Algorithms That Changed the World

Boruvkas Algorithm

Abstract: Boruvka algorithm is the first example found of an algorithm to find the minimum spanning tree of a graph. This report will investigate the Boruvka algorithm, including a particular focus on how it changed the world. This report will also demonstrate how the algorithm works looking at the pseudo code in a flow diagram and an example of how it works is given.

Student Number : **099562**

I certify that all material in this report which is not my own work has been identified

[illegible]

Why this algorithm is important?

Boruvka algorithm is used on a graph of connected vertices to find the minimum spanning tree. The minimum spanning tree is a subset of the edges of a connected and undirected graph that connects all the vertices together, without any cycles and with the minimum possible edge weight. Boruvka algorithm is important because it is the first algorithm created for finding a minimum spanning tree. It was developed by a Czech scientist named Otakar Borůvka, who is one of the most prominent Czech mathematicians of the 20th century. The algorithm was initially created in order to provide Moravia (a region in Czech Republic) with electricity in the most efficient way. The algorithm dates back to 1926 and it is the first algorithm developed to find the minimum spanning tree [3].

How it changed the world?

The practical problem that Boruvka faced was how to get electricity to all the towns in south Moravia. The problem can be stated as follows: there are several towns in the region, and the distances between pairs are known. The goal is to design a system of powerlines that would minimise the total distance, therefore the cost of construction, all whilst connecting every town. In this scenario it must be stated that it is not a requirement to connect each town to the source of electricity directly [1].

Being able to connect all nodes in a graph with the minimum weight has changed the world in terms of financial cost. Any scenario where there is a price to the weight of edges has benefitted from the minimum spanning tree algorithms. For instance, connecting computers in a network can be done in the most cost-efficient way thanks to minimum spanning trees, similarly connecting houses to a grid for electricity has also benefitted from minimum spanning trees.

An algorithm to find the minimal spanning tree is useful in clustering. If you want tot cluster many points into k clusters, an approach you could use is to first compute the minimum spanning tree and then drop the k-1 most expensive edges of this tree [2]. Dropping k-1 edges leaves you with K clusters. Although this method isn't used much in practice it is still a possible application of using minimal spanning trees that Boruvka pioneered.

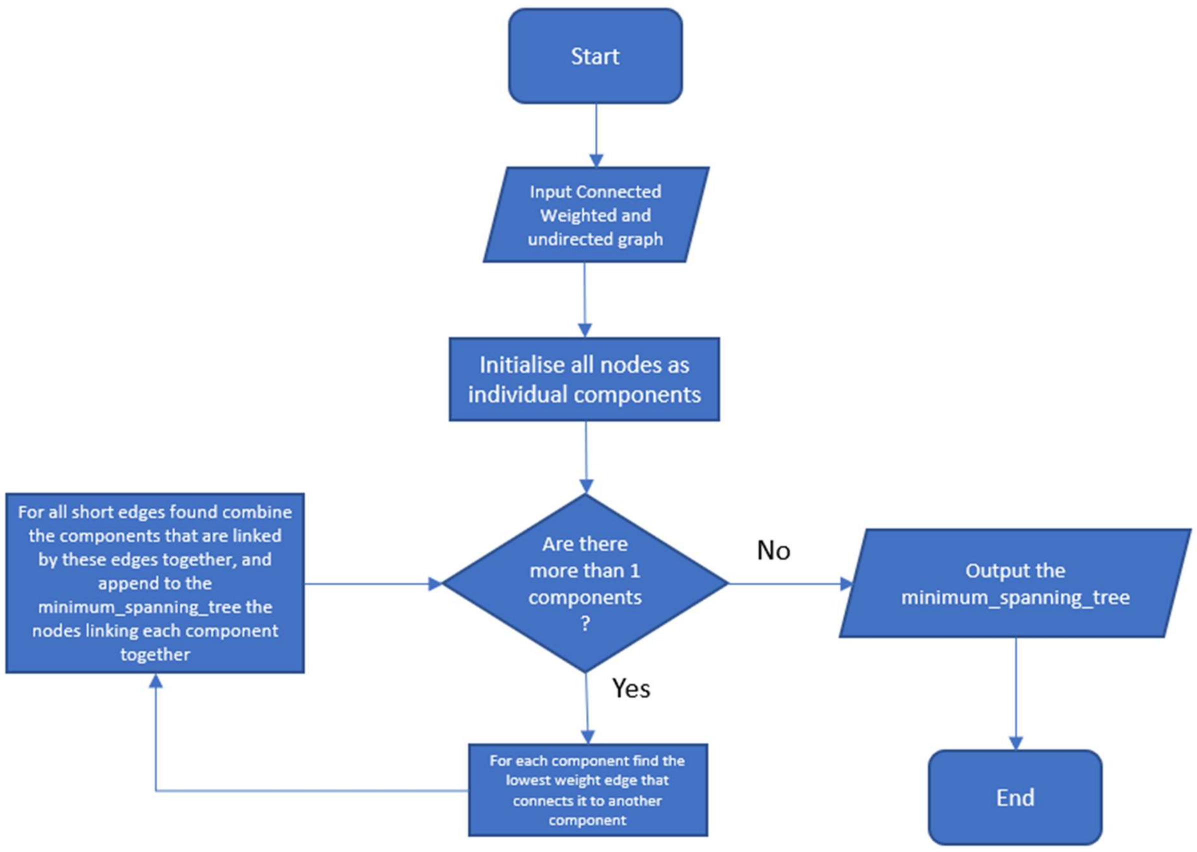
The traveling salesman problem is NP-hard, this means that there is no polynomial time algorithm that solves it. Instead we can use approximation algorithms, these include using a minimal spanning tree algorithm. Once the minimum spanning tree is found the tree is traversed in pre-order (Root, Left, Right). The root of the tree can be any point and so we choose the best of all possible solutions. This approximation technique yields results that are at most twice optimal, this seems poor however due to the speed at which this method performs means it can be used alongside other heuristics to get a better approximation.

The main principles of the algorithm

The main principles of this algorithm are to always find the cheapest tree that connects all the nodes together. In a fully connected graph, a tree is found every time, and it is guaranteed to be the cheapest. The input into the algorithm is a graph of connected nodes and the output is the order of nodes that are connected in the tree. In the outputted results there should never be a cycle else it's not a tree.

Boruvkas algorithm is a greedy algorithm. A greedy algorithm is a technique that algorithms use, it means that the algorithm always makes the choice that seems to be best at the moment, or in other words it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. One advantage of a greedy algorithm is that it is easy to analyse the run time

The pseudo-code to express the algorithm


Start
Input Connected Weighted and undirected graph
Initialise all nodes as individual components
Are there more than 1 components ?
No
Output the minimum_spanning_tree
End
Yes
For each component find the lowest weight edge that connects it to another component
For all short edges found combine the components that are linked by these edges together, and append to the minimum_spanning_tree the nodes linking each component together

Here is a Flowchart of how the algorithm works.

The following is how the algorithm works in words:

First of all, we must input into the algorithm a connected, weighted and undirected graph. The nodes of the graph are the initial components, at the beginning each component consists of one of the vertices. The Minimal_spanning_tree list needs to be initialised as empty.

While there is more than one component, do the following:

For each component.

1) Find the closest node that connects this component to another component, this is the smallest edge. A check is needed to see if the closest edge chosen is not to a node in the same component.
2) Add this closest edge to the Minimal_spanning_tree
3) Merge the components that are connected into a single component and update the component list.

Once there is just one component left return the Minimal_spanning_tree

This works because when all nodes are connected, they will all be in the same component.

The complexity (time and space) analysis

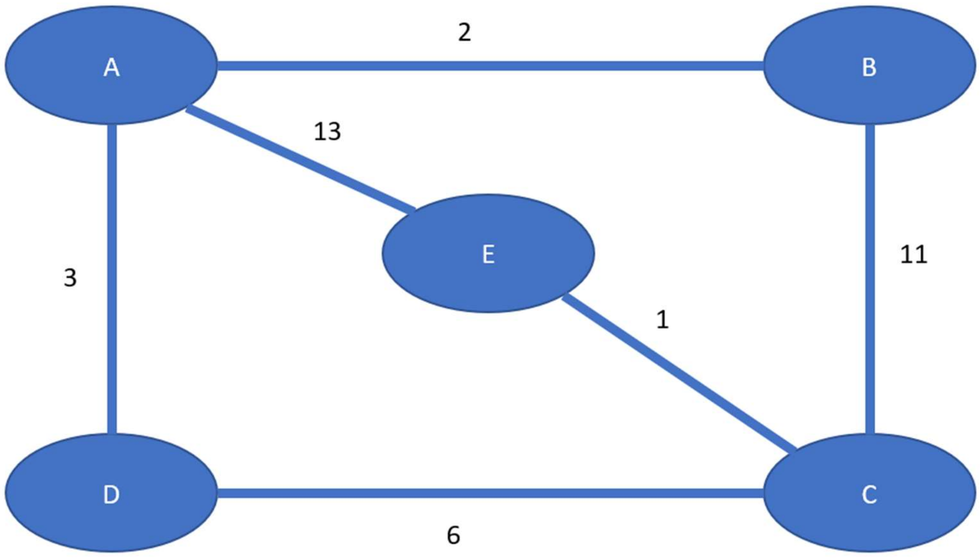
Boruvkas algorithm has been shown to take O(log V) iterations of the outer loop (longest loop in terms of time) until it terminates, for this reason it the time complexity of Boruvka's algorithm is O(E log V) [5]. Here E is the number of edges and V is the number of vertices in the connected input graph.

The space complexity of the algorithm is the total space taken by the algorithm with respect to the input size. Therefore the space complexity of this algorithm involves storing the edges and the vertices, and so the space complexity is $\Theta(E + V)$.

The limitations or constraints of the algorithm and the advances that could overcome them;

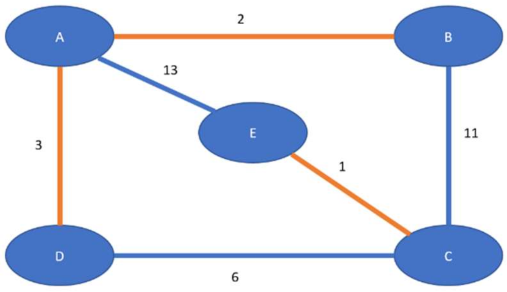
In terms of finding the minimum spanning tree there is no constraints or advances as the algorithm is proven correct and will always find the minimum spanning tree. Looking into the runtime of the algorithm there is a possibility of improvement, in this report I have shown the [6] sequential implementation which in a while loop looks through all edges, finds the shortest edges connecting two components, then it updates the components and the minimal_spanning_tree list. There could possibly be a way to do a parallel implementation of the algorithm, in source [6] we see an attempt of doing this. The parallel implementation shows an improvement over the sequential algorithm when finding the minimum edge using an edge by edge manner. The differences are only seen in large graph of 1000,000 nodes and 9,000,000 edges and so when you have smaller graphs the sequential implementation is preferred.

Example of how the algorithm works



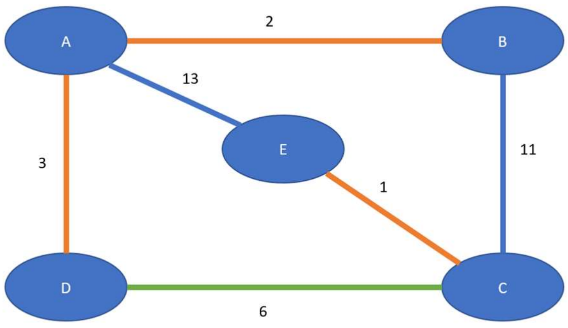
So in this example, the initial components will be {{A},{B},{C},{D},{E}}. After the first loop we will find the cheapest edges are:

{A} A-B	2
{B} A-B	2
{C} C-E	1
{D} A-D	3
{E} C- E	1



We therefore have the new components being: {{A,B,D}, {C,E}} considering that there is still more than 1 components, the while loop is still going, the next cheapest edges are:

{A,B,D} C-D	6
{C,E} C-D	6



Now the new components are {{A,B,D,C,E}}, the while loop ends now due to the fact that there is only one component.

The minimum spanning tree is {A-B,C-E,A-D, C-D}

References

[1] A. Kolokolova, "CS 6901 (Applied Algorithms) – Lecture 7", *Memorial University of Newfoundland*, 2014. [Accessed 9 December 2019].

[2] *Personal.utdallas.edu*, 2019. [Online]. Available: https://personal.utdallas.edu/~besp/teaching/mst-applications.pdf. [Accessed: 09- Dec- 2019].

[3] P. Jayawant and K. Glavin, "Minimum spanning trees", *Involve, a Journal of Mathematics*, vol. 2, no. 4, pp. 439-450, 2009. Available: 10.2140/involve.2009.2.439.

[4] J. Rangel-Mondragon, "The Traveling Salesman Problem 4: Spanning Tree Heuristic - Wolfram Demonstrations Project", *Demonstrations.wolfram.com*, 2012. [Online]. Available: https://demonstrations.wolfram.com/TheTravelingSalesmanProblem4SpanningTreeHeuristic/. [Accessed: 09- Dec- 2019].

[5] "Boruvka's algorithm | Greedy Algo-9 - GeeksforGeeks", *GeeksforGeeks*, 2018. [Online]. Available: https://www.geeksforgeeks.org/boruvkas-algorithm-greedy-algo-9/#:~:targetText=1)%20Time%20Complexity%20of%20Boruvka's,linear%20time%20O(E). [Accessed: 09- Dec- 2019].

[6] K. Fuh and S. Vemuri, "Parallelizing Bor&#367vka's Algorithm", *Kfuh1.github.io*, 2018. [Online]. Available: http://kfuh1.github.io/15418-project/. [Accessed: 09- Dec- 2019].