Borůvka's Algorithm

Student Number: 690065435

December 2022

Abstract

The minimum spanning tree problem is a fundamental problem in graph theory that has many applications in the real world. In this report, we explore Borůvka's algorithm – a greedy algorithm that finds a minimum spanning tree for a connected, edge-weighted undirected graph. We discuss the algorithm's principles, pseudocode, time and space complexity analysis, and applications. We compare Borůvka's algorithm to other minimum spanning tree algorithms such as Prim's and Kruskal's algorithms. We also review variants of Borůvka's algorithm and their use as two-approximation for the travelling salesperson problem, parallel computation of minimum spanning trees, and faster sequential algorithms for minimum spanning trees.

YouTube Video Link:

Word Count: 1,500

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

Signature: _____

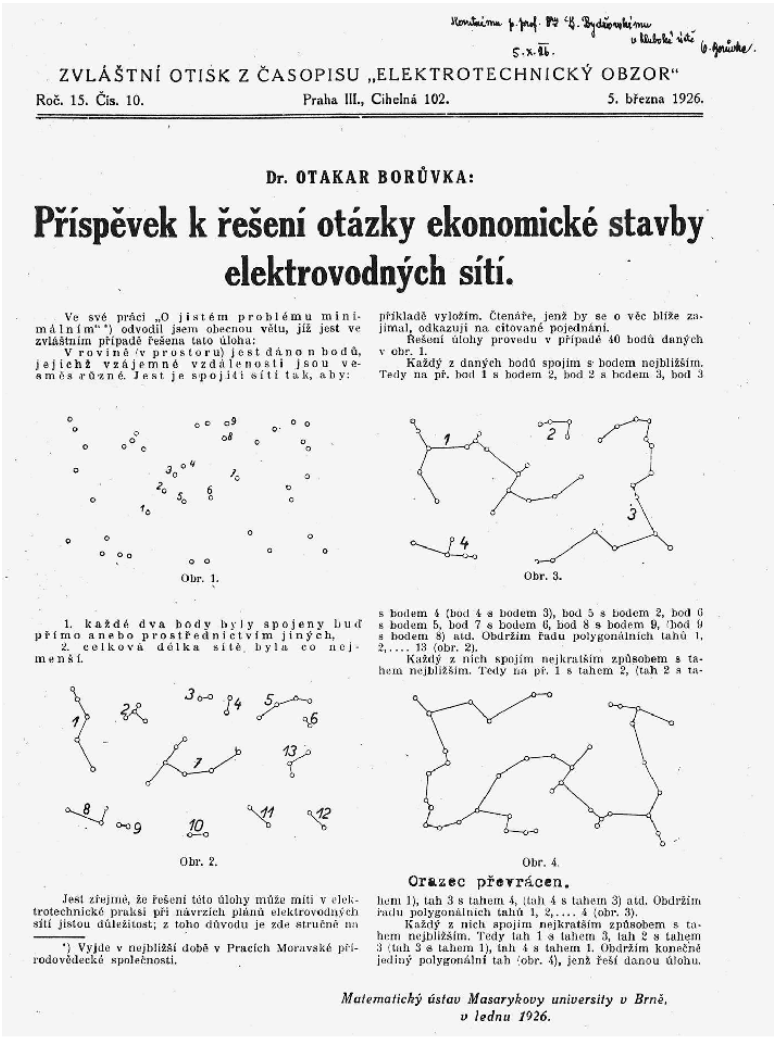
1 Principles of Borůvka's Algorithm

Borůvka's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected, edge-weighted undirected graph. It originated from Otakar Borůvka in 1926, as a method of constructing an efficient electricity network for Moravia, a region of the Czech Republic [1]. At the point of creation, computers as we know them today did not exist, so the algorithm was implemented by hand – this explains why the algorithm is simpler and uses basic data structures compared to other minimum spanning tree algorithms such as Prim's and Kruskal's algorithms. Borůvka's algorithm was the first to solve the minimum spanning tree problem in polynomial time [2], which was significant because it made it practical to solve the problem for large graphs and therefore more applicable to real-world problems.

It was independently rediscovered by numerous other researchers in later years, most notably by Georges Sollin in 1965, which has led to the algorithm also being known as Sollin's algorithm in parallel computing literature [3].

The algorithm uses a divide-and-conquer approach that is based on the idea of building a forest of trees, and is a hybrid of Prim's and Kruskal's algorithms to find a minimum spanning tree. As aforementioned, an advantage of Borůvka's algorithm over Prim's and Kruskal's is that it does not have to pre-sort the edges, nor does it have to maintain a priority queue. In each iteration, it finds the cheapest edge that connects two different trees and combines the trees into a single tree. Borůvka's algorithm continues to iterate until there is only one tree left – the minimum spanning tree.

Figure 1: Borůvka's Short Paper [1]



ZVLÁŠTNÍ OTISK Z ČASOPISU „ELEKTROTECHNICKÝ OBZOR"

Roč. 15. Čís. 10. Praha III., Cihelná 102. 5. března 1926.

Dr. OTAKAR BORŮVKA:

Příspěvek k řešení otázky ekonomické stavby elektrovodných sítí.

Ve své práci „O jistém problému minimálním" *) odvodil jsem obecnou větu, jíž jest ve zvláštním případě řešena tato úloha:

V rovině (v prostoru) jest dáno n bodů, jejichž vzájemné vzdálenosti jsou vesměs různé. Jest je spojiti sítí tak, aby:

1. každé dva body byly spojeny buď přímo anebo prostřednictvím jiných,
2. celková délka sítě byla co nejmenší.

Obr. 1.

Obr. 3.

Obr. 2.

Obr. 4.

Jest zřejmé, že řešení této úlohy může míti v elektrotechnické praksi při návrzích plánů elektrovodných sítí jistou důležitost; z toho důvodu je zde stručně na příkladě vyložím. Čtenáře, jenž by se o věc blíže zajímal, odkazuji na citované pojednání.

Řešení úlohy provedu v případě 40 bodů daných v obr. 1.

Každý z daných bodů spojím s bodem nejbližším. Tedy na př. bod 1 s bodem 2, bod 2 s bodem 3, bod 3 s bodem 4 (bod 4 s bodem 3), bod 5 s bodem 2, bod 6 s bodem 5, bod 7 s bodem 6, bod 8 s bodem 9, (bod 9 s bodem 8) atd. Obdržím řadu polygonálních tahů 1, 2,.... 13 (obr. 2).

Každý z nich spojím nejkratším způsobem s tahem nejbližším. Tedy na př. 1 s tahem 2, (tah 2 s tahem 1), tah 3 s tahem 4, (tah 4 s tahem 3) atd. Obdržím řadu polygonálních tahů 1, 2,.... 4 (obr. 3).

Každý z nich spojím nejkratším způsobem s tahem nejbližším. Tedy tah 1 s tahem 3, tah 2 s tahem 3 (tah 3 s tahem 1), tah 4 s tahem 1. Obdržím konečně jediný polygonální tah (obr. 4), jenž řeší danou úlohu.

Obrazec převrácen.

*) Vyjde v nejbližší době v Pracích Moravské přírodovědecké společnosti.

Matematický ústav Masarykovy university v Brně, v lednu 1926.

2 Pseudocode

Algorithm 1: Borůvka's Algorithm

Input : A connected, edge-weighted, undirected graph $G = (V, E)$
Output: T (a minimum spanning tree of G) and *totalCost* (the total cost of T)

Initialise a minimum spanning tree T = (V, E'), where E' = {}.
Initialise the total cost of the minimum spanning tree, $totalCost = 0$.
Initialise a list of components N, where N_k denotes the vertices in component k.
for *each vertex v in V* **do**
    $N_v = v$.
while $|N| > 1$ **do**
    Initialise an empty list of minimum connecting edges, $L = \{\}$.
    for *each component c in N* **do**
        Initialise the cheapest edge e to ∞.
        for *each edge i, j in c* **do**
            if *i, j contains an endpoint that isn't in c and i, j is cheaper than e* **then**
                Set e to i, j.
        if *e is not* ∞ **then**
            Add e to L.
    for *each edge e in L* **do**
        if *e connects two different components* **then**
            Merge the components N_i and N_j into a single component, N_k, such that $N_k = N_i \cup N_j$.
            Add e to E' in T.
            Add the edge's weight to the total cost: $totalCost = totalCost + e.weight$
return T and *totalCost*.

2.1 Flowchart

3 Time and Space Complexity Analysis

3.1 Time Complexity

Borůvka's algorithm has a time complexity of O(E log(V)), where E is the number of edges and V is the number of vertices in the graph.

The outer loop runs for log(V) iterations, as each iteration of the algorithm reduces the number of trees to at most half of the previous number of trees. Within each iteration, the algorithm performs a single pass over all the edges to find the minimum weight edge that connects two different components. This is performed in O(E) time, as the algorithm must consider each edge once. The algorithm then performs a single pass over the minimum weight edges to merge two components into one where possible, which is also performed in O(E) time. This gives a total time complexity of O(E log(V)).

It should be noted that the best, average, and worst-case time complexities of Borůvka's algorithm are all the same because the algorithm is not dependent on the input data.

3.2 Space Complexity

The space complexity of Borůvka's algorithm is O(V), as it only stores the component information for each vertex, which contains each vertex's parent and minimum weight edge.

4 Limitations and Constraints

A limitation of Borůvka's algorithm is that it may not always find the true minimum spanning tree of a graph when the cheapest edge that connects two different components is not part of the true minimum spanning tree. This is because Borůvka's algorithm only considers the cheapest edge that connects two different components, and does not consider the cheapest edge that connects two components that are already connected. Other algorithms such as Kruskal's algorithm and Prim's algorithm do not have this limitation, as they consider all edges in the graph.

Furthermore, Borůvka's algorithm can be slow to find the minimum spanning tree when the graph has a large number of components or when the components in the graph are large in size. In these cases, the algorithm will have to perform more iterations and contract more edges.

Compared to Prim's algorithm, which runs in O(E + V log(V)) amortised time when using a Fibonacci heap [4], Borůvka's algorithm is slightly slower. However, this variation of Prim's algorithm with the Fibonacci heap is more complicated to implement.

Borůvka's algorithm only works on undirected graphs as it does not account for the direction of the edges, thus it cannot solve the directed minimum spanning tree problem, which is more complex than the minimum spanning tree problem. Other algorithms such as the Chu-Liu/Edmonds' algorithm (originally proposed in 1965) can solve the directed minimum spanning tree problem [5].

5 Applications

5.1 Designing Networks in the Real-World

As Borůvka's algorithm finds a minimum spanning tree, it is most directly used in the design of networks, such as electrical networks, communication networks, and transportation networks [6]. In the original application of an electricity network for Moravia, the vertices represented towns, and edges represented the distances between towns. Borůvka used the assumption that it was not necessary to directly connect every town to the source of electricity – it was sufficient for a town to connect via another town that was already connected to power [1].

5.2 Two-Approximation for the Travelling Salesperson Problem

Borůvka's algorithm satisfies the triangle inequality and finds a minimum spanning tree, so it also acts as a two-approximation algorithm for the travelling salesperson problem [7], which is NP-hard and thus not possible to solve in polynomial time [8]. Because it produces a two-approximation, the output is at most twice the cost of the optimal solution. This can be proven as the total cost of a full walk is at most twice the cost of the minimum spanning tree, and the algorithm returns a path with a cost less than the full walk, as our pre-order walk replaces two or more edges of the full walk with a single edge [7].

Algorithm 2: Two-Approximation for the Travelling Salesperson Problem with MST-DFS [7]

1 Set a vertex as the start.
2 Construct a minimum spanning tree, T.
3 Create a list of vertices, H, that is ordered according to when they are visited in a pre-order tree walk of T, and add the start vertex at the end.
4 Return the path H.

5.3 Parallel Computation of Minimum Spanning Trees

There are several other algorithms that are more optimal for finding a minimum spanning tree depending on the input graph – Prim's algorithm is faster for dense graphs, and Kruskal's algorithm is faster for sparse graphs [9]. However, this only considers sequential implementations of the algorithms

– Borůvka's algorithm has become increasingly popular because it is easy to parallelise and is therefore well-suited to distributed computing [10]. This contrasts with the aforementioned algorithms, which start with a single component and seek to add edges to it, making it difficult to parallelise them as we must keep and check edges in a strict order. As Borůvka's algorithm starts with multiple components and seeks to connect them with the shortest edge, it can be easily parallelised by distributing the edges between processors to determine the shortest connecting edge for each vertex [11].

5.4 Faster Sequential Algorithms for Minimum Spanning Trees

The concepts behind Borůvka's algorithm have also been used to develop faster sequential algorithms. For example, the expected linear time minimum spanning tree algorithm proposed by Karger, Klein, and Tarjan runs in O(E) time. It involves an adaptation of Borůvka's algorithm, known as the Borůvka step [12, 13], alongside a step to remove F-heavy edges in linear time [14].

References

[1] J. Nešetřil, E. Milková, and H. Nešetřilová, Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history *Discrete mathematics*, vol. 233, no. 1-3, pp. 3–36, 2001.

[2] A. Gupta and A. Bansal, "15-859e: Advanced Algorithms, Lecture #1: Deterministic MSTs." https://www.cs.cmu.edu/~anupamg/advalgos15/lectures/lecture01.pdf, 2015. Date Accessed: 27 December 2022.

[3] M. Sollin, Le trace de canalisation *Programming, Games, and Transportation Networks*, 1965.

[4] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[5] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs *Combinatorica*, vol. 6, no. 2, pp. 109–122, 1986.

[6] R. L. Graham and P. Hell, On the history of the minimum spanning tree problem *Annals of the History of Computing*, vol. 7, no. 1, pp. 43–57, 1985.

[7] T. Andreae and H.-J. Bandelt, Performance guarantees for approximation algorithms depending on parametrized triangle inequalities *SIAM Journal on Discrete Mathematics*, vol. 8, no. 1, pp. 1–16, 1995.

[8] M. Jünger, G. Reinelt, and G. Rinaldi, The traveling salesman problem *Handbooks in operations research and management science*, vol. 7, pp. 225–330, 1995.

[9] C. F. Bazlamaçcı and K. S. Hindi, Minimum-weight spanning tree algorithms a survey and empirical study *Computers & Operations Research*, vol. 28, no. 8, pp. 767–785, 2001.

[10] A. Mariano, A. Proenca, and C. D. S. Sousa, A generic and highly efficient parallel variant of boruvka's algorithm in *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 610–617, IEEE, 2015.

[11] S. Chung and A. Condon, Parallel implementation of Bouvka's minimum spanning tree algorithm in *Proceedings of International Conference on Parallel Processing*, pp. 302–308, IEEE, 1996.

[12] B. Dixon, M. Rauch, and R. E. Tarjan, Verification and sensitivity analysis of minimum spanning trees in linear time *SIAM Journal on Computing*, vol. 21, no. 6, pp. 1184–1192, 1992.

[13] V. King, A simpler minimum spanning tree verification algorithm in *Workshop on Algorithms and Data Structures*, pp. 440–448, Springer, 1995.

[14] D. R. Karger, P. N. Klein, and R. E. Tarjan, A randomized linear-time algorithm to find minimum spanning trees *Journal of the ACM (JACM)*, vol. 42, no. 2, pp. 321–328, 1995.