Counting the Number of Minimum Spanning Trees

Nathan James Runtuwene, 13514083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514083@std.stei.itb.ac.id

Abstract - One graph can have multiple minimum spanning trees. In this paper we review about one way to find the number of minimum spanning trees in a graph using Kruskal's Algorithm and also Kirchhoff's Matrix Tree Theorem. By considering all the edges of the same weight that could possibly be a part of one of the minimum spanning trees and counting the combination possibilities using Kirchhoff's Matrix Tree and then multiplying all the results, we could get the number of minimum spanning trees a certain graph has.

Minimum spanning tree, Kruskal Algorithm, Graph Theory, Kirchhoff's Theorem

I. Introduction

A spanning tree of a graph is a sub-graph that connects all the vertices and does not have any cycles or in other words a tree. The minimum spanning tree (MST) of a weighted graph is the spanning tree of that weighted graph with the minimum total weighting for the edges used in the MST. The MST has a few implementation in real life with the most common example being how to minimize the cost of building a communication network.

An MST has a few properties. One main property we are going to look at in this paper is that a graph could have more than one MSTs. So how many exactly are there on a certain graph? That is the question we are going to answer.

First we look at how to count the weight of an MST of a certain graph. There are a few algorithms that can be used to count the total cost of the MST of a graph. The more popular and commonly used ones are Kruskal's algorithm and Prim's Algorithm. Both of these algorithm are greedy algorithms and both have the time complexity of O(e log v) with e representing the number of edges of the graph and v representing the number of vertices in the graph.

II. Preliminaries

For the theorems used in this paper we are going to look at a few basic algorithms and data structures used in graph theory.

A. Adjacency Matrix

The adjacency matrix is square matrix used to represent a graph. It is only one of the many ways to represent a graph. For a simple graph with n vertices, the adjacency matrix is a matrix of size n x n with elements which consists of only 0s and 1s. $A_{i,j}$ is equal to 1 if there is an edge connecting vertex i to vertex j, and 0 otherwise.

Consider the following graph.

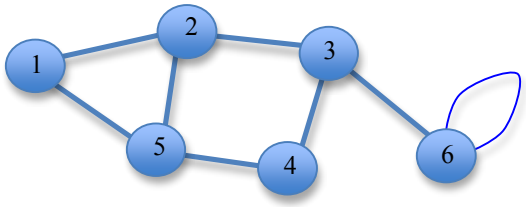

Fig. 2.1 Example of a graph

An adjacency matrix equivalent to the graph on Fig. 1 would be

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Fig 2.2 Adjacency matrix

B. Degree Matrix

The degree matrix is a matrix representing the degree of each of the nodes of a graph. If a graph has n nodes than the matrix will be of size n x n, and for every node i in the graph the cell $A_{i,I}$ will be the degree of that node. Every other cell will be filled with 0s, which means the matrix will only be filled on its diagonal.

The degree matrix corresponding to the graph on Fig 2.1 would be

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

Fig 2.3 Degree matrix

C. Incidence Matrix

An incidence matrix is a rectangular matrix by size v x e with v is the amount of vertex and e is the amount of edges. The incidency matrix is another way of representing a graph. The elements of an incidency matrix are determined as follows: for every edge k connecting vertex i to j and i<j then $A_{i,k}$=1 and $A_{j,k}$=-1, the rest of the matrix is filled with 0s.

The incidence matrix for Fig 2.1 without the self loop edge on node 6 is

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & -1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Fig 2.4 Incidence matrix

D. Laplacian Matrix

The Laplacian matrix (L) of a graph is defined as: $L = D - A$ where D is the degree matrix of the graph and A is the adjacency matrix of the graph.

The elements of the Laplacian matrix could also be defined as:

$$L_{i,j} = \begin{cases} \deg(v_j) & \text{if i = j} \\ -1 & \text{if i} \neq \text{j and } (v_i, v_j) \in \text{E} \\ 0 & \text{otherwise} \end{cases}$$

The Laplacian matrix is only to define a graph with no self-looping edge.

The Laplacian matrix can also be factored into the product of the incidence matrix of the graph and its transpose.

The Laplacian matrix equivalent to the graph in Fig 2.1 without the self-loop edge on node 6 is

$$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix}$$

Fig 2.5 Laplacian matrix

E. Kruskal Algorithm

The Kruskal algorithm finds the MST of a graph by first sorting all the edges of the graph by weight into an increasing order. Starting with an empty graph, for each edge starting from the smallest weighted edge try using that edge as a part of our MST, if taking an edge will not create a cycle in the already created graph then use it in the MST. An edge will create a cycle when used if its two endpoints are already connected by other edges that are already used in the graph. One way of checking whether the endpoints are already connected is to use the union set data structure.

F. Prim Algorithm

The Prim algorithm is similar with the Kruskal algorithm in terms of the greedy algorithm. It selects the smallest weighted edge available and uses it in the MST. The difference from Kruskal is that after selecting the first edge to be used in the MST, Prim checks the next smallest edge available from the edges connected to the already formed tree instead of from the whole set of edges.

III. Theorems

A. Kirchhoff's Matrix Tree Theorem

Kirchhoff's matrix tree theorem is a theorem about the number of spanning trees in a simple graph. The theorem states that the number of spanning trees in a simple graph is equal to the value of any cofactor in the graph's Laplacian matrix. Any cofactor of the Laplacian matrix is the same because by adding rows and columns, switching them around, and multiplying a row by -1 we can change one cofactor of the matrix into another.

IV. Counting Minimum Spanning Trees

A. Algorithm

Above we have shown how to calculate the cost of an MST using Kruskal's algorithm and how to count the amount of spanning trees in a graph using Kirchhoff's matrix tree theorem. Now we are going to combine those two to count the number of MSTs in a graph.

To do so we turn to another property of the MST, which is the costs of the edges selected will be the same.

Which means that if you want to change any edge from the MST you can only change it to another of the same weight that is not used in the current MST.

The Kruskal Algorithm rejects edges of a certain weight if the two vertices it connects are already connected by edges with weight smaller than or equal to its own weight. And thus if we want to insert this edge to the MST we have to remove the edge with the same cost as the current edge and is connected to the current edge via the spanning tree already placed. Which means that for all edges with the same weight there are as many possibilities to choose a set from those edges so that there are no cycles formed as the amount of spanning trees those edges could create.

With those conditions, the number of MSTs of a graph would be equal to the multiplication of the amount of spanning trees of connected edges with equal weight.

A more step by step instructions are:

First, we begin like we would on the Kruskal algorithm, initialize the union-data set, sort the edges by their weight, and start processing the edges from the smallest weight. After that, for every set of edges with the same weight we change the endpoint vertices of those edges to whatever union set those vertices are now in. Then, we create a graph using only those edges. After that, we find the connected components using graph traversal algorithms and for each connected component we find the number of spanning trees they have using the Kirchhoff matrix tree theorem and multiply them to the answer. Next, for every connected component you unite them in the union-data set.

V. Conclusion

In conclusion, the amount of minimum spanning trees in a graph can indeed be calculated, although with a time complexity relatively far slower than only counting the cost of the MST because we have to count the determinant of the Laplacian matrix at every unique edge cost. If we already know a limit to the maximum number of edges with the same cost this problem could most likely be solved in faster time by using normal combinatorics or by hard coding every possibility of the edges.

VI. Acknowledgment

First, I would like to thank God for the opportunity to make this paper. I would like to thank my parents and my family for their support since I am young. I would like to thank Mr. Rinaldi and Ms. Harlili for their guidance through this semester Discrete Mathematics class. And finally, I would also like to thank my friends for helping me out when I am having troubles with this paper.

References

[1] http://www14.informatik.tu-muenchen.de/konferenzen/Jass08/courses/1/pieper/Pieper_Paper.pdf, visited on December 9, 2015

[2] http://www.maths.manchester.ac.uk/~mrm/Teaching/DiscreteMaths/LectureNotes/IntroToMatrixTree, visited on December 9, 2015

[3] http://math.fau.edu/locke/Graphmat.htm, visited on December 9, 2015

[4] http://www.math.kun.nl/~bosma/Students/jannekebc3.pdf, visited on December 9, 2015

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Desember 2015

Nathan James Runtuwene / 13514083