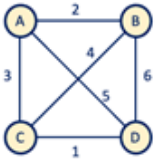
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



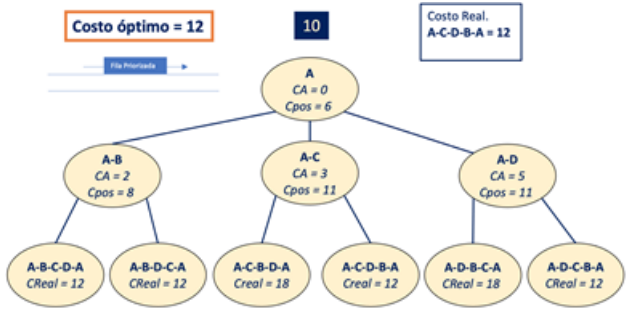


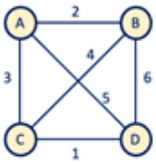
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



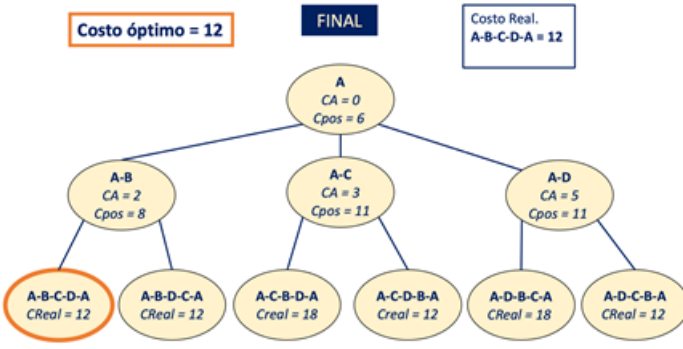


	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0





	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



Procedure 1 TSP

Input: *G* : *Graph*

```
Create a priority queue of nodes, where the priority is lowest possible cost.
Calculate the possible initial cost and put it in the priority queue.
while the priority queue is not empty do
  if is the possible cost less than optimal cost? then
    for each edge(u, v) coming out of v do
      if u is not visited then
        if is it the last connection? then
          Calculate actual accumulated value
          If the new real cost is better than the optimal cost, it is updated.
        else
          Put a new node with u in the priority queue.
        end if
      end if
    end for
  end if
end while
return the optimal value
```