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 report 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. A minimum spanning tree is a subset of the edges that connects all the vertices together without any cycles and with the minimum possible total edge weight [1]. The algorithm originated from Otakar Borůvka in 1926 as a method of constructing an efficient electricity network for Moravia, a region of the Czech Republic [2]. Later, it was independently rediscovered by numerous other researchers, 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 is based on the idea of building a forest of trees, and uses a hybrid of Prim's and Kruskal's algorithms to find a minimum spanning tree. An advantage of Borůvka's algorithm over Prim's and Kruskal's algorithms 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 edges that connects different trees and uses them to combine trees together. Borůvka's algorithm continues to iterate until there is only one tree left – the minimum spanning tree.

When Borůvka's algorithm was originally discovered, 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 that were later discovered. Borůvka's algorithm was the first to solve the minimum spanning tree problem in polynomial time [4], which was significant because it made it practical to solve the problem for large graphs and therefore more applicable to real world problems.
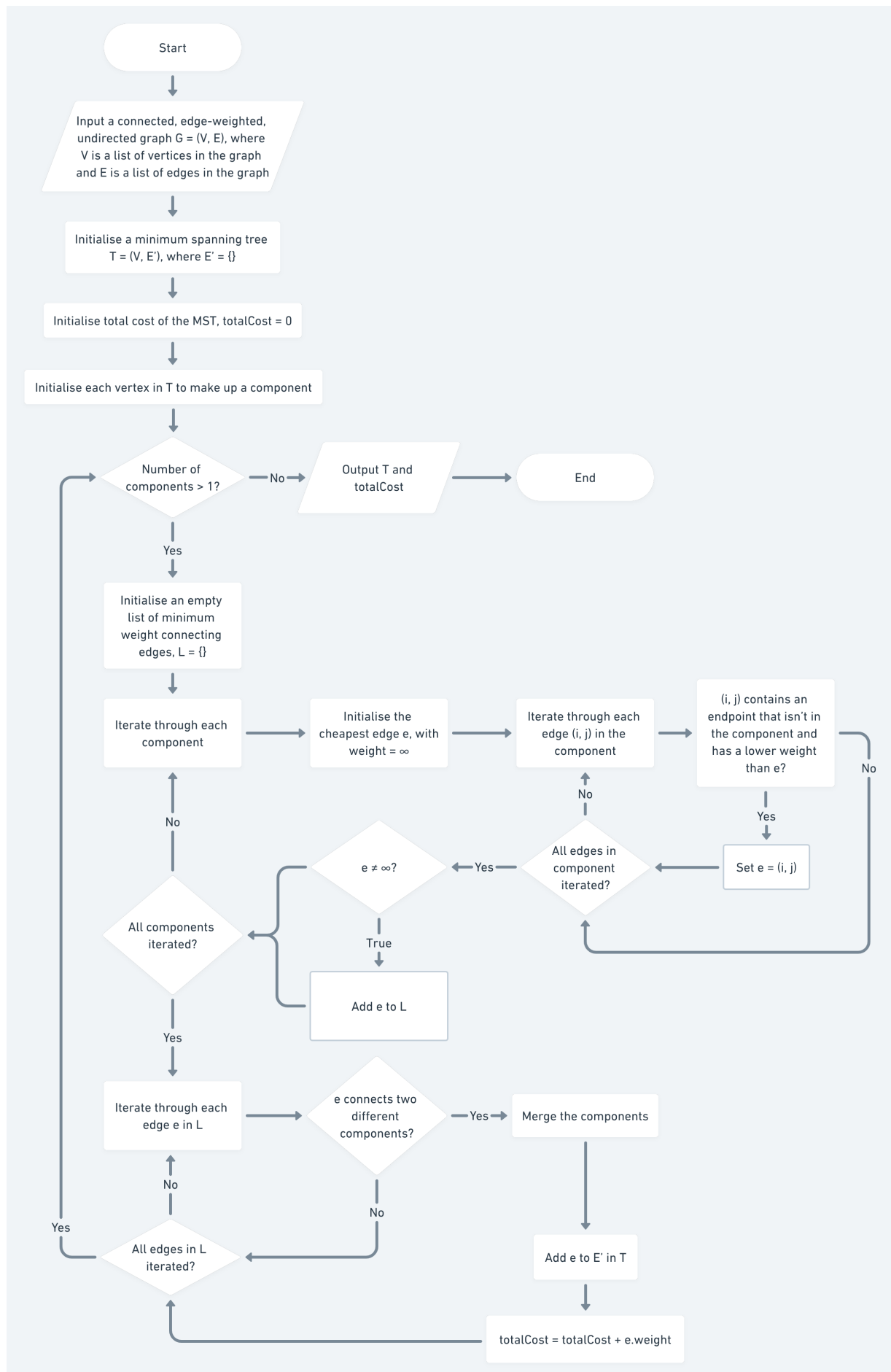
Figure 1: Borůvka's Short Paper [2]
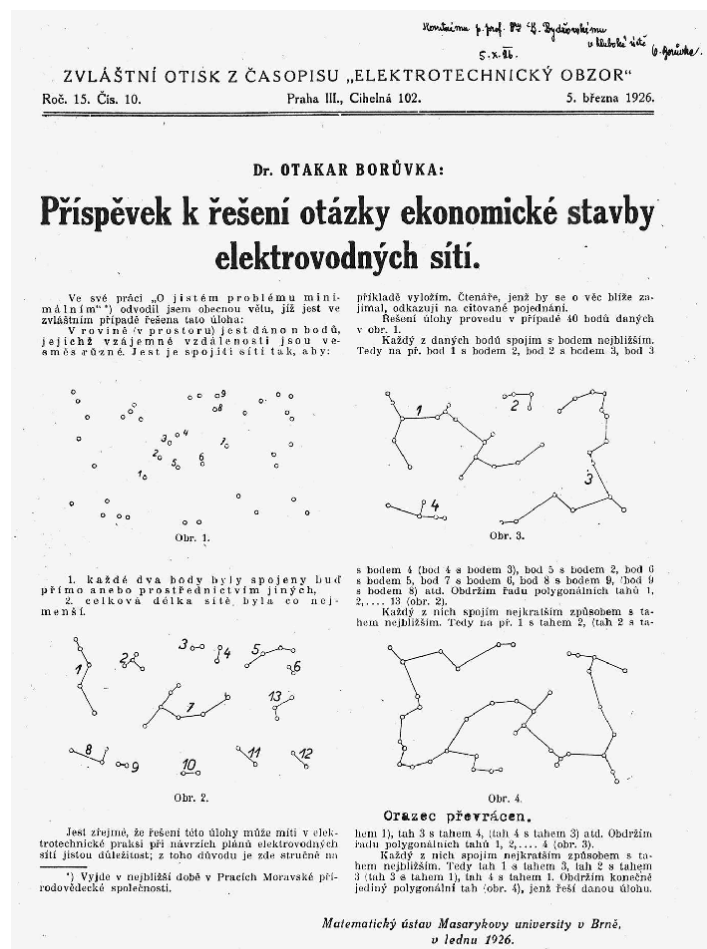
2   Pseudocode

Algorithm 1: Borůvka's Algorithm

Input   : A connected, edge-weighted, undirected graph $G = (V, E)$, where V is a list of vertices in the graph, and E is a list of edges in the graph

Output: T (a minimum spanning tree of G) and *totalCost* (the total cost of T)

1 Initialise a minimum spanning tree $T = (V, E')$, where $E' = \{\}$.
2 Initialise the total cost of the minimum spanning tree, $totalCost = 0$.
3 Initialise a list of components N, where N_k denotes the vertices in component k.
4 **for** *vertex $v \in V$* **do**
5 $\quad N_v = v$.

6 **while** $|N| > 1$ **do**
7 $\quad$ Initialise an empty list of minimum weight connecting edges, $L = \{\}$.
8 $\quad$ **for** *component $c \in N$* **do**
9 $\quad\quad$ Initialise the cheapest edge e, with weight $= \infty$.
10 $\quad\quad$ **for** *edge $i, j \in c$* **do**
11 $\quad\quad\quad$ **if** *i, j contains an endpoint that isn't in c and weight of $i, j < e$* **then**
12 $\quad\quad\quad\quad$ Set e to i, j.

13 $\quad\quad$ **if** *e is not ∞* **then**
14 $\quad\quad\quad$ Add e to L.

15 $\quad$ **for** *edge $e \in L$* **do**
16 $\quad\quad$ **if** *e connects two different components* **then**
17 $\quad\quad\quad$ Merge the components N_i and N_j into a single component, N_k, such that $N_k = N_i \cup N_j$.
18 $\quad\quad\quad$ Add e to E' in T.
19 $\quad\quad\quad$ Add the edge's weight to the total cost: $totalCost = totalCost + e.weight$

20 **return** T and *totalCost*.

2.1 Flowchart

Start

Input a connected, edge-weighted, undirected graph G = (V, E), where V is a list of vertices in the graph and E is a list of edges in the graph

Initialise a minimum spanning tree T = (V, E'), where E' = {}

Initialise total cost of the MST, totalCost = 0

Initialise each vertex in T to make up a component

Number of components > 1? —No→ Output T and totalCost → End

Yes

Initialise an empty list of minimum weight connecting edges, L = {}

Iterate through each component → Initialise the cheapest edge e, with weight = ∞ → Iterate through each edge (i, j) in the component → (i, j) contains an endpoint that isn't in the component and has a lower weight than e? —No

No

All edges in component iterated? ←Yes— Set e = (i, j)

Yes → Set e = (i, j)

e ≠ ∞? ←Yes— All edges in component iterated?

No

All components iterated?

True

Add e to L

Yes

Iterate through each edge e in L → e connects two different components? —Yes→ Merge the components

No

No

All edges in L iterated?

Yes

Add e to E' in T

totalCost = totalCost + e.weight

3 Time and Space Complexity Analysis

3.1 Time Complexity

Figure 2: Summary of Borůvka's Algorithm [5]

BORŮVKA: Add **_ALL_** the safe edges[2] and recurse.

Borůvka's algorithm run on the example graph. Thick edges are in F.
Arrows point along each component's safe edge. Dashed (gray) edges are useless.
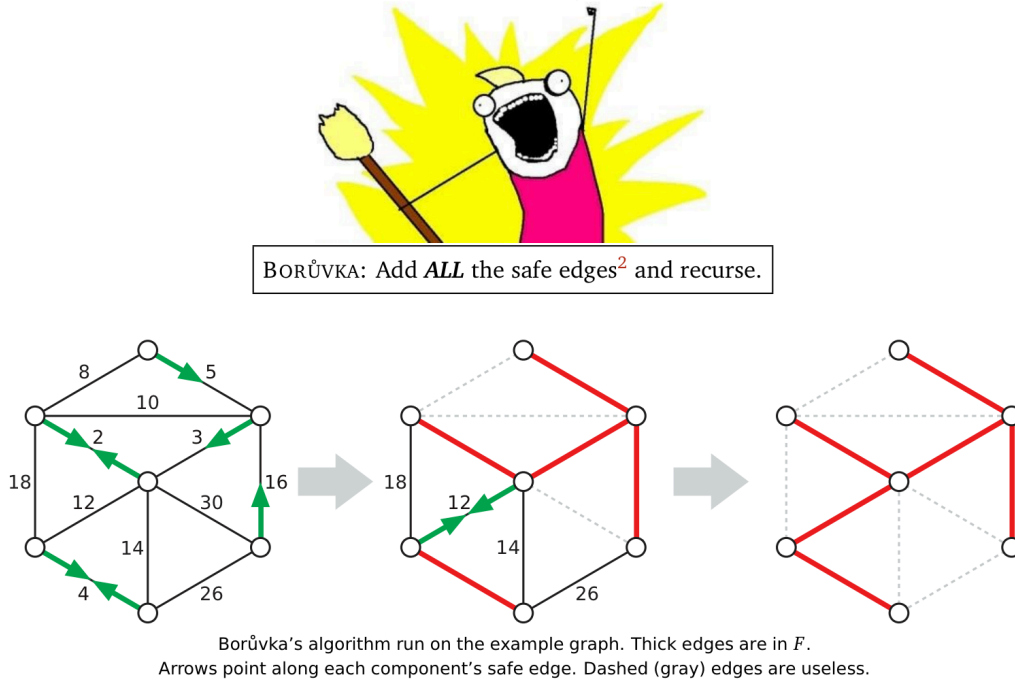
Borůvka's algorithm has a time complexity of O(E log(V)), where E is the number of edges and V is the number of vertices in the graph.

The outer loop runs for log(V) iterations, as each iteration reduces the number of components by at least a factor of two by combining them – the worst case occurs when the components coalesce in pairs [5]. Within each iteration, the algorithm initialises an empty list of minimum weight edges that connect two components together. For each component, it iterates over the edges and updates the minimum weight edge that connects to another component – this is executed in O(E) time, as the algorithm must consider each edge once. The algorithm then performs a single pass over the minimum weight connecting edges to merge two components into one where possible, which is also performed in O(E) time because each edge can be a minimum connecting edge. This gives a total time complexity of O(E log(V)).

However, Borůvka's algorithm often runs faster than the worst-case runtime of O(E log(V)), as this assumes that the number of components only reduces by a factor of two in each iteration [5]. For many graphs, the number of components reduces by a factor of more than two in each iteration, which improves the runtime of the algorithm significantly. The algorithm runs in O(E) time for a broad class of graphs such as planar graphs [6], which are graphs that can be drawn on a plane without any edges crossing each other. It does so by removing all but the cheapest edge between each pair of components [7]. In contrast, the time analysis for Prim's and Kruskal's algorithms apply to all graphs – this means that Borůvka's algorithm is more efficient for planar graphs amongst others.

3.2 Space Complexity

The space complexity of Borůvka's algorithm is O(E + V), as we must keep a list of components that is equal to the number of vertices at the start, and a list of minimum weight connecting edges.

4 Limitations and Constraints

4.1 Worst-Case Time Complexity

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. 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. In the worst case of Borůvka's algorithm, components will coalesce in pairs, and the number of components only reduces by a factor of two – this means the algorithm must perform more iterations, up to O(log V).

Compared to Prim's algorithm, which runs in $O(E + V \log(V))$ amortised time when using a Fibonacci heap [8], Borůvka's algorithm is slightly slower in worst-case scenarios. However, this variation of Prim's algorithm with the Fibonacci heap is more complicated to implement, and there are many types of graphs that Borůvka is more efficient at solving, such as planar graphs. In the next section we will also explore parallelisation with Borůvka's algorithm, which is not possible with Prim's. Based on these factors, Borůvka's algorithm is generally more efficient than Prim's algorithm in practice.

4.2 Directed Minimum Spanning Tree Problem

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 [9].

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 [1]. 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 [2].

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 [10], which is NP-hard and thus not possible to solve in polynomial time [11]. 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 [10].

Algorithm 2: Two-Approximation for the Travelling Salesperson Problem with MST-DFS [10]

---

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 [12]. However, this only considers sequential implementations of the algorithms – Borůvka's algorithm has become increasingly popular because it is easy to parallelise [13]. This contrasts with the aforementioned algorithms, which are intrinsically serial – they 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 parallelised by distributing the edges between processors to determine the shortest connecting edge for each vertex [14]. The parallel implementation of Borůvka's algorithm enables faster performance on multi-core or distributed systems, giving it an advantage over other classicial minimum spanning tree problems when working at a large scale.

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 [15, 16], alongside a step to remove F-heavy edges in linear time [17].

References

[1] 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.

[2] 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.

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

[4] 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.

[5] J. Erickson, "Lecture 25: Minimum Spanning Trees [Fa'14]." https://courses.engr.illinois.edu/cs498374/fa2014/notes/25-mst.pdf, 2014. Date Accessed: 28 December 2022.

[6] D. Cheriton and R. E. Tarjan, Finding minimum spanning trees *SIAM journal on computing*, vol. 5, no. 4, pp. 724–742, 1976.

[7] M. Mareš, Two linear time algorithms for MST on minor closed graph classes tech. rep., ETH Zurich, 2002.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.