Borůvka’s Algorithm Analysis

Student Number: 700040999

Word Count: 1,500 (including titles), 1,485 (not including titles)

Abstract: In this report, Borůvka’s algorithm is discussed, including an explanation of the logic behind the algorithm, as well as an analysis against Prim’s Algorithm, Kruskal’s Algorithm, and the Reverse-Delete Algorithm. Numerous applications for minimum spanning trees are identified and explained as well. The complexity of Borůvka’s algorithm, Prim’s Algorithm, Kruskal’s Algorithm, and the Reverse-Delete Algorithm are also compared.

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

Signed: Jenni Schofield

Algorithm Principles

Borůvka's algorithm was designed by Otakar Borůvka in 1926 to help create an efficient energy network for Moravia, where he lived. It is the first example of the minimum spanning tree problem which, given a set of nodes with connections, the algorithm tries to find the tree connecting all the nodes that has the least cost. After Borůvka's discovery, the algorithm was rediscovered multiple times [1], [2], namely in 1965 by Georges Sollin [3]. This led to Borůvka's algorithm occasionally being called Sollin's algorithm, mainly in the literature about parallel computing. The greedy algorithm works by initialising all nodes in a network to individual subtrees and getting each node's minimum cost path. Once all the minimum cost paths have been found, the nodes are connected accordingly. The connected nodes are then grouped into new subtrees, and the process repeats until there is only one subtree, the minimum spanning path. The minimum spanning tree problem is common, and Borůvka's algorithm is one of four commonly used classic algorithms. The key aspect of Borůvka's algorithm is the concept of a Borůvka step. A Borůvka step is the action of choosing the best edge for each vertex, then combining them into subtrees, and trimming out loops, non-minimums, and unconnected vertices. This step was used in Karger *et al.*'s paper, where they combined Borůvka's algorithm with random sampling to improve performance. The main principles of the algorithm are how well it can be parallelised, where each vertex's shortest path analysis is run in parallel, and how rather than growing out from one node, like Prim's algorithm and Kruskal's algorithm, it examines the whole graph and handles the tree formation for each node at the same time.

Pseudocode

Algorithm 1 Borůvka's Algorithm

```
  Input: G, an undirected, weighted Graph
  Output: T, the minimum spanning tree
procedure BORŮVKA'S ALGORITHM
    Initialise T to be a list of subtrees,
    each containing one vertex in the graph
    while the size of T is greater than 1 do
        for each subtree (C) in T do
            E, an empty list of edges
            for every vertex (V) in C do
                Find the shortest path from V to a vertex that is not V
                Add that path to E
            end for
        end for
        Combine the vertices that are connected in E to form a larger
        subtree, and update T to include the new subtrees
```

Space and Time Complexity

The time complexity of Borůvka's algorithm is *O(E log V)*. This is derived from the outer loop (the while loop) running at *O(log V)* since every iteration of that loop halves the size of T, multiplied by the inner loop's complexity, which evaluates the same number of times as there are edges, *O(E)*. Thus, by combining the *O(log V)* of the while loop with the *O(E)* of the innermost loop, the total complexity is *O(E log V)*. The space complexity of Borůvka's algorithm is *O(E + V)*, as every vertex and all the edges have to be stored in memory. For comparison, Prim's algorithm's space complexity is also *O(E log V)*, unless it is optimised to use a Fibonacci heap, then it improves to *O(E + V log V)*. Kruskal's algorithm's space complexity is also *O(E* *log V)*. Since both Prim's and Kruskal's algorithms are required to store the vertices and edges in memory, their space complexities are both *O(E + V)*, the same as Borůvka's. There is also the reverse-delete algorithm, which is a reversal of Kruskal's algorithm, but unlike Kruskal's algorithm, runs with a time complexity of $O(E \log V(\log \log V)^3)$. As the reverse-delete algorithm is the reversal of Kruskal's algorithm, it also has a space complexity of *O(E + V)*.

Limitations and Advances

Despite Prim's algorithm, Borůvka's algorithm, and Kruskal's algorithm all having the same time complexity, most minimum spanning tree solutions use Prim's algorithm over Kruskal's algorithm or Borůvka's algorithm. Prim's algorithm is more efficient than Borůvka's, as demonstrated in Marpaung and Arnita [4], and although Kruskal's algorithm's efficiency is on par with Prim's algorithm, Kruskal's is better fitted for sparse graphs, whereas Prim's is better for dense graphs. When using these algorithms in a real-world scenario, such as the electricity network example, the graph generated would likely be dense. Recent research has been examining the extension of Borůvka's algorithm, namely in Karger *et al.*'s paper, where the Borůvka step is combined with a random-sampling step to achieve a time complexity of *O(E)*. The current best minimum spanning tree algorithm, written by Bernard Chazelle, also uses the Borůvka step to reduce the number of vertices to be connected [5]. Chazelle's algorithm performs in $O(m\alpha(m,n))$, where α "represents the classical functional inverse of Ackermann's function, n represents the number of vertices, and m represents the number of edges." Another extension combines Prim's algorithm's logic and Borůvka's algorithm's inherent parallelism to create an algorithm that improves on other variations of Borůvka's parallel algorithm and Prim's algorithm alone [6]. Where Borůvka's algorithm lacks, such as efficiency, rather than trying to advance past it, it makes sense to choose a more optimal classic minimum spanning tree algorithm. The reason Borůvka's algorithm has not sunk into disuse is not only was it the first minimum spanning tree algorithm ever, but its two competitors both are sequential and as we move into the modern age, parallelism is being more thoroughly explored[7].

Application of this Algorithm

Minimum spanning trees are often used in resource management, such as telecommunication networks, transportation networks, and utility services. [8] An example is in ecotoxicological analysis, where different types of bacteria are examined for how toxic certain heavy metals are. A minimum spanning tree can be drawn to connect the different bacteria for each test, and then each minimum spanning tree for each heavy metal can then be compared to see the toxicity of the heavy metals compared to another [9]. Another practical example is being able to identify curves in images for computer vision. In Suk and Song's paper, they construct a minimum spanning tree out of edges generated by the SOBEL operator, which then moves to remove the "hairs", which are noise-causing branches of the minimum spanning tree [10]. One of the older examples of minimum spanning tree use is in the 1957 paper by Sneath to aid in the classification of bacteria. His method was to group bacteria with high similarity scores while minimising the difference between two groups of bacteria. In his implementation, every feature of the bacteria has the same weight, which allows future taxonomists to adjust certain features to be more or less important. Minimum spanning trees can also be used for clustering, and in Xu *et al.*'s paper, they apply the clustering implementation of a minimum spanning tree to gene expression data. They then take it one step further and suggest that to separate the minimum spanning tree into separate clusters, a certain number of the longest edges of the minimum spanning tree should be cut. In a less academic approach, a minimum spanning tree could also be used to evaluate the shortest distance to connect all cities by rail, or connect all the houses in a neighbourhood to the electricity grid. These examples are different from the travelling salesman problem, which asks for a path between all the cities, instead of just having the cities connected. Although Borůvka's algorithm can not be used just by itself for the travelling salesman problem, another algorithm uses it as a component. The Christofides-Serdyukov algorithm solves the travelling salesman problem by first generating a minimum spanning tree from the cities given, then getting all the vertices with an odd degree. After that, the minimum matching list is found from the list of odd-degree vertices. A matching list is a set of edges that do not have common vertices. After that, the matching set is combined with the minimum spanning tree, then an Eulerian circuit is formed. An Eulerian circuit is a path where every edge is visited exactly once. After this, the Eulerian circuit is converted into a Hamiltonian circuit by eliminating duplicates. A Hamiltonian circuit is where every vertex is visited exactly once, but may repeat edges. Converting from an Eulerian circuit to a Hamiltonian circuit guarantees that there won't be any repeat edges or repeat vertices in the resulting path. This algorithm is the best polynomial time approximation algorithm for general travelling salesman problems [11]. Using Borůvka's algorithm in the Christofides-Serdyukov algorithm allows for parallelisation in the initial step, and speeds up the algorithm where Prim's algorithm and Kruskal's algorithm would not.

Conclusion

Overall, Borůvka's algorithm founded the concept of a minimum spanning tree problem before computers were even invented. The applications of the minimum spanning tree problem stretch across a variety of fields, including taxonomy, biology, computer vision, and genetics, as well as city planning and travelling salesman problem optimising. The space complexity of Borůvka's algorithm matches Prim's algorithm, Kruskal's algorithm, and the reverse-delete algorithm. The time complexity of Borůvka's algorithm matches the binary heap implementation of Prim's algorithm, and Kruskal's algorithm and outperforms the reverse-delete algorithm. The Fibonacci heap implementation of Prim's algorithm outperforms Borůvka's algorithm, which along with how as graph theory developed, Prim's algorithm and Kruskal's algorithm surpassed Borůvka's algorithm's efficiency, despite having the same time complexity. However, Borůvka's algorithm remains relevant today due to the parallel nature of the algorithm, compared to Prim's algorithm and Kruskal's algorithm's sequential nature. Borůvka's algorithm by itself is rarely used in practical applications now, but the Borůvka step is an essential part of the latest solutions to the minimum spanning tree problem.

References

[1] G. Choquet, ‘Étude de certains réseaux de routes’, *Comptes Rendus Académie Sci.*, vol. 206, pp. 310–313, 1938.

[2] K. Florek, J. Łukaszewicz, J. Perkal, H. Steinhaus, and S. Zubrzycki, ‘Sur la liaison et la division des points d’un ensemble fini’, *Colloq. Math.*, vol. 2, no. 3–4, pp. 282–285, 1951.

[3] G. Sollin, ‘Le tracé de canalisation’, *Program. Games Transp. Netw.*, 1965.

[4] F. Marpaung and Arnita, ‘Comparative of Prim’s and Boruvka’s algorithm to solve minimum spanning tree problems’, *J. Phys. Conf. Ser.*, vol. 1462, no. 1, p. 012043, Feb. 2020, doi: 10.1088/1742-6596/1462/1/012043.

[5] B. Chazelle, ‘A minimum spanning tree algorithm with inverse-Ackermann type complexity’, *J. ACM*, vol. 47, no. 6, pp. 1028–1047, Nov. 2000, doi: 10.1145/355541.355562.

[6] D. A. Bader and G. Cong, ‘Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs’, *J. Parallel Distrib. Comput.*, vol. 66, no. 11, pp. 1366–1378, Nov. 2006, doi: 10.1016/j.jpdc.2006.06.001.

[7] A. Mariano, A. Proenca, and C. Da Silva 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*, Turku, Finland: IEEE, Mar. 2015, pp. 610–617. doi: 10.1109/PDP.2015.72.

[8] R. L. Graham and P. Hell, ‘On the History of the Minimum Spanning Tree Problem’, *IEEE Ann. Hist. Comput.*, vol. 7, no. 1, pp. 43–57, 1985, doi: 10.1109/MAHC.1985.10011.

[9] J. Devillers and J. C. Dore, ‘Heuristic potency of the minimum spanning tree (MST) method in toxicology’, *Ecotoxicol. Environ. Saf.*, vol. 17, no. 2, pp. 227–235, Apr. 1989, doi: 10.1016/0147-6513(89)90042-0.

[10] M. Suk and O. Song, ‘Curvilinear feature extraction using minimum spanning trees’, *Comput. Vis. Graph. Image Process.*, vol. 26, no. 3, pp. 400–411, Jun. 1984, doi: 10.1016/0734-189X(84)90221-4.

[11] N. Christofides, ‘Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem’, *Oper. Res. Forum*, vol. 3, no. 1, p. 20, Mar. 2022, doi: 10.1007/s43069-021-00101-z.