Borůvka's algorithm

Evan Alba

Introduction

Borůvka's algorithm is a greedy algorithm for finding a minimum spanning tree in a graph. It was published in 1926 by Otakar Borůvka, a Czech mathematician as a method of constructing an efficient electricity network for the South-Moravia district in the region Moravia, Czech Republic. [2] (Fun Fact: The first textbook on graph theory was written by Dénes Kőnig, and published in 1936. This algorithm was published way before graph theory was being taught interestingly.) Besides constructing an efficient electricity network for the South-Moravia district, another application of Borůvka's algorithm is used parallel computing literature, in which Borůvka's algorithm is typically refered as Sollin's algorithm due to the computer scientist George Sollin. In the first place using Borůvka's algorithm, we define our problem as follows:

Given a connected (undirected) graph $G = (V,\ E)$ with real weights assigned to its edges. Find a spanning tree $(V,\ T)$ of $G(i.e.\ T \subseteq E)$ with the minimal weight $w(T)$.

Intuition

The main intuition behind the algorithm is to simply iteratively build the minimum spanning tree by finding the minimum edges that connect different connected components in the graph. In other words, our intuition goes as follows:
1. Start with isolated vertices. (Initially, each vertex in the graph is considered an isolated component.)
2. Find the minimum edges. (During each iteration, we find the minimum edge that is adjacent to each component. Note: If we have a tie in the minimum edge, we pick the minimum edge we have seen first.)
3. Add the minimum edges to the minimum spanning tree. (Note: We add the minimum edges as long as they do not cycles.)
4. Unite Components. (Unite if possible 2 separate components if the minimum edge does not already it's vertex endpoints under 1 component united.)
5. Repeat until all are connected. (We repeat the process of finding minimum edges and add them to the minimum spanning tree until all vertices are connected in a single component in which contains the minimum spanning tree we are looking for.)

Therefore, by using Borůvka's algorithm to calculate the minimum spanning tree, we can get the minimum weighted cost for connecting to all the vertices. Additionally, if there is a forest disconnected graphs while using Borůvka's algorithm, it identifies the minimum spanning tree for each individual connected graph component. The outcome then becomes a forest collection of minimum-spanning trees.

Pseudocode and Detailed Description

```
function BORUVKA(G):
    INITIALIZE-SINGLE-COMPONENT(G)
    // Let A = Minimum Spanning Tree Set
    A = ∅
    while A does not form a spanning tree
        for each tree T in G.forest
            e = MINIMUM-WEIGHT-EDGE(T, G)
            A = A ∪ {e}
            if FIND-SET(e.u) ≠ FIND-SET(e.v)
                UNION(e.u, e.v)
    return A

function INITIALIZE-SINGLE-COMPONENT(G)
    for each vertex v in G.V
        MAKE-SET(v)

function MINIMUM-WEIGHT-EDGE(T, G):
    min_edge = NIL
    min_weight = ∞
    for each edge e in G.E
        if exactly one endpoint of e belongs to T
            v = other endpoint of e (not in T)
            if w(e) < min_weight
                min_edge = e
                min_weight = w(e)
    return min_edge

function MAKE-SET(x):
    x.p = x
    x.rank = 0

function FIND-SET(x):
    if x ≠ x.p // not the root?
        x.p = FIND-SET(x.p) // the root becomes the parent
    return x.p // return the root

function UNION(x, y):
    LINK(FIND-SET(x), FIND-SET(y))

function LINK(x, y):
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1
```

Note: The pseudocode of MAKE-SET, FIND-SET, UNION, and LINK come from the Introduction to Algorithms (CLRS) textbook. [1]

e.g. A visual of Borůvka's algorithm in action. [3]

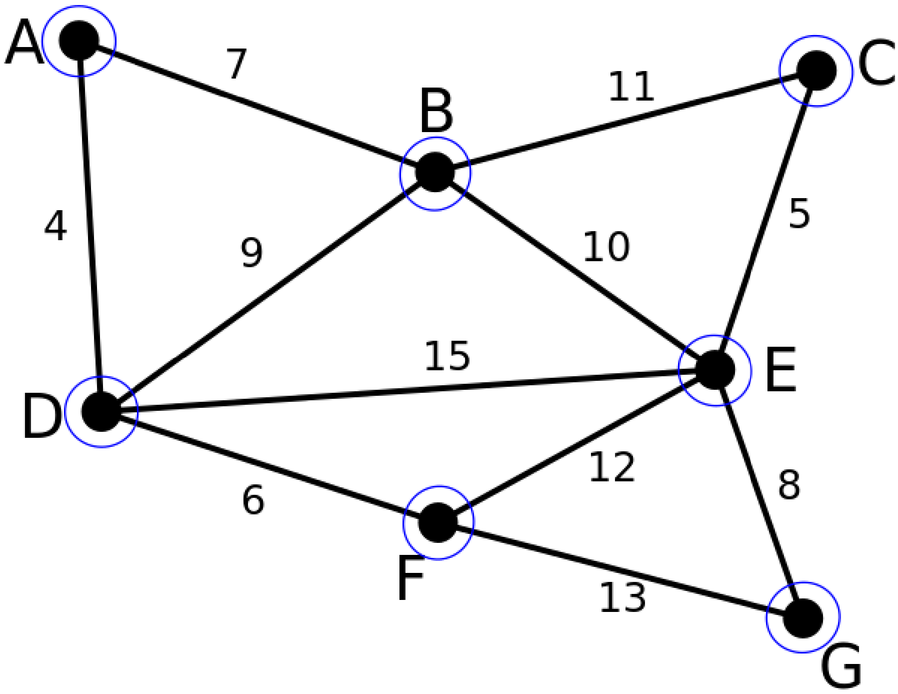

Figure 1: Components: {A}, {B}, {C}, {D}, {E}, {F}, {G}. Our original starting weighted graph where every vertex by itself is a component (blue circles) and every edge contains a weight.

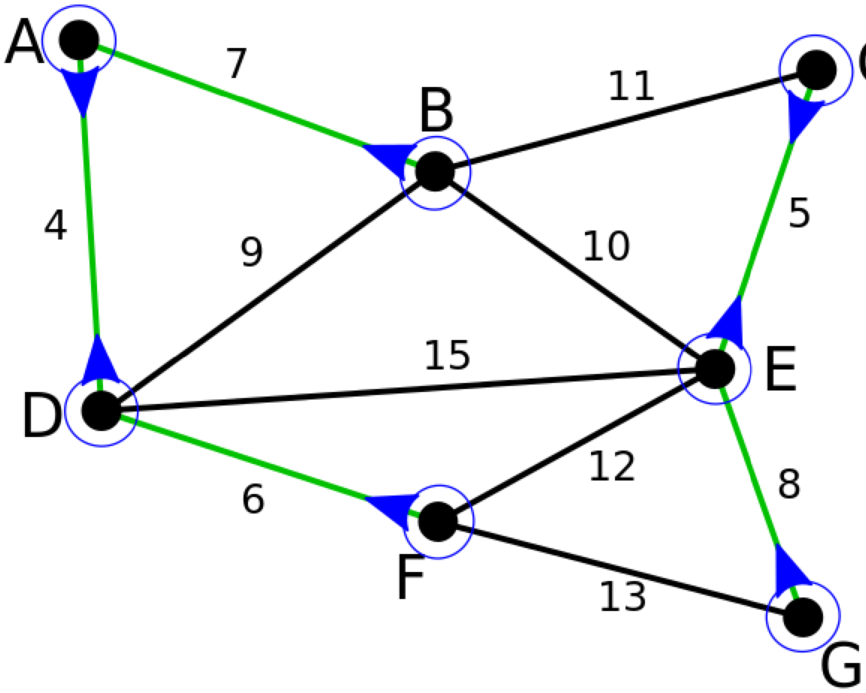

Figure 2: Components: {A, B, D, F}, {C, E, G}. During the first iteration of the outer loop, the edge with smallest weight adjacent of every component is added to the minimum spanning tree. Note: Some edges may be selected twice such as (AD, CE). If a minimum edge end point vertices do not belong to the same component, unite them into a component together. After this 1st iteration, we have not finished Borůvka's algorithm due two components remaining, we need to have only 1 big component in which it contains the minimum spanning tree of the graph given.

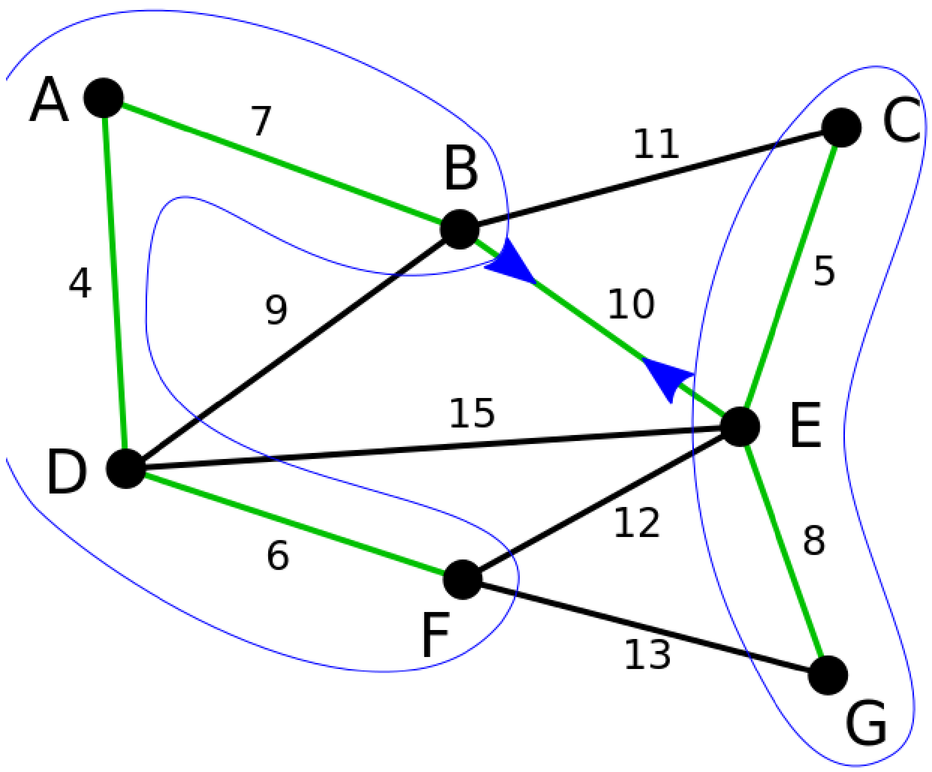

Figure 3: Components: {A, B, C, D, E, F, G}. In the second and final iteration, the minimum weighted edge out of each of the two remaining components is added; coincidentally, it is the same one (B, E). Afterwards, one component remains, and we are done. Note: The edge (B, D) is not considered because both vertex endpoints are in the same component.

Run Time Analysis

Table 1: Empirical study of Minimum Spanning Tree algorithms

Sample sizes of N vertices:	**10**	**100**	**123**	**146**
Boruvka				
Barabasi-Albert Random Graph 1	2.43e-5	0.000138216	9.14e-5	7.88e-5
Barabasi-Albert Random Graph 2	1.51e-5	0.000110803	1.04e-4	0.000138769
Barabasi-Albert Random Graph 3	6.57e-6	0.000110668	6.80e-5	0.000108274
Barabasi-Albert Random Graph 4	1.58e-5	6.97e-5	7.07e-5	1.13e-4
Barabasi-Albert Random Graph 5	1.07e-5	0.000101907	0.000148186	8.29e-5
	Average:	Average:	Average:	Average:
	1.45e-5	0.0001062612	9.64e-5	1.04e-4
Kruskal				
Barabasi-Albert Random Graph 1	1.66e-5	0.000154373	0.000136166	0.000133298
Barabasi-Albert Random Graph 2	1.04e-5	0.000153346	0.000159567	0.000201087
Barabasi-Albert Random Graph 3	6.57e-6	0.000131004	0.000120207	0.000194174
Barabasi-Albert Random Graph 4	9.77e-6	0.000132552	0.000145113	0.000192999
Barabasi-Albert Random Graph 5	1.65e-5	0.000153909	0.000186299	0.000139383
	Average:	Average:	Average:	Average:
	1.20e-5	0.0001450368	0.0001494704	0.0001721882
Prim				
Barabasi-Albert Random Graph 1	4.10e-5	0.00113051	0.00149131	0.00153231
Barabasi-Albert Random Graph 2	1.67e-5	0.00119967	0.00160627	0.00203915
Barabasi-Albert Random Graph 3	1.61e-5	0.0010315	0.00126785	0.00213344
Barabasi-Albert Random Graph 4	2.12e-5	0.00113705	0.00134214	0.00214334
Barabasi-Albert Random Graph 5	2.60e-5	0.00112834	0.00129528	0.00168224
	Average:	Average:	Average:	Average:
	2.42e-5	0.001125414	0.00140057	0.0019060960000000001
Boruvka				
Erdos-Renyi Random Graph 1	9.29e-6	0.000109706	0.000105559	0.000157308
Erdos-Renyi Random Graph 2	7.96e-6	9.75e-5	0.000103243	0.00013404
Erdos-Renyi Random Graph 3	7.89e-6	0.000107925	0.000127167	0.000159334
Erdos-Renyi Random Graph 4	4.59e-6	0.000149366	0.000125105	0.000156576
Erdos-Renyi Random Graph 5	9.68e-6	8.87e-5	0.000131142	0.000182932
	Average:	Average:	Average:	Average:
	7.88e-6	0.0001106292	0.0001184432	0.00015803800000000002
Kruskal				
Erdos-Renyi Random Graph 1	7.21e-6	0.000188983	0.000276763	0.000342615
Erdos-Renyi Random Graph 2	5.69e-6	0.000183724	0.000228981	0.000340827
Erdos-Renyi Random Graph 3	6.13e-6	0.000168772	0.000299339	0.000436534
Erdos-Renyi Random Graph 4	3.12e-6	0.000231831	0.000240922	0.000391516
Erdos-Renyi Random Graph 5	7.07e-6	0.0001753	0.000259229	0.000471722
	Average:	Average:	Average:	Average:
	5.85e-6	0.00018972199999999997	0.00026104679999999997	0.0003966428
Prim				
Erdos-Renyi Random Graph 1	1.80e-5	0.00113712	0.00133722	0.00161312
Erdos-Renyi Random Graph 2	1.03e-5	0.00118436	0.00118496	0.00175618
Erdos-Renyi Random Graph 3	9.99e-6	0.00112847	0.00134836	0.00180173
Erdos-Renyi Random Graph 4	8.09e-6	0.00122451	0.0012377	0.00164951
Erdos-Renyi Random Graph 5	2.21e-5	0.00122675	0.00135016	0.00191823
	Average:	Average:	Average:	Average:
	1.37e-5	0.0011802420000000002	0.00129168	0.001747754
Boruvka				
Random Tree 1	1.10e-5	4.79e-5	5.33e-5	6.67e-5
Random Tree 2	5.55e-6	4.20e-5	5.51e-5	7.64e-5
Random Tree 3	1.15e-5	2.91e-5	6.80e-5	6.17e-5
Random Tree 4	6.72e-6	2.63e-5	5.73e-5	6.98e-5
Random Tree 5	6.62e-6	3.09e-5	5.54e-5	7.74e-5
	Average:	Average:	Average:	Average:
	8.28e-6	3.52e-5	5.78e-5	7.04e-5
Kruskal				
Random Tree 1	8.00e-6	6.70e-5	6.05e-5	7.04e-5
Random Tree 2	4.63e-6	5.09e-5	6.12e-5	7.38e-5
Random Tree 3	9.86e-6	3.60e-5	8.14e-5	6.21e-5
Random Tree 4	5.41e-6	3.25e-5	6.80e-5	0.000104344
Random Tree 5	1.59e-5	3.86e-5	6.12e-5	7.34e-5
	Average:	Average:	Average:	Average:
	8.75e-6	4.50e-5	6.64e-5	7.68e-5
Prim				
Random Tree 1	2.02e-5	0.00113064	0.00145761	0.00177217
Random Tree 2	9.12e-6	0.00102145	0.00162498	0.00225912
Random Tree 3	2.44e-5	0.000939006	0.00138988	0.00188811
Random Tree 4	1.56e-5	0.00107853	0.00145543	0.00182982
Random Tree 5	2.93e-5	0.000858781	0.00137783	0.00207076
	Average:	Average:	Average:	Average:
	1.97e-5	0.0010056814000000002	0.001461146	0.001963996

Table 2: System Specifications

Operating system: Ubuntu 22.04.4 LTS
CPU: Intel(R) Core(TM) i5-5250U CPU @ 1.60GHz
Memory: 16GiB

The runtime for Borůvka's algorithm is $O(|E|log|V|)$. A faster randomized minimum spanning tree algorithm based on Borůvka's algorithm due to Karger, Klein, and Tarjan runs in expected O(E) time. The best known (deterministic) minimum spanning tree algorithm by Bernard Chazelle is also based on Borůvka's algorithm and runs in $O(E\alpha(E,V))$ time, where α is the inverse Ackermann function (a very slowly growing function.).[3]

Feedback

Speaker: Erik Stenstrom

Reviewer 1 (Eric Galvan):
- Great examples that effectively communicate the algorithm's purpose, tic-tac-toe as an example.
- The algorithm makes use of the min-max algorithm so it leverages some of the knowledge we used in class to make it easier to follow. It was not confusing. - It's an especially useful algorithm in the sense that it helps the computer always pick the most optimal move in a game of tic-tac-toe.

Reviewer 2 (Evan Alba): I like that Erik used images to explain how the mini-max algorithm works internally using a great example of the game tic tac toe. I was a bit confused on where the +10 or -10 plays a role in the pseudocode. Maybe he should add comments to help indicate better in the pseudocode. The general idea behind the algorithm is to calculate the best move so we can win in tic-tac-toe.

Reviewer 3 (Erik Burgess): I think using the algorithm for tic tac toe was a great way to explain how the algorithm works. It was clear what the algorithm is doing and what it is trying to accomplish. The asymptotic runtime is a little unclear at this time but it appears to be $O(mn)$.

Speaker: Evan Alba

Reviewer 1 (Eric Galvan):
Gave lots of great comparisons to algorithms we studied in class like Djikstras and Kruskal's which made it easy to follow. A lot of pseudocode to go through and more visualization to pair would help. This algorithm is a great alternative to finding a minimum spanning tree. It was mentioned that it was created before graph theory existed so it probably works great as a base for legacy research.

Reviewer 2 (Erik Burgess):
I feel the example was the most effective at explaining how the algorithm works. It appears to be a different way to find minimum spanning tree similar to prims or djikstras. The asymptotic runtime is $O(Elog(V))$.

Reviewer 3 (Erik Stenstrom):
The example was really good to understand how it works, it made it very clear and easy. Runtime is the same as prims which makes sense, more explanation of the pseudo code would be nice.

Speaker: Eric Galvan

Reviewer 1 (Evan Alba):
I like how you showed the difference using images the difference between jump point search and A* algorithm. I wish you did a bit more research on the runtime analysis on the difference between why the jump point is important compared to the A* algorithm. The goal of the algorithm is to find the shortest path between two points, one being the goal.
Reviewer 2 (Erik Burgess):
This algorithm searches for "interesting nodes" and searches these nodes neighbors for a end goal. I feel the illustrations helped with the understanding quite a bit. Has a runtime of O(n). **Reviewer 3 (Erik Stenstrom):**
The visual of how the algorithm searches was really helpful in understanding how it works, it is a very interesting algorithm in how it does find the shortest path between two points, especially with the interesting points. I am still a bit confused on how the algorithms decision making.

Speaker: Erik Burgess

Reviewer 1 (Eric Galvan):
Finds a matching substring in O(n+m) runtime. A good point of reference, for example other algorithms for comparison, would be helpful.
Reviewer 2 (Evan Alba):
Z Algorithm. It stores the longest substring using an auxiliary array called Z. I wish you went more in depth on the runtime analysis on why this is important compared to other algorithms we might find out there. I recommend you add comments on your pseudocode for your algorithm so you can help clarify the algorithm and the important steps it takes.
Reviewer 3 (Erik Stenstrom):
Longest substring algorithm called z algorithm, the example helped me make sense. It would be nice if you talked about the differences and benefits/drawbacks of this compared to other string matching algorithms.

Feedback Explanation

From the feedback, I improved the explanation of the pseudocode. Originally in the presentation, I only featured pseudocode of Borůvka's function and not of the function calls made in Borůvka's function. Now, what I did was add more pseudocode in detail on what goes behind the scenes in each function call we call in Borůvka's pseudocode function.

References

[1] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.

[2] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. "Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history". In: *Discrete Mathematics* 233.1 (2001). Czech and Slovak 2, pp. 3–36. ISSN: 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(00)00224-7. URL: https://www.sciencedirect.com/science/article/pii/S0012365X00002247.

[3] Wikipedia contributors. *Borůvka's algorithm*. URL: https://en.wikipedia.org/wiki/Bor%C5%AFvka's_algorithm.