Zero : Jurnal Sains, Matematika, dan Terapan
E-ISSN : 2580-5754; P-ISSN : 2580-569X
Volume 10, Number 2, 2026
DOI: 10.30829/zero.v10i2.28487
Page: 499-510

Clean Water Pipeline Network Optimization using Minimum Spanning Tree Algorithms in Argopuro Housing, Jember

[1]Agustina Pradjaningsih
Mathematics Department, FMIPA, Universitas Jember, Jember, Indonesia

[2]Audy Putri Lestari
Administration Division, PT Sarifeed Indojaya, Banyuwangi, Indonesia.

[3]Firda Fadri
Mathematics Department, FMIPA, Universitas Jember, Jember, Indonesia

[4]Apriani Soepardi
Industrial Engineering Department, Universitas Pembangunan Nasional Veteran Yogyakarta

Article Info

Article history:

Accepted: 10 July 2026

Keywords:

Borůvka Algorithm;
Kruskal Algorithm;
Minimum Spanning Tree (MST);
Prim Algorithm;
Solin Algorithm;

ABSTRACT

Infrastructure development requires cost-efficient planning to ensure sustainable urban services. This study optimized the clean water pipeline network in Argopuro Housing, Jember, using Minimum Spanning Tree algorithms. The system was modeled as a weighted undirected graph representing junctions and pipe segments. Four algorithms Prim's, Kruskal's, Sollin's, and Borůvka's were applied to determine the optimal configuration and evaluate computational performance. All algorithms produced the same optimal pipeline length of approximately 31.00 km, representing a 40.24% reduction from the initial network. Execution time differed, with Prim's algorithm being the fastest (0.0008 s), followed by Kruskal's (0.0012 s), Borůvka's (0.0067 s), and Sollin's (0.2476 s). These results demonstrate that MST-based optimization effectively improves network efficiency. However, the study was limited to a single dataset and pipeline length as the main criterion. Future research should incorporate additional constraints and larger network scenarios for broader applicability.

Corresponding Author:

Agustina Pradjaningsih,
Mathematics Department
Universitas Jember, Jember, Indonesia
Email: agustina.fmipa@unej.ac.id

1. INTRODUCTION

Infrastructure network planning plays an important role in supporting sustainable development, particularly in public services. Inefficient infrastructure network design can result in excessive network length, higher maintenance requirements, and increased operational costs, ultimately reducing system sustainability and service reliability. As a result, mathematical optimization approaches are widely applied to support cost-effective infrastructure planning by identifying optimal network configurations that ensure full connectivity at minimal total cost. Several researchers have conducted research on public service-related infrastructure networks. Researchers [1] state that the research on the power grid confirms the effectiveness and validity of the new restoration strategy.

The national transmission network in South Sulawesi province is declared optimal by [2]. Researchers [3] obtained optimal electricity distribution channels at PT. PLN (Persero) UP3 Pematangsiantar. Optimization of *fiber-optic networks at* Tanjungpura University was carried out by [4], at the University of Bengkulu by [5], and in Tanjung Selor by [6], where the third research minimized the length of fiber-optic networks. The effectiveness of wifi networks in impacting energy efficiency is demonstrated by [7]. Researchers [8] have optimized urban mobility networks. *The* optimization of the sales transportation networks in Nigeria, Central Lombok, Ngawi, and Johannesburg is shown by [9], [10], [11], [12], respectively. Meanwhile, the optimization of the road network integrated with time and socioeconomic interaction is shown by [13]. Regarding the optimization of the water distribution pipeline network, [14] demonstrated it in Talang Belido-Muaro Jambi and [15] in Gombolharjo-Cilacap.

One of the most widely used mathematical models for network optimization is the Minimum Spanning Tree (MST), which aims to connect all nodes in a network with the minimum possible total edge weight. Infrastructure systems, such as clean water pipeline networks, can be modeled as graphs, where nodes represent junctions and edges represent physical connections. In clean water pipeline applications, the graph is typically undirected and weighted, with edge weights corresponding to pipe lengths or installation costs [16], [17], [18], [19], [20], [21], [22], [23], [24]. Due to its mathematical simplicity and guaranteed optimality for weighted undirected graphs, MST has been extensively implemented in transportation systems, energy networks, telecommunications, and water distribution planning.

Several algorithms have been developed to solve MST problems, including Prim's, Kruskal's, Sollin's, and Borůvka's algorithms [25], [26], [27], [28], [29], [30], [31], [32]. Although these algorithms theoretically produce the same optimal spanning tree for a given weighted graph, they differ in computational mechanisms, execution time, and implementation structure. Recent studies have highlighted the importance of evaluating not only solution optimality but also computational efficiency, particularly in large-scale infrastructure systems. However, most previous research has been conducted using simulated datasets, abstract network models, or non-operational systems.

Despite the growing body of literature on MST applications, there remains a clear research gap in the comparative implementation of multiple MST algorithms on real operational clean water pipeline networks in Indonesia. Existing Indonesian studies generally focus on single-algorithm applications or conceptual demonstrations, without systematically comparing algorithmic performance using real pipeline data from local water utilities. Moreover, limited empirical evidence is available regarding how different MST algorithms perform computationally when applied to actual urban distribution networks characterized by dense residential layouts and spatial irregularities.

In Indonesia, clean water distribution systems are managed by regional water utilities (Perusahaan Daerah Air Minum/PDAM). Network efficiency directly influences service reliability, water loss, and operational expenditure. The clean water pipeline network in Sumbersari Subdistrict, Jember, managed by PERUMDAM Tirta Pandalungan, features multiple cycles, indicating redundant pipe connections and a non-optimal total pipeline length. Although such redundancy increases installation and maintenance costs, no prior study has systematically optimized this specific network using MST-based approaches while simultaneously evaluating and comparing the computational performance of several classical MST algorithms.

The novelty of this study lies in its empirical and comparative evaluation of four classical MST algorithms—Prim's, Kruskal's, Sollin's, and Borůvka's—applied to an actual clean water pipeline network in an Indonesian urban residential area. Unlike prior studies that rely on simulated or theoretical datasets, this research integrates real operational data and emphasizes both optimization effectiveness and computational efficiency. This dual focus provides practical guidance to infrastructure planners and local water utilities on selecting appropriate algorithms for cost-efficient, scalable network optimization.

Therefore, this study aims to optimize the clean water pipeline network in Argopuro Housing, Sumbersari District, Jember, using Minimum Spanning Tree algorithms. Specifically, this research compares the performance of Prim's, Kruskal's, Sollin's, and Borůvka's algorithms in terms of optimal pipeline length and execution time to support more efficient, evidence-based decision-making in clean water distribution planning.

2. RESEARCH METHOD

2.1 Research Design

This study employed a quantitative and computational approach to optimize a clean water pipeline network using graph-based methods. The objective was to evaluate and compare the performance of Minimum Spanning Tree (MST) algorithms in minimizing total pipeline length while maintaining full network connectivity.

The pipeline network was represented as a weighted undirected graph $G = (V, E)$, where V denotes the set of nodes corresponding to pipeline junctions, and E denotes the set of edges representing pipe segments. Each edge was assigned a weight equal to its physical length in kilometers. This representation enabled the application of MST algorithms to determine the optimal network configuration.

Four classical MST algorithms—Prim's, Kruskal's, Sollin's, and Borůvka's—were implemented and applied to the same dataset under identical computational conditions. For each algorithm, the procedure was defined explicitly: (1) initialize the graph representation, (2) iteratively select edges based on algorithm-specific criteria while avoiding cycles, (3) continue the process until all nodes are connected, and (4) compute the total edge weight and execution time. This clear termination condition replaces ambiguous expressions and ensures reproducibility.

To ensure fair comparison, all algorithms were executed using the same input data, graph structure, and computational environment. The primary performance metric was total pipeline length, and execution time (in seconds) was used to evaluate computational efficiency.

Several assumptions were adopted to focus on structural optimization. Edge weights were based solely on pipe length, and the model did not incorporate hydraulic constraints, variations in pipe diameter, or demand distribution. Additionally, the undirected graph assumption implied bidirectional flow during planning.

2.2 Data Source and Variables

The clean water pipeline network data used in the study are secondary data obtained from PERUMDAM Tirta Pandalungan Jember during the period from September to December 2024. The dataset is obtained from the utility's operational database, which records the infrastructure layout and network attributes used for planning and maintenance. This includes information about pipe junctions (nodes), pipe connections (edgings), geographic coordinates, and pipe lengths, reflecting actual field conditions as shown in Figure 1.

Figure 1. Pipeline Network

Coordinate data is provided in a geographic coordinate system (latitude-longitude) and standardized during preprocessing to ensure consistent spatial representation. This step is necessary to maintain accurate mapping and prevent distortion during network topology reconstruction, as summarized in Table 1.

Table 1. Pipeline Network Data

Edge	TA	Longitude_TA	Latitude_TA	TT	Longitude_TT	Latitude_TT	Weight
e1	v1	113.704607	-8.175343819	v2	113.7035557	-8.17527291	0.11979
e2	v2	113.7035557	-8.17527291	v5	113.7035136	-8.176235779	0.10435
e3	v1	113.704607	-8.175343819	v6	113.7045741	-8.17621289	0.09798
e4	v5	113.7035136	-8.176235779	v6	113.7045741	-8.17621289	0.11448
e5	v2	113.7035557	-8.17527291	v3	113.7027894	-8.175332624	0.08388
...	...	...	...	...	...	...	...
e656	v291	113.7042307	-8.189968641	v293	113.7039175	-8.189940214	0.03301
e657	v293	113.7039175	-8.189940214	v295	113.704148	-8.189220363	0.08235
e658	v292	113.7036793	-8.189894754	v293	113.7039175	-8.189940214	0.02172
e659	v292	113.7036793	-8.189894754	v294	113.7039031	-8.189152714	0.08484
e660	v271	113.7018405	-8.190800761	v275	113.7070947	-8.191336863	0.58158

Before analysis, data preprocessing was conducted to improve data quality. Duplicate entries were removed, incomplete records were verified against original documentation, and coordinate values were standardized. The network topology was then reconstructed programmatically to ensure proper connectivity among nodes and to avoid unintended isolated components.

The pipeline network was modeled as a weighted undirected graph $G = (V, E)$, where V represents junction points, and E represents pipe segments connecting pairs of nodes [16], [18], [19], [20]. The undirected representation assumes bidirectional flow at the planning stage, a common approach in distribution network modeling.

Edge weights were defined based on pipe length (km), serving as a proxy for infrastructure cost. The primary variable analyzed was the total pipeline length, while execution time (seconds) was used to evaluate algorithm performance.This study has several data-related limitations. The dataset represents a single residential network, which may not capture the variability of larger or more complex systems. Additionally, potential biases may arise from historical infrastructure decisions or measurement tolerances. Therefore, the results should be interpreted within the context of similar network characteristics, and further validation on larger datasets is recommended.

2.3 Analytical Methods and Algorithm

The analytical framework of this study was based on graph theory and Minimum Spanning Tree optimization. The initial pipeline network was first constructed as a weighted undirected graph. Subsequently, four MST algorithms—Prim's, Kruskal's, Sollin's, and Borůvka's algorithms—were applied to the same graph to ensure consistency in comparison.

Prim's algorithm was implemented by iteratively adding the minimum-weight edge connected to the growing tree without creating cycles. According to [24],[26],[29],[30], Prim's algorithm finds the MST by connecting nodes by considering the distance with minimum weight, which will produce a single tree. Prim's algorithm begins by initiating a graph $G = (V, E)$ where the set V of nodes and the set E of edges are $\emptyset$ and continues:

a. Select an initial vertex and insert it into the graph V.
b. The minimum weighted edge (u, v) with the closest distance to the starting nodes is selected.
c. Select another minimum weight edge (u', v') where $u \in V$ and $v \notin V$ without causing the tree to cycle.
d. The above steps are repeated until the total edges in the MST meet $|E| = n - 1$, where n is the number of nodes.

Kruskal's algorithm was executed by sorting all edges in ascending order and selecting the smallest edge that did not form a cycle. Kruskal's algorithm works by adding the edge with the smallest weight. The MST generated by Kruskal will not have a cycle because the added edge is an edge that does not result in the formation of a cycle in the tree [23],[26],[29]. In a graph $G = (V, E)$ the edges in the formation of MST with Kruskal's algorithm are sorted by their weights, and $w(e_1) \leq w(e_2) \leq \cdots \leq (w(e_m)$ then proceed as follows:

a. $\forall v \in V$ a separate set will be created $\forall v \in V, C(v) = \{v\}$.
b. Select the edge with the smallest weight that does not cause a cycle $C(u) \neq C(v)$, then add the edge to the MST.
c. Components are merged until all nodes are connected in the MST.
d. Repeat until the total edges in the MST meet $|E| = n - 1$, where n is the number of nodes.

Sollin's algorithm was applied by identifying and removing the largest-weight edges that formed cycles while maintaining network connectivity. . Based on the explanation from [28], Sollin's algorithm will produce an MST with no cycle so that the resulting tree has the smallest weight. This algorithm runs by removing the most significant edge, the existence of which results in the formation of a cycle. The edges in a graph $G(V, E)$ are sorted by their weights $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$ and proceed with the following algorithm:

a. The sorted edges will be checked one by one from the largest to see whether they have a cycle.
b. If the edge is detected to have a cycle, then the edge will be deleted, provided that the deleted edge does not cause the nodes in the tree to become unconnected. Meanwhile, if the edge does not have a cycle, then the edge will be left and included in the MST.
c. The steps are repeated until the total edges in the MST meet $|E| = n - 1$.

Borůvka's algorithm was executed by initially treating each node as an independent component and repeatedly merging components by selecting the minimum outgoing edge from each node. According to [21],[23],[24],[25],[26] the explanation of Boruvka's algorithm works in an iterative way, where each node will choose the closest node connected by the edge with the smallest weight, so that a tree will be formed. The trees will be merged into the set of MST(T), then:

a. $\forall$ node $u \in V$, the node selects the edge (u, v) with the smallest weight to connect (u) to another node (v).
b. The selected edges are added to the set T.
c. Merge the vertices (u) and (v) connected by edges T into one main tree.

d. Repeat the process until there is only one tree that includes all the vertices.

For each algorithm, two performance indicators were evaluated: (1) the total length of the resulting MST, and (2) the execution time required to obtain the solution. All algorithms were applied under identical conditions to ensure objective comparison.

2.4 Software and Tools

All computations and graph constructions were performed using the Python programming language. Python was selected for its flexibility, computational efficiency, and the availability of graph-processing libraries. The pipeline network graph was constructed programmatically, and each MST algorithm was implemented using standard algorithmic procedures. Execution time was recorded automatically during the computational process to evaluate algorithm efficiency. The following Python pseudocode describes the fourth approach to the algorithm [33].

```
# Prim's Algorithm
def prim(G, start_node):
    E = []
    V = set([start_node])
    edges = [(weight, start_node, v) for v, weight in G[start_node].items()]
    heapq.heapify(edges)
    while edges and len(V) < len(G):
        weight, u, v = heapq.heappop(edges)
        if v not in V:
            V.add(v)
            E.append((u, v, weight))
            for next_v, next_weight in G[v].items():
                if next_v not in V:
                    heapq.heappush(edges, (next_weight, v, next_v))
    return E
```

```
class DisjointSet: # Disjoint Set (Union-Find) - Kruskal
    def __init__(self, vertices):
        self.parent = {v: v for v in vertices}
        self.rank = {v: 0 for v in vertices}
    def find(self, v):
        if self.parent[v] != v:
            self.parent[v] = self.find(self.parent[v])
        return self.parent[v]
    def union(self, u, v):
        root_u = self.find(u)
        root_v = self.find(v)
        if root_u == root_v:
            return False
        if self.rank[root_u] < self.rank[root_v]:
            self.parent[root_u] = root_v
        elif self.rank[root_u] > self.rank[root_v]:
            self.parent[root_v] = root_u
        else:
            self.parent[root_v] = root_u
            self.rank[root_u] += 1
        return True

def kruskal(G): # Kruskal's Algorithm
    edges = []
    for u in G:
        for v, weight in G[u].items():
            if u < v:
                edges.append((weight, u, v)), edges.sort()
    ds = DisjointSet(G.keys())
    mst = []
    for weight, u, v in edges:
        if ds.union(u, v): mst.append((u, v, weight))
        if len(mst) == len(G) - 1:
            break
    return mst
```

```
# Sollin's Algorithm
def sollin(G):
    edges = sorted(G.edges(data=True), key=lambda x: x[2]
     ['weight'], reverse=True)
    removed_edges = []
    remaining_edges = list(G.edges(data=True))
    for edge in edges:
        u, v, w = edge
        def is_connected(G):
            return nx.is_connected(G)
        G.remove_edge(u, v)
        if not is_connected(G):
            G.add_edge(u, v, weight=w['weight'])
        else:
            removed_edges.append(edge)
            remaining_edges.remove(edge)
    return G, removed_edges, remaining_edges
```

```
class DisjointSet: # Disjoint Set (Union-Find) - Borůvka
    def __init__(self, vertices):
        self.parent = {v: v for v in vertices}
    def find(self, v):
        if self.parent[v] != v:
            self.parent[v] = self.find(self.parent[v])
        return self.parent[v]
    def union(self, u, v):
        root_u = self.find(u)
        root_v = self.find(v)
        if root_u != root_v:
            self.parent[root_v] = root_u
            return True
        return False
def boruvka(G): # Boruvka's Algorithm
    ds = DisjointSet(G.keys())
    num_components = len(G)
    mst = []
    while num_components > 1:
        cheapest = {}
        for u in G:
            for v, weight in G[u].items():
                set_u = ds.find(u)
                set_v = ds.find(v)
                if set_u != set_v:
                    if set_u not in cheapest or cheapest[set_u][0] >
weight:
                        cheapest[set_u] = (weight, u, v)
                    if set_v not in cheapest or cheapest[set_v][0] >
weight:
                        cheapest[set_v] = (weight, v, u)
        for weight, u, v in cheapest.values():
            if ds.union(u, v):
                mst.append((u, v, weight))
                num_components -= 1
    return mst
```

3. RESULT AND ANALYSIS

3.1 Initial Graph/Network Analysis

The initial network was modeled in Python as a weighted undirected graph (Figure 2), showing dense connections and multiple cycles within residential clusters. These redundancies increased the total length to 51.876 km without improving coverage, justifying the use of MST algorithms to achieve a more optimal and efficient network configuration.

3.2 Results of MST Algorithms

3.2.1 Prim's Algorithm

Prim's algorithm generated an MST (Figure 3) using a greedy vertex-based approach, iteratively selecting minimum-weight edges without forming cycles. The resulting network significantly reduced pipeline length while maintaining full connectivity. It also showed the fastest execution time, indicating high computational efficiency and practical suitability for pipeline optimization.

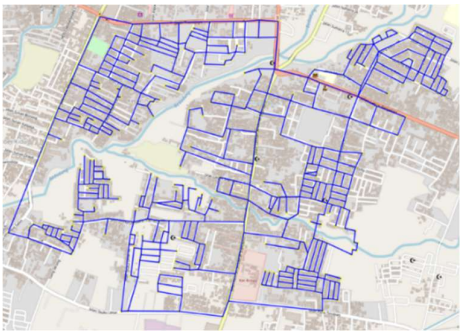

Figure 2. Initial Graph

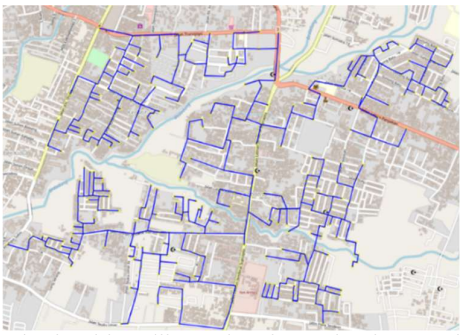

Figure 3. MST result using Prim's algorithm: illustrating the optimal network topology produced through a vertex-based greedy expansion process.

3.2.2 Kruskal's Algorithm

Kruskal's algorithm generated an MST (Figure 4) by selecting edges in ascending weight while avoiding cycles. The resulting network matched Prim's optimal solution, confirming MST consistency. Although slightly slower than Prim's, it remained efficient and suitable for large-scale applications, effectively ensuring connectivity while eliminating redundant pipeline segments.

3.2.3 Sollin's Algorithm

Sollin's algorithm produced an MST (Figure 5) by iteratively removing cycle-forming edges while preserving connectivity. The resulting network matched the optimal length of other algorithms, confirming theoretical consistency. However, it showed significantly longer execution time, indicating lower computational efficiency and limited practicality for large-scale network optimization.

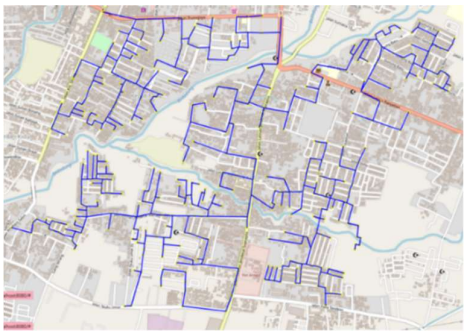

Figure 4. MST result using Kruskal's algorithm: illustrating the optimal network topology obtained through an edge-sorting and cycle-avoidance process.

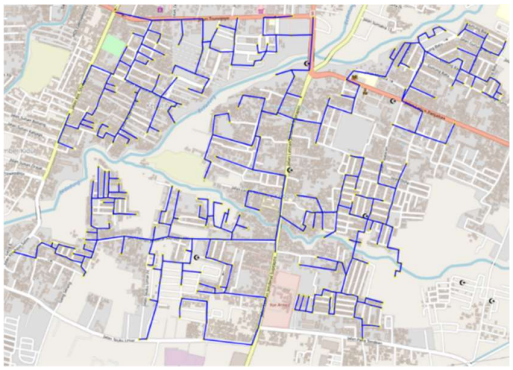

Figure 5. MST result using Sollin's algorithm: illustrating the optimal network topology obtained through an iterative cycle-elimination process.

3.2.4 Borůvka's Algorithm

Borůvka's algorithm generated an MST configuration (Figure 6) using a component-merging process. The resulting network achieved the same optimal length as other algorithms, confirming solution consistency. In terms of performance, it was faster than Kruskal's and Sollin's but slightly slower than Prim's, indicating a balanced computational efficiency.

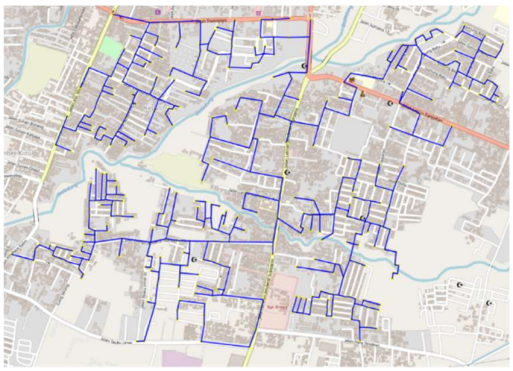

Figure 6. MST result using Borůvka's algorithm: illustrating the convergence of component-based merging into a single optimal spanning tree.

3.2.5 Execution Time Interpretation and Statistical Considerations

The execution times of the MST algorithms were obtained from single deterministic runs under identical computational conditions. As these algorithms produce consistent results for the same input and environment, repeated executions yield identical timing outcomes. Therefore, inferential statistical tests such as t-tests or ANOVA were not applied, since they require multiple independent observations.

Instead, the comparison focused on descriptive performance analysis. Execution times ranged from 0.0008 to 0.2476 seconds, with substantial variability across algorithms. Sollin's algorithm required significantly longer computation—approximately 300 times slower than Prim's algorithm—reflecting differences in computational mechanisms rather than random variation.

Although these differences appear small for a medium-scale network, they may become significant in large-scale or repeatedly optimized systems. Future studies should consider repeated experiments across diverse datasets and computational environments to enable statistical validation and assess algorithm scalability more comprehensively.

3.3 Comparative Performance Analysis

A comparative summary of the four Minimum Spanning Tree algorithms is presented in Table 2. All algorithms produced the same optimal pipeline length, reducing the total network length from 51.876 km to 31.000 km, corresponding to a 40.24% improvement over the initial configuration. This confirms that MST optimization effectively minimizes the length of the infrastructure while maintaining full network connectivity.

Table 2. Comparison of Prim's, Sollin's, Kruskal's, and Boruvka's algorithms

Algorithm	Initial Length (km)	Length After MST (km)	Time (seconds)
Prim's algorithm	51.876	31.000	0.0008
Kruskal's algorithm	51.876	31.000	0.0012
Sollin's algorithm	51.876	31.000	0.2476
Boruvka's algorithm	51.876	31.000	0.0067

Although the resulting network structures were identical, differences in computational performance were observed. Prim's algorithm showed the fastest execution time, followed by Borůvka's and Kruskal's algorithms, while Sollin's algorithm required substantially longer computation due to its iterative cycle-elimination process.

From a practical perspective, computational efficiency becomes important when optimization is applied to larger networks or repeated planning scenarios. While all algorithms guarantee the same optimal solution, selecting a computationally efficient method can improve operational feasibility. In this study, Prim's algorithm demonstrated the most efficient performance for practical implementation.

3.4 Data Characteristics, Robustness, and Potential Bias

The analysis was conducted using a real-world clean water pipeline dataset from Argopuro Housing, which represents a residential distribution network with specific spatial characteristics, node density, and connectivity patterns. These structural properties may differ from pipeline systems in other urban or regional contexts. Consequently, network size, edge density, and spatial configuration can influence the computational behavior and execution time of MST algorithms.

Although all algorithms produced identical optimal network topologies—confirming the theoretical equivalence of MST solutions for weighted undirected graphs—the observed computational performance should be interpreted within the context of the analyzed dataset.

Using real operational data improves the practical relevance of the findings by demonstrating that MST-based optimization can be applied effectively in an actual water distribution system. However, broader validation is recommended. Future research should apply the same comparative approach to multiple pipeline networks with varying sizes and complexities to better evaluate algorithm scalability, robustness, and performance across diverse infrastructure conditions.

3.5 Discussion and Implications

The results demonstrate that the optimized network configuration can serve as a structural reference for evaluating existing distribution layouts, planning network expansion in residential areas, or identifying redundant pipeline segments. The identical network configurations produced by all four algorithms confirm the theoretical property of MST optimality in weighted undirected graphs and validate the reliability of the computational implementation. Therefore, algorithm selection becomes relevant for improving computational efficiency in large-scale infrastructure planning.

From a practical perspective, these findings provide useful insights for water utility agencies responsible for infrastructure planning and maintenance. Furthermore, the relatively simple computational structure of MST algorithms makes them suitable for integration into digital planning tools such as geographic information systems (GIS) or decision-support platforms used in urban infrastructure management.

Future research may extend this approach by incorporating additional engineering constraints, including variations in pipe diameter, hydraulic pressure requirements, flow capacity, and heterogeneous installation costs. Applying the comparative framework to larger, more complex pipeline systems could also provide further insight into algorithmic scalability. Integrating MST-based structural optimization with hydraulic simulation models would enable the development of more comprehensive decision-support tools for water distribution management.

4. CONCLUSION

This study demonstrated the effectiveness of Minimum Spanning Tree (MST) algorithms in optimizing a real-world clean water pipeline network in Argopuro Housing, Jember. The network was modeled as a weighted undirected graph and analyzed using four classical MST algorithms: Prim's, Kruskal's, Sollin's, and Borůvka's. The results confirmed that all algorithms generated the same optimal spanning tree while successfully reducing structural redundancy and maintaining full network connectivity. Although the optimal solutions were identical, differences in computational efficiency were observed, indicating that algorithm selection may influence operational performance, particularly in repeated optimization tasks or larger infrastructure systems.

This research provides empirical evidence that MST-based optimization can support efficient and systematic planning of water distribution networks using real operational data. However, the study considered only pipeline length as the optimization criterion. Future research should incorporate additional engineering constraints, such as installation and operational costs, variations in pipe diameter, hydraulic performance, and demand distribution, to develop more comprehensive and practical network optimization models.

ACKNOWLEDGEMENT

The authors would like to express their appreciation to PERUMDAM Tirta Pandalungan Jember for providing the clean water pipeline network data used in this study. The authors also gratefully acknowledge the institutional support provided by the Lembaga Penelitian dan Pengabdian Kepada Masyarakat (LP2M), University of Jember, through the Internal Grant scheme of the Research Group–Community Service (Keris-Dimas), which facilitated the provision of research resources and facilities necessary for the completion of this work. In addition, the authors acknowledge the academic contributions and constructive discussions from members of the Mathematical Optimization and Computations (MOCo) research group in the Department of Mathematics, whose insights and suggestions enhanced the quality of this research.

5. REFERENCES

[1] A. Łukaszewski and Ł. Nogal, "The Application of the Modified Prim ' s Algorithm to Restore the Power System Using Renewable Energy Sources," *Symmetry (Basel).*, vol. 14, no. 5, 2022, doi: 10.3390/sym14051012.

[2] K. Kusnadi, W. Gata, and F. N. Arviantino, "Application of Kruskal and Sollin Algorithms on the National Transmission Network of South Sulawesi Province," *METIK J. (Media Teknol. Inf. dan Komputer).*, vol. 6, no. 1, 2022, doi: 10.47002/metik.v6i1.260.

[3] N. W. Wisesa and R. fitri Mawardiningrum, "Analysis of the Use of the Boruvka Algorithm Method in Electricity," *Int. J. Sci. Math. Educ.*, vol. 1, no. 1, pp. 29–35, 2024, doi: 10.62951/ijsme.v1i1.14.

[4] N. J. Triami, Y. Yundari, and F. Fran, "MINIMUM SPANNING TREE PADA JARINGAN FIBER OPTIC DI UNIVERSITAS TANJUNGPURA," *Bimaster Bul. Ilm. Mat. Stat. dan Ter.*, vol. 09, no. 1, pp. 223–230, 2020, doi: 10.26418/bbimst.v9i1.38909.

[5] R. Efendi, B. Susilo, and Y. A. Prasetyo, "Perbandingan Algoritma Boruvka dan Algoritma Sollin pada Optimasi Kebutuhan Kabel Fiber Optik Universitas Bengkulu," *J. Sci. Appl. Informatics*, vol. 4, no. 2, pp. 175–181, 2021, doi: 10.36085/jsai.v4i2.1623.

[6] S. Syahdan and A. Efendy, "Penerapan Algoritma Sollin Pada Jaringan Kabel Telkom Tanjung Selor Berbantu Maple," *JSB J. Sains Benuanta*, vol. 2, no. 1, pp. 1–8, 2023, doi: 10.61323/jsb.v2i1.

[7] H. M. Saad *et al.*, "Enhancing energy balance in wireless sensor networks through optimized minimum spanning tree," *PeerJ Comput. Sci.*, vol. 10, pp. 1–26, 2024, doi: 10.7717/peerj-cs.2269.

[8] D. A. Samalna, V. R. Nwokam, J. M. Ngossaha, I. Tchappi, A. A. A. Ari, and K. Kolyang, "Optimization of cyber-physical urban mobility systems in developing countries: A dependency structure matrix approach with advanced artificial intelligence techniques," *J. Infrastruct. Policy Dev.*, vol. 8, no. 14, p. 8126, Nov. 2024, doi: 10.24294/jipd8126.

[9] A. Tamber, F. Ikpotokin, and O. Linus, "The Minimum Spanning Tree of the Nigeria Roads Network through Multiple-Roads Network System," *Niger. Ann. PURE Appl. Sci.*, vol. 3, no. 2, pp. 151–157, Jul. 2020, doi: 10.46912/napas.170.

[10] N. Made, A. Ulandari, and S. Subarinah, "Implementasi Algoritma Kruskal dalam Menentukan Rute Terdekat pada Tempat Pariwisata di Daerah Lombok Tengah," *Griya J. Math. Appl.*, vol. 1, pp. 578–589, 2021, doi: 10.29303/griya.v1i4.117.

[11] A. Ahsanti, A. Insyafilla, N. N. Fatimah, W. C. D. Wahyuni, and D. Rahmadi, "Optimalisasi Jaringan Jalan Antar Kecamatan dengan Minimum Spanning Tree dan Algoritma Prim di Kabupaten Ngawi," *Basis J. Ilm. Mat.*, vol. 4, no. 1, pp. 1–11, Mar. 2025, doi: 10.30872/basis.v4i1.1451.

[12] T. Moyo, A. Y. Kibangou, and W. Musakwa, "Societal context-dependent multi-modal transportation network augmentation in Johannesburg, South Africa," *PLoS One*, vol. 16, no. 4, p. e0249014, Apr. 2021, doi: 10.1371/journal.pone.0249014.

[13] H. Wen, Y. Ye, and L. Zhang, "Optimizing road networks in underdeveloped regions for improving comprehensive efficiency integrated by accessibility, vulnerability and socioeconomic interaction," *Reliab. Eng. Syst. Saf.*, vol. 243, p. 109848, Mar. 2024, doi: 10.1016/j.ress.2023.109848.

[14] I. S. Sinaga, N. Rarasati, W. Syafmen, and G. Kholijah, "MST dalam Perencanaan Jaringan Pipa Air Minum dengan Perbandingan Matriks Ketetanggaan Berbobot dan Algoritma Sollin," *FIBONACCI J. Pendidik. Mat. dan Mat.*, vol. 9, no. 2, pp. 179–196, 2023, doi: 10.24853/fbc.9.2.179-196.

[15] W. I. Ilahy, M. Ahmad, and B. P. Hartono, "Optimasi Jaringan Distribusi Air di Desa Gombolharjo Menggunakan Algoritma Prim," *JaMES J. Math. Educ. Sci.*, vol. 6, no. 2, pp. 177–183, 2023, doi: 10.32665/james.v6i2.1896.

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

[17] A. K. Nemani and R. K. Ahuja, "Minimum Spanning Trees," *Wiley Encycl. Oper. Res. Manag. Sci.*, 2011, doi: 10.1002/9780470400531.eorms0816.

[18] S. Pettie, "Minimum Spanning Trees," *Encycl. Algorithms*, pp. 1322–1325, 2016, doi: 10.1007/978-1-4939-2864-4_239.

[19] H. A. Taha, *Operations Research An Introduction*. 2017. [Online]. Available: https://elibrary.pearson.de/book/99.150005/9781292165561

[20] J. Joseph B. Kruskal, *17: On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*. 1956. doi: 10.7551/mitpress/12274.001.0001.

[21] Paryati and S. Krit, "The Implementation Of Kruskal's Algorithm For Minimum Spanning Tree In A Graph," *MATEC Web Conf. 348*, vol. 01001, pp. 1–13, 2021, doi: 10.1051/matecconf/202134801001.

[22] M. Alshammari, "Maintaining Minimum Spanning Trees in Fully Dynamic Graphs," *Int. J. Intell. Syst. Appl. Eng.*, vol. 12, no. 4, pp. 1562–1567, 2024, [Online]. Available: https://ijisae.org/index.php/IJISAE/article/view/6451

[23] L. Šubelj, "Computing Well-Balanced Spanning Trees of Unweighted Networks," *Algorithms*, vol. 18, no. 2, p. 760, 2025, doi: 10.3390/a18120760.

[24] K. Tapp, "On the minimum spanning tree distribution in grids," *Eur. J. Comb.*, vol. 133, no. March, 2026, doi: 10.1016/j.ejc.2025.104325.

[25] N. R. Latha, G. Shyamala, and G. R. Prasad, "Exploring the parallel implementations of the three classical MST algorithms," in *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2017*, 2017, pp. 340–346. doi: 10.1109/ICICCT.2017.7975216.

[26] D. Rachmawati, Herriyance, and F. Y. P. Pakpahan, "Comparative Analysis of the Kruskal and Boruvka Algorithms in Solving Minimum Spanning Tree on Complete Graph.," in *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics, DATABIA 2020 - Proceedings (2020)*, 2020, pp. 55–62. doi: 10.1109/DATABIA50434.2020.9190504.

[27] 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. 012043, 2020, doi: 10.1088/1742-6596/1462/1/012043.

[28] N. R. Brien, "A formal correctness proof of Boruvka's minimum spanning tree algorithm," 2020. doi: 10.26021/10196.

[29] J. Chen, "The analysis and application of Prim algorithm , Kruskal algorithm , Boruvka algorithm," *Appl. Comput. Eng.*, vol. 19, no. 1, pp. 84–89, 2023, doi: 10.54254/2755-2721/19/20231012.

[30] R. Zhang, "The comparison of three MST algorithms," *Appl. Comput. Eng.*, pp. 191–197, 2023, doi: 10.54254/2755-2721/17/20230939.

[31] D. Kurniawan, W. Wamiliana, and C. S. N. Fauzi, "Perbandingan kompleksitas algoritma prim, algoritma kruskal, dan algoritma sollin untuk menyelesaikan masalah minimum spanning tree Title," *Komputasi*, vol. 2, no. 1, pp. 60–67, 2014, doi: 10.23960/komputasi.v2i1.1005.

[32] S. Gupta, A. Mahajan, S. Shah, and V. Negi, "Exploring the Maze: A Comparative Study of Prims and Kruskals MST Algorithms," in *15th International Conference on Computing Communication and Networking Technologies, ICCCNT 2024 (2024)*, 2024, pp. 1–5. doi: 10.1109/ICCCNT61001.2024.10726020.

[33] W. McKinney, *Python for Data Analysis*, 3rd ed. 2018. [Online]. Available: https://wesmckinney.com/book/