

# Problema K-minimum spanning tree

Roberto Juan Cayro Cuadros, Gabriel Alexander Valdivia Medina, Giulia Alexa Naval Fernández, Rodrigo Alonso Torres Sotomayor

*Universidad Católica San Pablo*

---

## Resumen

El presente trabajo presenta una breve investigación del problema *k-minimum spanning tree*, explicando su funcionamiento, demostrando que pertenece al conjunto de los problemas NP-completos, y dando opciones de algoritmos para su resolución.

---

## 1. Introducción al problema

Según múltiples fuentes[? ][? ], el k-MST o k-minimum spanning tree problem, árbol de expansión de peso mínimo k en español es un problema computacional que pide un árbol de mínimo costo con exactamente  $k$  vértices que forme un subgrafo del grafo original.

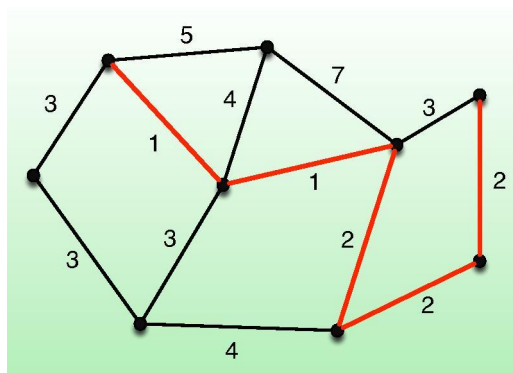


Figura 1: 6-MST del grafo G. Fuente: Wikipedia Commons

## 2. Demostración NP-completo

No es posible suponer la naturaleza del problema, y establecer que es NP-hard o NP-completo, sin la evidencia correspondiente, para probarlo este debe pertenecer a NP, además que un problema NP-completo pueda reducirse al mismo.

### 2.1. Demostrar que $k\text{-MST} \in NP$

Para demostrar que un problema pertenece a la clase NP, se debe crear un algoritmo no determinístico que resuelva el problema en tiempo polinomial:

$k\text{-MST}(G, k)$

```
1. x ← árbol. «“HEAD
2. for t ← 0, to k
3.     do u ← ESCOGER(G)
4.     if u is not in x
5.         do x . add(u)

    »”=====
6. t ← 0
7. while t < k
8.     do u ← ESCOGER(G)
9.     if u is not in x
10.        do x . add(u)
11.        t++
```

»X será el árbol a construirse, junto con el bucle while y una variable t confirmaremos la adición de exactamente k vértices, escogeremos algún  $u$  de  $G$ , si  $u$  ya está dentro de  $x$  el valor de  $t$  no cambie por lo tanto se repite hasta tomar otro vértice. »

### 2.2. Transformación NP-completo $\alpha k\text{-MST}$

El segundo paso para demostrar que un problema pertenece a los NP-completos, es transformar un problema NP-completo conocido para que pueda ser resuelto por el algoritmo del  $k\text{-MST}$ . Una transformación sencilla es la que se puede hacer desde el problema de Steiner.

### 2.2.1. Steiner problem

Según el artículo de Shivam Gupta[? ], el Steiner problem es un problema NP-completo de los 21 problemas de Karp, usado en problemas de optimización y mayormente enfocado en estructuras de grafos aunque tambien visto en aplicaciones de modelación de redes con más de 2 terminales. El problema consiste en que, dado un grafo no-dirigido de aristas con peso, generar un arbol dado un *Sub-set* de vertices los cuales formarán este arbol. Además, pueden añadirse nuevos vertices del grafo al *sub-set* para lograr las conexiones entre estos, llamados *Steiner-vertices*.

La decisión asociada al problema será averiguar si existe un árbol que una todos los vértices de un *sub-set*  $R$ , usando máximo  $M$  aristas. Los vertices deberán ser exactamente los dados en el Sub-set. Esta decisión es conocida por ser del grupo de los NP-completos. La principal diferencia con el k-MST es que aquí recibimos un conjunto específico de vectores para conformar nuestro árbol, pudiendo usar vértices fuera de la relación para conectarlos. El k-MST no recibe esta relación, sólo el número de vértices exactos que necesita.

### 2.3. Entradas y salidas

«“¡HEAD =====

»”Steiner-tree

»”””»¡b31219c1695ae33080e9348ba7482411382272ae

Steiner-problem
<p><i>Entrada:</i></p> <ul style="list-style-type: none"> <li>*Grafo no-dirigido <math>G</math> con aristas de peso.</li> <li>*Sub-set de vertices <math>R</math>.</li> <li>*Número <math>M</math>.</li> </ul>
<p><i>Salida:</i></p> <ul style="list-style-type: none"> <li>*Arbol de menor peso con los vertices de <math>S</math> y los Steiner-vertices si fueran necesarios.</li> </ul>

### $k$ -MST

k-MST
<i>Entrada:</i> *Grafo no-dirigido G con aristas de peso. *Número k de vértices.
<i>Salida:</i> *Árbol de menor peso con k-vertices y k-1, aristas.

#### 2.3.1. Transformación

Una primera aproximación será que dada la entrada G para Steiner, se puede tomar el mismo grafo para k-MST, puesto que tiene las aristas pesadas y un número determinado de vertices. De esta forma aseguramos la transformación y no afectara la salida porque siempre busaremos el arbol de menor peso , se usará el tamaño del sub-set de vertices siendo este igual a k.

Pero no podemos asegurar que esta transformación pudiera también resolver al Steiner tree, siendo esta una de las propiedades en una transformación polinómica. Como tenemos de entrada un G, y k, podriamos calcular todas las permutaciones de G en k. Y necesariamente una de ellas corresponderia a la solución para Steiner:

$$\text{Total de permutaciones (Tn)} = \binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

$$S \subseteq Tn.$$

Sin embargo, calcular todas las permutaciones de una cantidad  $n$  de elementos es un proceso con una complejidad  $O(!n)$ , que no entra dentro de complejidad polinomial, además que esta reducción planteada no resolverá el problema de k-MST, ya que este necesitaría solo 1 árbol de menor peso. Es necesario entonces otro tratamiento para que el k-MST opere con los vértices que el algoritmo Steiner pide. Siguiendo la transformación de R. Ravi [? ], otra idea es añadir un árbol con aristas de peso 0 en cada vértice que pertenezca a R, y transformar  $k$  como  $k = |R|(X + 1)$ , siendo X la cantidad de vértices que tendrán cada uno de estos árboles, denotado como

$X = |V(G)| - |R|$  De esta forma, el k-MST utilizará los vértices de  $R$  sí o sí como parte de su solución.

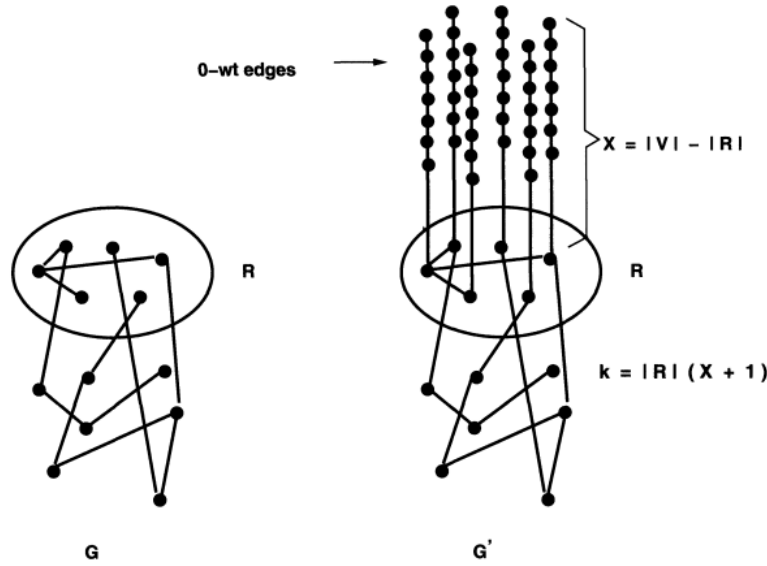


Figura 2: Transformación de la entrada de Steiner a entrada de k-MST. Fuente: [www.contrib.andrew.cmu.edu](http://www.contrib.andrew.cmu.edu)

En este nuevo grafo  $G'$ , las aristas que unen los nuevos vértices de  $X$  tendrán un peso de 0, las aristas correspondientes a las aristas originales de  $G$  tendrán un peso de 1, y el resto de pares del grafo tendrán un peso de  $\infty$ . De este modo, el algoritmo del k-MST encontrará el árbol de menor peso con el parámetro  $k$  en  $G'$ , y verificará si es de igual o menor peso que  $M$ , satisfaciendo el requisito de las  $M$  aristas debido a que estas tendrán peso 1.

Por otra parte cumpliremos la propiedad de la transformación polinomial, en donde redujimos el problema de Steiner a k-MST, por ello la solución a k-MST mediante otra transformación polinómica podrá ser la solución a Steiner-tree, gracias al valor de la transformación de  $k = |R|(X + 1)$ , puesto que las aristas de los árboles agregados son de peso 0, estos serán agregados a la solución y por el  $k$  se confirma que en la solución de k-MST estarán los vértices de  $R$ .

### **3. Algoritmo de fuerza bruta**

Para el algoritmo de fuerza bruta, es suficiente una modificación al algoritmo Prim convencional, limitando su avance a  $k$  nodos y haciendo que se repita con cada nodo del grafo  $G$  como origen. Finalmente, decidir qué árbol de todos los obtenidos ha sido el más corto.

```

1 #include <iostream>
2 #include <queue>
3 #include <vector>
4 using namespace std;
5 typedef vector<int> valNode;
6 typedef vector<valNode> adyacencias;
7
8 int PrimsMST(int sourceNode, vector<adyacencias>& graph, int
9 K)
10 {
11     //Guardar detalles del nodo.
12     priority_queue<valNode, vector<valNode>, greater<valNode
13 >> k;
14     int count = 0;
15     vector<int> aux = { 0,sourceNode };
16     k.push(aux);
17     bool* nodesAdded = new bool[graph.size()];
18     memset(nodesAdded, false, sizeof(bool) * graph.size());
19     int mst_tree_cost = 0;
20
21     while (count != K)
22     {
23         // Nodo con m nimo costo
24         valNode itemNode;
25         itemNode = k.top();
26         k.pop();
27         int Node = itemNode[1];
28         int Cost = itemNode[0];
29
30         //Checar si el nodo ya se a adi
31         if (!nodesAdded[Node])
32         {
33             mst_tree_cost += Cost;
34             count++;
35             if (count == K)
36                 break;
37             nodesAdded[Node] = true;
38
39             //Nodos vecinos quitados de priority queque
40             for (auto& node_cost : graph[Node])
41             {
42                 int adjacency_node = node_cost[1];
43                 if (nodesAdded[adjacency_node] == false)
44                 {
45                     k.push(node_cost);
46                 }
47             }
48         }
49     }
50 }

```

```

44         }
45     }
46 }
47 }
48 delete[] nodesAdded;
49 return mst_tree_cost;
50 }
51
52
53 int main()
54 {
55     adyacencias fromNode_0_in_graph_1 = { {1,1}, {2,2},
56     {1,3}, {1,4}, {2,5}, {1,6} };
57     adyacencias fromNode_1_in_graph_1 = { {1,0}, {2,2}, {2,6}
58     };
59     adyacencias fromNode_2_in_graph_1 = { {2,0}, {2,1}, {1,3}
60     };
61     adyacencias fromNode_3_in_graph_1 = { {1,0}, {1,2}, {2,4}
62     };
63     adyacencias fromNode_4_in_graph_1 = { {1,0}, {2,3}, {2,5}
64     };
65     adyacencias fromNode_5_in_graph_1 = { {2,0}, {2,4}, {1,6}
66     };
67     adyacencias fromNode_6_in_graph_1 = { {1,0}, {2,2}, {1,5}
68     };
69
70     int num_of_nodes = 7; // Total Nodes (0 to 6)
71     vector<adyacencias> primsgraph;
72     primsgraph.resize(num_of_nodes);
73     primsgraph[0] = fromNode_0_in_graph_1;
74     primsgraph[1] = fromNode_1_in_graph_1;
75     primsgraph[2] = fromNode_2_in_graph_1;
76     primsgraph[3] = fromNode_3_in_graph_1;
77     primsgraph[4] = fromNode_4_in_graph_1;
78     primsgraph[5] = fromNode_5_in_graph_1;
79     primsgraph[6] = fromNode_6_in_graph_1;
80
81     // As we already know, we have to choose the source
82     vertex,
83     // so we start from the vertex 0 node.
84     cout << "k-mst : " << PrimsMST(3, primsgraph, 3) <<
85     std::endl;
86     return 0;
87 }

```



#### 4. Algoritmo aproximado

#### Bibliography

#### Referencias

- [1] Gupta, S. (Junio de 2022). geeksforgeeks. Obtenido de <https://www.geeksforgeeks.org/steiner-tree/>
- [2] Matt Elder, S. C. (2007). CS880: Approximation Algorithms. Obtenido de <https://pages.cs.wisc.edu/~shuchi/courses/880-S07/scribe-notes/lecture26-2.pdf>
- [3] R. Ravi, R. S. (12 de Julio de 2006). Spanning Trees—Short or Small. Obtenido de SIAM (Society for Industrial and Applied Mathematics: <https://epubs.siam.org/doi/pdf/10.1137/S0895480194266331>
- [4] Wikipedia. (Junio de 2022). Wikipedia. Obtenido de <https://en.wikipedia.org/wiki/K-minimumspanningtree>