

Borůvka's Algorithm

Algorithms that Changed the World

SID: 720007925

Abstract

In real world applications, abstraction is key when solving complex problems. Many of these problems can boil down to a graphical abstraction. Things, places, or ideas can be represented as nodes connected by edges modelling their relationship. Often these edges have a cost and, in the case of finding optimal solutions to problems, we may need a method for linking all nodes with a minimum cost. Borůvka's

Algorithm is the oldest Minimum Spanning Tree algorithm aiming to solve just this. This report encapsulates the key ideas of the algorithm, as well as commenting on its complexity, applications, and relevancy for Graph theory.

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

I have used GenAI for the Mandatory Appendix for this assessment

I declare that I have referenced use of GenAI outputs within my assessment in line with the University referencing guidelines.

J. Thomas

Word Count: 1384

1 Introduction

Dating back to 1926, Borůvka’s algorithm is the oldest and most influential recorded example of a minimum spanning tree algorithm [1]. This report discusses the key principles of the algorithm, its complexity, limitations, and applications.

Otakar Borůvka was tasked with finding the optimal method of connecting the Czech region of Moravia with an electrical network [2]. To do this, he devised an algorithm to join areas of the region with the least cost connection, by representing the region with a graph abstraction. The underlying algorithm is used to find the minimum spanning tree (MST) of a graph, defined as a set of vertices and edges. The minimum spanning tree is the least cost subgraph, connecting all vertices with no cycles. Borůvka’s algorithm was rediscovered in 1965 by Georges Sollin [3], hence it is sometimes referred to as Sollin’s algorithm. The algorithm, as with all MST algorithms, requires the graph being undirected, fully connected, and weighted.

2 Principles

Borůvka’s algorithm is outlined with pseudocode in Algorithm 1. To perform the algorithm, first the graph is split into a forest of connected components, containing just one vertex each (hence no edges). From here, the minimum edge connecting each component to any other is added to the trees of the forest. In this step, at worst, the new number of trees is $\frac{|V|}{2}$ since every vertex could pair with just one other. At best, there could be one tree remaining in the forest, which would only occur in certain special cases. If the number of connected components in the forest is now 1, the MST is found. If not, the algorithm continues to iterate connecting each component with the minimum weighted edge until there is one tree remaining. The process of finding shortest edges is done for all components before the edges are added to the forest. Since edges are added to connect two components, there is no concern of introducing a cycle to the graph, since there are no edges added within a component. A worked example of the algorithm is provided in Figures 1 to 4.

In a triangular graph, with each edge weighing 1, on the first iteration if (a, b) , (b, c) , & (c, a) are all included, a cycle is formed. Because of this, a tie-breaking rule must be used in the algorithm. This has to be applied consistently throughout the algorithm and typically chooses the lower numbered vertex as a priority. With consistent tie breaks the algorithm will not produce a cycle. This is because the algorithm relies on the cut-property of graphs [4], saying that if just two cuts (or components) are taken, the minimum link edge between the components is included in the MST for the overall graph. Because of this idea, the MST produced definitely has no cycles and is optimal.

Borůvka’s is a greedy algorithm, meaning it makes locally optimal decisions, without regard for the overall problem. This property is due to the algorithm choosing the least weighted connection between trees at each stage. There is no backtracking or reconsideration of decisions after they have been made. It uses a divide and conquer method, splitting the vertices up and solving from there. The algorithm makes deterministic decisions, with no random or stochastic steps. This means the algorithm will always produce the same output in the same steps from the same input. This helps predictability of output and its complexity. The only potential introduction of unpredictability would be from the tie-break rule, if not applied consistently. Because the algorithm traverses graphs edge by edge (unlike Prim’s or Kruskal’s traversing on vertices), the algorithm is particularly efficient on sparse graphs where $|V| \simeq |E|$. Since the algorithm does not rely on a breadth or depth first search, it can be parallelised for computation on multiple processors. This makes Borůvka’s algorithm particularly desirable for high performance applications.

3 Complexity

As can be seen by Algorithm 1, there are two loops, one nested inside the other. The inner for loop executes once for each tree in the forest, each of the steps inside the loop are linear and therefore bound

Algorithm 1 Borůvka's Algorithm

Require: $G = (V, E)$ such G is undirected, weighted, and connected, and $V = \{V_1, V_2, \dots, V_n\}$

Ensure: MST is minimum spanning tree of G

```
 $F \leftarrow \{(V_1, \emptyset), (V_2, \emptyset), \dots, (V_n, \emptyset)\}$  ▷ Forest initialised with each vertex as a component/tree  
 $MST \leftarrow (V, \emptyset)$  ▷ An empty MST  
while  $|F| > 1$  do ▷ Iterating while multiple trees in forest  
   $E' \leftarrow \emptyset$   
  for component  $T$  in  $F$  do  
    find shortest edge  $e$  connecting  $T$  to another component  $T'$  ▷ Resolve ties consistently  
     $E' \leftarrow E' \cup \{e\}$   
  end for  
  Add  $\forall e \in E'$  to  $MST$   
  Merge components in  $F$  using  $E'$  ▷ Using Union-Find structure for efficiency  
end while  
return  $MST$ 
```

by $O(|E|)$ since it searches the edges for the shortest. On each iteration of the outer loop, the number of trees in the forest, at a minimum, halves. This is because, in the worst case, each tree pairs up with just one other. Within the loop, $O(\log|V|)$ represents the halving nature of the number of trees and thus the search space. Due to the nesting of loops we have an overall complexity of $O(|E|\log|V|)$. The space complexity is of $O(|V|)$ since the algorithm stores the vertices V multiple times in other structures, but only stores the edges E in one and hence the largest definer of growth is the vertices.

Some optimisations can be made to the basic implementation shown in Algorithm 1 to improve the complexity. Every time two trees join, any other internal edges of the component can be removed from the search list since they definitely cannot be added. This will decrease the search space for the minimum edge link with each iteration. More complex data structures can also be used to increase efficiency. The Union-Find structure [5] facilitates efficient merging of components in graphs. When combined with path compression and union by rank, the complexity of any merge or find operations is reduced to $O(\alpha(n))$ where α is the inverse Ackermann function, which grows so slowly it can be compared to linear time.

4 Limitations

One main limitation of Borůvka's MST algorithm is shared with other MST algorithms in that they require a connected graph as input. Without connection in the graph, no MST exists. The simple implementation of Borůvka's algorithm in Algorithm 1 requires a precondition of an undirected, weighted, and connected graph. The algorithm assumes this condition is satisfied and continues. If the precondition is not satisfied the algorithm will not terminate, or will terminate incorrectly. The pseudocode can be improved on by including explicit checks for satisfaction of preconditions although this will come at a detriment to the efficiency. Other limitations of the basic implementation have been discussed under the Complexity title in Section 3, with potential solutions.

5 Applications

The original application and motivation for the development of the algorithm was the problem of optimising the Czech electricity network [2]. After Borůvka's friend, an employee of the electricity company, had suggested he helped design an efficient network, the algorithm was developed. This was the first real world application of the algorithm and led to the use of Borůvka's algorithm for other optimisation problems on transport networks, water systems, or circuit design [6].

Since each component picks the shortest link to another independently of the other components, due to its greedy nature, the algorithm is easily parallelisable [7]. A parallel version would see each component picking an edge simultaneously over multiple threads of execution. This means that Borůvka's is particularly favoured over Prim's and Kruskal's for performance critical applications like big data and distributed systems, despite them having the same base complexity.

Being the oldest MST algorithm, Borůvka's has had many amendments and iterations made to it over the decades. In 1995, Karger et al. proposed "A randomized linear-time algorithm to find minimum spanning trees" [8]. The algorithm is based on the idea of a "Borůvka step", being a step in the algorithm taken from the original Borůvka's algorithm. These steps are intertwined with random sampling steps. With the introduction of a stochastic nature, the algorithm has the same worst case as Borůvka's original of $O(|E|\log|V|)$, however, in an expected case, the adapted algorithm runs in $O(|E|)$, i.e. linear time. This vast improvement using the original shows not only how strong of a base Borůvka's was, but also how influential it has been on the MST field.

6 Conclusion

To summarise, Borůvka's algorithm is the oldest MST algorithm. It computes a minimum spanning tree by treating each vertex as a tree in a forest and then iteratively connecting each tree to its cheapest neighbour, before merging the trees that are now connected. Due to the merging nature, the search space at least halves on each iteration and therefore the algorithm has a complexity of $O(|E|\log|V|)$.

The influence of the algorithm is profound. Being the original MST algorithm, Borůvka's has led to the development of many other algorithms, including parallelised and stochastic variants computing MSTs extremely quickly. Borůvka's has had applications on practical problems since its inception; being initially developed to optimise Czech electricity networks.

Figures

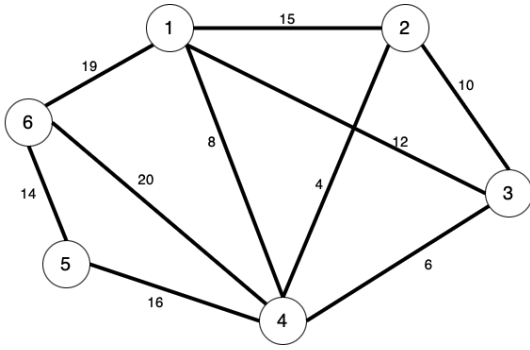


Figure 1: Initialisation of a Graph

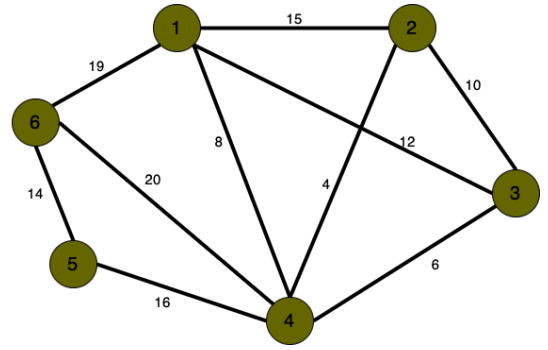


Figure 2: Step 1 of Borůvka's

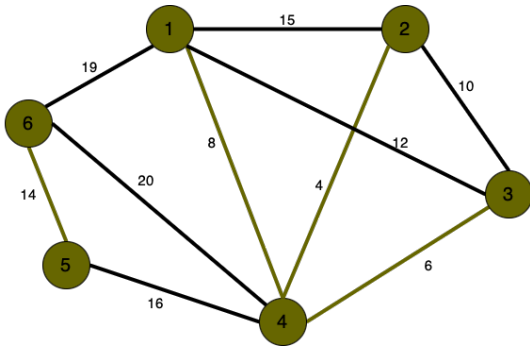


Figure 3: Step 2 of Borůvka's

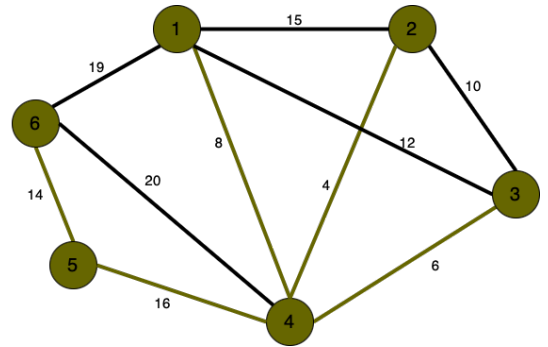


Figure 4: Step 3 of Borůvka's – Solution

References

- [1] O. Borůvka, “O jistém problému minimálním,” *Práce Mor. Přírodově d. spol. v Brně (Acta Societ. Scient. Natur. Moraviae)*, 1926.
- [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] G. Sollin, “Le trace de canalisation,” *Programming, Games, and Transportation Networks*, 1965.
- [4] S. Pettie and V. Ramachandran, “An optimal minimum spanning tree algorithm,” *Journal of the ACM (JACM)*, vol. 49, no. 1, pp. 16–34, 2002.
- [5] J. Gustedt, “Efficient union-find for planar graphs and other sparse graph classes,” *Theoretical Computer Science*, vol. 203, no. 1, pp. 123–141, 1998.
- [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] S. Chung and A. Condon, “Parallel implementation of borůvka's minimum spanning tree algorithm,” in *Proceedings of International Conference on Parallel Processing*, pp. 302–308, IEEE, 1996.
- [8] 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.

AI Appendix

Me: You are a computer science final year student. Write the main principles about Borůvka's algorithm

ChatGPT: Borůvka's algorithm is a greedy algorithm used to find the Minimum Spanning Tree (MST) of a connected, undirected graph. It was one of the earliest algorithms for this purpose and is particularly useful for parallel processing. Below are the main principles and steps of Borůvka's algorithm:

Main Principles of Borůvka's Algorithm:

1. Divide and Conquer:

- The algorithm repeatedly divides the graph into smaller connected components (initially, each vertex is its own component).
- Each iteration merges components by adding the smallest edge connecting a component to any other.

2. Greedy Strategy:

- At each step, the smallest weight edge is chosen for inclusion in the MST.
- This ensures that the MST grows incrementally with minimal weight at each step.

3. Parallelism-Friendly:

- Each vertex or component independently selects the smallest outgoing edge, making it suitable for distributed or parallel processing.

4. Iterative Merging:

- As components are merged, the algorithm reduces the graph's complexity until all vertices belong to a single connected component, forming the MST.

Steps in Borůvka's Algorithm:

1. Initialize Components:

- Treat each vertex as an individual component.

2. Find the Smallest Edge for Each Component:

- For every component, find the smallest weight edge connecting it to any other component.

3. Merge Components:

- Add all selected smallest edges to the MST.
- Merge components connected by these edges into a single component.

4. Repeat Until Convergence:

- Continue finding and merging components until only one connected component remains, which represents the MST.

5. Terminate:

- The algorithm terminates when all vertices belong to a single connected component.

Key Characteristics:

- Time Complexity: $O(E \log V)$, where E is the number of edges and V is the number of vertices. With more efficient data structures like a union-find, the time complexity can improve.

- Space Complexity: $O(E + V)$, since the graph's edges and vertices need to be stored.

Borůvka's algorithm is conceptually simpler than Prim's or Kruskal's algorithms and can be particularly efficient for graphs with many edges or in parallel/distributed environments.

Me: Write the pseudo code to describe Borůvka's algorithm

ChatGPT: *Pseudo-code for Borůvka's Algorithm*

Input: A connected, undirected graph $G = (V, E)$ with vertices V and edges E

Output: A Minimum Spanning Tree (MST)

1. Initialize MST as an empty set.
2. Create a component array, 'components', where each vertex is its own component.
3. While there is more than one component:
 - (a) Initialize 'smallest_edge' array to store the smallest outgoing edge for each component.
 - (b) For each edge (u, v) in E :
 - i. Determine the components of u and v .
 - ii. If u and v belong to different components:
 - Check if this edge is the smallest for either component.
 - Update the 'smallest_edge' for the components if necessary.
 - (c) Add all selected edges in 'smallest_edge' to the MST.
 - (d) Merge the components connected by the selected edges:
 - i. Use a union-find data structure to update component information.
 - (e) Remove duplicate edges and update the graph representation.
4. Return the MST.

Detailed Explanation of the Pseudo-code:

1. Initialization:
 - Start with an empty MST and treat each vertex as a separate component.
2. Find Smallest Edges:
 - For every edge in the graph, determine which components the edge connects.
 - If the edge connects two different components, consider it as a candidate for merging.
3. Merge Components:
 - Add the selected edges (the smallest edges for each component) to the MST.
 - Update the component information to reflect the merging of components.
4. Repeat Until One Component Remains:
 - The process continues until all vertices belong to a single connected component, representing the MST.
5. Return the MST:
 - The MST is the set of edges collected during the process.

This pseudo-code outlines the high-level structure of Borůvka's algorithm, which can be implemented efficiently using a union-find data structure to manage components and their merging.

Me: Write the complexity analysis of the algorithm

ChatGPT: *Complexity Analysis of Borůvka's Algorithm:*

Borůvka's algorithm is efficient for finding the Minimum Spanning Tree (MST), especially in dense graphs and parallel processing environments. Here's a breakdown of the time and space complexity:

Time Complexity

1. Initialization:

- Assigning each vertex to its own component and initializing data structures takes $O(V)$, where V is the number of vertices.

2. Finding the Smallest Edge:

- For each component in every iteration, the algorithm examines all edges in the graph to find the smallest outgoing edge for each component.
- Scanning all edges takes $O(E)$, where E is the number of edges.

3. Component Merging:

- Merging components is performed using a *union-find data structure* with path compression and union by rank.
- The amortized cost of a union or find operation is $O(\alpha(V))$, where α is the inverse Ackermann function (very small, nearly constant for practical values of V).

4. Number of Iterations:

- In each iteration, the number of components is reduced significantly (at least halved), since every component adds at least one edge to the MST.
- This guarantees the number of iterations is at most $\log V$.

Overall Time Complexity: $O(E \log V)$

Space Complexity

1. Graph Representation

- The graph requires $O(E + V)$ space to store the edges and vertices.

2. Union-Find Data Structure

- The union-find data structure requires $O(V)$ space to store parent and rank arrays.

3. Temporary Data Structures

- Additional arrays, such as the smallest outgoing edge for each component, require $O(V)$ space.

Overall Space Complexity: $O(E + V)$

Summary of Complexity

- Time Complexity: $O(E \log V)$
- Space Complexity: $O(E + V)$

Why Borůvka's Algorithm is Efficient

- *Parallelism*: Finding the smallest outgoing edge for each component can be parallelized, making the algorithm suitable for distributed systems.
- *Graph Density*: For dense graphs $E \approx V^2$, the algorithm efficiently handles large numbers of edges.

- *Union-Find Optimization*: The use of efficient union-find operations ensures that merging components is fast, even in large graphs.

Me: Write the limitations or constraints of the algorithm and the methods or tools that could overcome them

ChatGPT: *Limitations or Constraints of Borůvka's Algorithm*

1. Dependency on Connected Graphs:

- Borůvka's algorithm assumes that the input graph is connected. If the graph is disconnected, it will only produce the MST for one connected component.
- *Solution*: Pre-process the graph to identify connected components (using BFS or DFS). Then run Borůvka's algorithm separately for each component.

2. High Edge Count:

- For dense graphs with $E \approx V^2$, iterating over all edges repeatedly can become computationally expensive.
- *Solution*: Use edge list reduction: Remove unnecessary edges during iterations by maintaining only the smallest outgoing edges for each component. Employ priority queues or sorted edge lists to speed up edge selection.

3. Space Complexity in Large Graphs:

- Storing all edges in memory for large, dense graphs can consume significant space.
- *Solution*: Use streaming algorithms that process edges in batches. Implement distributed graph processing frameworks (e.g., Apache Spark or GraphX) to manage memory more effectively.

4. Difficulty in Sparse Graphs:

- Borůvka's algorithm may perform poorly compared to Kruskal's or Prim's algorithm for very sparse graphs due to unnecessary iterations and redundant checks.
- *Solution*: For sparse graphs, Kruskal's algorithm (with sorted edge lists) or Prim's algorithm (using adjacency lists and a priority queue) may be faster.

5. Serial Implementation Bottlenecks:

- Borůvka's strength lies in its ability to parallelize the edge selection process. However, in a purely sequential implementation, the performance advantage is diminished.
- *Solution*: Implement Borůvka's algorithm in a **parallel or distributed framework** like CUDA, OpenMP, or Apache Spark to leverage its inherent parallelism.

6. Overhead from Union-Find Operations:

- Frequent union-find operations can introduce some overhead, especially if the data structure is not implemented efficiently.
- *Solution*: Use optimized union-find with path compression and union by rank to reduce the amortized cost of operations.

7. Handling Negative Weights:

- Borůvka's algorithm can handle negative edge weights, but special care is needed to ensure correctness when weights are unusual (e.g., highly negative).
- *Solution*: No special tool is needed, as Borůvka naturally handles negative weights, but thorough testing should be done for edge cases.

Tools and Methods to Overcome Limitations:

1. Union-Find Optimization

- Use libraries or frameworks with efficient implementations of union-find (e.g., Boost Graph Library in C++).

2. Graph Libraries

- Libraries like NetworkX (Python) or LEDA (C++) provide efficient graph algorithms and can be used to implement or adapt Borůvka's algorithm.

3. Parallel Computing Frameworks

- Use OpenMP or CUDA to exploit the algorithm's parallelism.
- Distributed systems like Hadoop or Apache Spark can process extremely large graphs efficiently.

4. Graph Preprocessing

- Perform initial preprocessing to reduce the graph size, such as removing non-relevant edges or compressing the graph.

5. Hybrid Algorithms

- Combine Borůvka's algorithm with Kruskal's or Prim's algorithm for different phases of the graph traversal to optimize performance based on graph density.

By addressing these limitations with the above tools and methods, Borůvka's algorithm can be adapted to efficiently handle a wide range of graph scenarios.

Me: Write the applications of this algorithm in the real world systems for problem solving

ChatGPT: *Real-World Applications of Borůvka's Algorithm*

Borůvka's algorithm is widely used in scenarios that require efficient computation of Minimum Spanning Trees (MSTs), especially in large-scale or parallel processing systems. Below are some of its key applications in real-world systems for problem-solving:

1. Network Design

- Telecommunication Networks:
 - Used to design cost-effective communication networks (e.g., laying cables for telephony or internet) by minimizing the total connection cost.
- Power Grid Optimization:
 - Helps in planning electrical distribution systems to minimize the cost of connecting substations and power plants.
- Transportation Networks:
 - Used to design roads, railways, and pipeline networks with minimal construction cost.

2. Data Clustering

- Hierarchical Clustering:
 - Borůvka's algorithm can be applied to form clusters by considering the graph where nodes represent data points and edges represent distances. The MST helps identify clusters by removing the most significant edges.
- Spatial Clustering:
 - Common in geographic information systems (GIS) and satellite image analysis for grouping spatial data points.

3. Computer Networks

- Routing Protocols:
 - Used in network routing algorithms like Spanning Tree Protocol (STP) to prevent loops and optimize data flow in computer networks.
- Wireless Sensor Networks (WSNs):
 - Helps in organizing sensor nodes into energy-efficient communication structures by minimizing energy costs and ensuring reliable connectivity.

4.Distributed Systems and Parallel Computing

- Distributed Graph Algorithms:
 - Borůvka’s algorithm is inherently parallelizable, making it suitable for distributed systems where different nodes process parts of the graph simultaneously.
- Cloud Computing:
 - Used to optimize the layout of distributed systems and reduce latency in large-scale data centers.

5.VLSI Design

- Chip Layout Design:
 - Used to design efficient wiring layouts in Very Large Scale Integration (VLSI) circuits, where minimizing wire length reduces manufacturing costs and improves performance.

6.Geographic Information Systems (GIS)

- Road Network Simplification:
 - Used to simplify and optimize the connectivity of road maps by constructing MSTs that preserve essential connections.
- Pipeline Optimization:
 - Applied in laying out pipelines for water, gas, or oil distribution with minimum cost.

7.Bioinformatics

- Phylogenetic Tree Construction:
 - Used to infer evolutionary relationships between species by analyzing genetic similarities and constructing an MST for gene sequence data.
- Protein Interaction Networks:
 - Helps identify minimum interactions required to achieve connectivity in protein network analysis.

8.Image Processing

- Image Segmentation:
 - Applied to group similar regions in an image by constructing an MST based on pixel intensity differences, aiding in object detection and boundary identification.

9.Game Development

- Terrain Analysis:
 - In strategy games, Borůvka’s algorithm can be used to connect points of interest on a map with minimal path cost, such as cities or resources.
- Pathfinding:
 - Used for efficient navigation mesh construction in game environments.

10.Finance and Economics

- Supply Chain Optimization:

- Used to optimize logistics and supply chain networks to minimize transportation and distribution costs.
- Market Analysis:
 - Helps in clustering and grouping similar stocks or assets based on correlations, aiding in portfolio diversification strategies.

Why Borůvka’s Algorithm is Ideal for These Applications:

- *Scalability*: Its divide-and-conquer approach makes it suitable for large-scale graphs.
- *Parallelization*: Can be easily parallelized, ideal for distributed systems.
- *Flexibility*: Handles weighted graphs and adapts to a variety of use cases, from physical infrastructure to abstract data networks.

By leveraging Borůvka’s algorithm in these areas, industries can optimize costs, enhance efficiency, and improve system design in diverse domains.

Me: What version of ChatGPT is this?

ChatGPT: You’re interacting with ChatGPT with GPT-4 capabilities, enhanced for contextual understanding and equipped with tools like browsing, image generation, and Python-based computation. This version is designed for more advanced problem-solving, up-to-date information retrieval, and richer interaction compared to earlier iterations.

Critical Comparison

ChatGPT handles producing a report on Borůvka’s algorithm surprisingly well. The GenAI model highlights the key principles of the algorithm well and explains the steps clearly. When discussing the complexity, GPT uses V and E denoting the number of edges and vertices, however, later on it uses them as sets containing edges and vertices. This can lead to some lack of clarity, hence why my report has used $|V|$ and $|E|$ when describing the sizes of the sets. GPT highlights the union-find data structure as a way to improve the algorithm, which is fair comment on improvements. The pseudocode GPT provides is largely similar to my pseudocode, although mine has better structure and presentation, as well as some comments for clarity. GPT’s pseudocode aims to remove edges from the search space in step 3)e) although this may not be the clearest way to communicate that. GPT contradicts itself when discussing cases where $E \approx V^2$. In the Complexity section GPT says Borůvka’s is efficient in this case, but performance on dense graphs is noted as a limitation in the next section. In reality, Borůvka’s performs well on dense graphs, although selecting Prim’s or Kruskal’s may be more effective. Limitation 7, from GPT, notes handling of negative weights is a limitation, but then immediately explains that Borůvka’s can handle negative weights completely fine. In reality handling negative weights is not a limitation and GPT should have a bit more clarity on what is meant here. When listing applications, GPT finds ten applications. These applications are realistically applications of any MST algorithm and are not unique to Borůvka’s. Whilst they are accurate, the applications listed aren’t best indicative of cases you’d choose Borůvka’s over another method. My report focuses specifically on cases where Borůvka’s is beneficial. Overall, GPT produces a reasonable report on Borůvka’s algorithm, although some areas contradict themselves and other areas could use some more clarity. GPT excels at explaining the pseudocode and complexity due to its ability to decompose the algorithm. GenAI can be used to aid learning of algorithms as demonstrated by ChatGPT’s report, although care should be taken to ensure everything said is consistent and correct.