Minimum Spanning Tree using MapReduce Framework

Subhash Chandra Bose (2023202012)
Nikith Shandilya (2023201057)

1 Introduction

Minimum Spanning Tree (MST) is a fundamental problem in graph theory. It involves finding a subset of edges in a weighted, undirected graph such that:

- All vertices are connected.
- The total edge weight is minimized.

Applications of MST include network design, clustering, and circuit design.

Traditional MST algorithms include:

- **Prim's Algorithm**
- **Kruskal's Algorithm**
- **Borůvka's Algorithm**

For large-scale graphs, traditional algorithms face challenges in handling memory and computation limits. Distributed frameworks like MapReduce address these limitations by enabling parallel processing of data.

2 Traditional MST Algorithms

2.1 Kruskal's Algorithm

2.1.1 Description

Kruskal's Algorithm is a greedy algorithm that builds the MST by sorting all edges in increasing order of their weights and then adding edges one by one to the MST, ensuring that no cycles are formed. The process stops when the MST has $V - 1$ edges, where V is the number of vertices.

2.1.2 Pseudocode

```
Input: Graph G(V, E) with weights w(e)
Output: Minimum Spanning Tree T

1. Initialize T = {}
2. Sort all edges E in non-decreasing order of weights
3. For each edge (u, v) in sorted order:
        a. If adding (u, v) to T does not form a cycle:
            i. Add (u, v) to T
4. Return T
```

Listing 1: Kruskal's Algorithm

2.2 Prim's Algorithm

2.2.1 Description

Prim's Algorithm builds the MST by starting with an arbitrary vertex and iteratively adding the smallest edge that connects a vertex in the MST to a vertex outside the MST. This algorithm uses a priority queue to efficiently find the minimum-weight edge.

2.2.2 Pseudocode

```
Input: Graph G(V, E) with weights w(e)
Output: Minimum Spanning Tree T

1. Initialize T = {}, MST_set = {start_vertex}
2. While T has fewer than V-1 edges:
        a. Find the smallest edge (u, v) where u in MST_set
           and v not in MST_set
        b. Add (u, v) to T
        c. Add v to MST_set
3. Return T
```

Listing 2: Prim's Algorithm

2.2.3 Complexity

- Time Complexity: $O(E \log V)$ using a binary heap or Fibonacci heap for the priority queue.
- Space Complexity: $O(V + E)$.

2.3 Borůvka's Algorithm

2.3.1 Description

Borůvka's Algorithm is a greedy algorithm that starts with each vertex as its own component. In each iteration, the algorithm finds the smallest edge for each component and adds it to the MST. Components are merged, and the process repeats until only one component remains.

2.3.2 Pseudocode

```
Input: Graph G(V, E) with weights w(e)
Output: Minimum Spanning Tree T

1. Initialize T = {}, each vertex is its own component
2. While there are more than 1 component:
        a. For each component:
            i. Find the smallest edge (u, v) leaving the
               component
        b. Add all selected edges to T and merge components
```

```
9 3. Return T
```

Listing 3: Borůvka's Algorithm

2.3.3 Complexity

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

2.4 Comparison of Algorithms

Algorithm	Time Complexity	Space Complexity	Suitable For
Kruskal's	$O(E \log E)$	$O(V + E)$	Sparse graphs
Prim's	$O(E \log V)$	$O(V + E)$	Dense graphs
Borůvka's	$O(E \log V)$	$O(V + E)$	Parallel systems

Table 1: Comparison of MST Algorithms

2.4.1 Complexity

- Time Complexity: $O(E \log E + E\alpha(V))$, where E is the number of edges, V is the number of vertices, and α is the inverse Ackermann function (from the Union-Find structure).
- Space Complexity: $O(V + E)$.

3 Borůvka's Algorithm and MapReduce

The algorithm is efficient in parallel environments since the edge selection for each component is independent.

3.1 MapReduce Framework

MapReduce is a programming model for processing large datasets in parallel. It consists of:

- **Map Phase:** Processes input data and produces intermediate key-value pairs.
- **Reduce Phase:** Aggregates intermediate results to produce the final output.

MapReduce features fault tolerance, scalability, and simplicity, making it ideal for distributed graph algorithms.

4 Implementation

The implementation uses PySpark, a Python-based API for distributed data processing with Spark. The graph is represented as an edge list with weights.

4.1 Key Steps

1. **Finding Minimum Edges:** The map phase identifies the minimum edge for each component, and the reduce phase aggregates the results.
2. **Component Merging:** Components are updated using a Union-Find data structure with path compression.
3. **Iteration:** The process is repeated until all vertices belong to a single component.

4.2 Assumptions and Challenges

- Assumed uniform graph partitioning for balanced processing.
- Encountered challenges with:
  - **Component Convergence:** Resolved using efficient Union-Find operations.
  - **Load Balancing:** Managed by redistributing edges.
  - **Communication Overhead:** Minimized by local aggregation of intermediate results.

5 Performance Benchmarks

The implementation was tested on graphs of varying sizes. The results demonstrate scalability and efficiency in processing large datasets.

5.1 Setup

- Framework: PySpark on a cluster with 8 worker nodes.
- Dataset: Synthetic graphs with different numbers of vertices and edges.

5.2 Results

Number of Nodes	Execution Time (seconds)
10^2	0.17
10^3	1.15
10^4	19.94
10^5	40.32

Table 2: Performance of MST computation on different graph sizes.

The algorithm scales nearly linearly with the graph size. Communication overhead slightly affects performance as the graph size increases.

6 Conclusion and Future Work

The implementation of Borůvka's algorithm using the MapReduce framework demonstrates:

- Efficient processing of large-scale graphs.
- Scalability with additional computing nodes.
- Suitability for distributed environments.

Future Work:

- Explore optimizations to reduce communication overhead further.
- Evaluate other distributed graph processing frameworks, such as GraphX or Pregel.
- Apply the approach to real-world datasets to assess performance under practical constraints.