Journal of Engineering Innovations & Technology

journal homepage: www.journal-eit.online

Paper Type: Original Article

Intelligent Optimization Algorithm Based on Minimum Steiner Tree

Yuhong Xiao[1#], Yulin Chen[2#], Kai Li[2#], Ziqiang Gong[3&], Zongze HE[4&], Jiahao Li[3*]

[1]*Shenzhen Senior High School Group International Division, Shenzhen, China*

[2]*University College London, London, United Kingdom*

[3]*Shenzhen Aurora International Academy, Shenzhen, China*

[4]*Fairmont Schools, Anaheim, United States*

[#]Co-first author

[&]Co-second author

[*]Corresponding Author: **Jiahao Li**

Abstract

This study explores the application and comparison of the Minimum Steiner Tree (STP) algorithm and the Minimum Spanning Tree (MST) algorithm in optimizing the communication and transportation network in the central area of Kanazawa City. Through a detailed analysis of the classic STP problem and related algorithms, this paper proposes an optimized Ant Colony Optimization algorithm (IACO) and verifies its efficiency on standard datasets. The results show that this algorithm significantly reduces the total connection cost and improves network efficiency and robustness when connecting key nodes. Therefore, this paper suggests that the IACO algorithm has significant application value in urban network planning and recommends further exploration of its potential in larger and more complex networks to optimize the design of urban communication and transportation networks.

Keywords: Minimum Steiner Tree (STP), Network Optimization, Heuristic Algorithm, Graph Theory, Minimum Spanning Tree (MST), Ant Colony Optimization (ACO)

Corresponding Author: Jiahao Li

1 | Introduction

Network design is a critical branch in the field of information technology. It involves constructing and optimizing the structure and functionality of communication networks to meet specific performance requirements and business needs. This includes determining the network topology, selecting appropriate transmission media, configuring network devices, and ensuring the network's reliability, security, and scalability. With the rapid development of the internet and mobile communication technologies, network design plays a crucial role in supporting modern communication, data transmission, cloud computing services, and the Internet of Things (IoT). Designing an efficient network requires not only considering cost-effectiveness but also ensuring that the network can adapt to the constantly changing technological environment and user demands while maintaining high flexibility and connectivity.

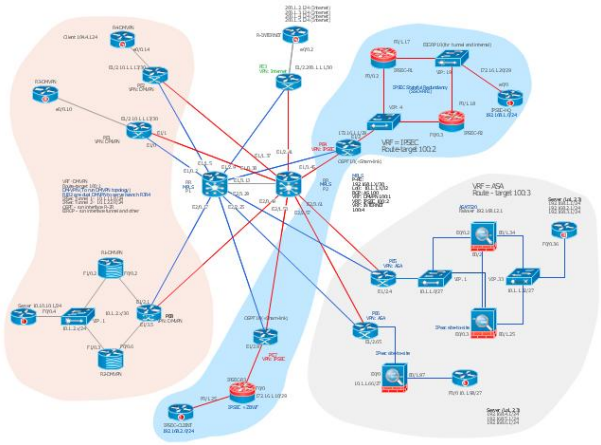

Figure 1: Background of Network Design Problem Research

In the academic field, research on network design problems typically employs a series of rigorous scientific methods. Researchers often use mathematical programming techniques, such as Linear Programming (LP) and Integer Programming (IP), to formalize network design problems and seek optimal solutions. These methods effectively address resource allocation and cost minimization issues. Additionally, graph theory algorithms play a central role in network topology design, particularly the Minimum Spanning Tree (MST) algorithm and shortest path algorithms like Dijkstra or Floyd, which provide efficient solutions for establishing physical or logical network connections.

As the scale and complexity of the problems increase, traditional exact algorithms may become computationally infeasible. Therefore, researchers have turned to heuristic and metaheuristic algorithms, such as Genetic Algorithms (GA), Simulated Annealing (SA), and Ant Colony Optimization (ACO). These algorithms can provide high-quality approximate solutions for large-scale problems within a reasonable time frame. Their parallelism and flexibility make them particularly popular in modern network design challenges.

Network design is crucial in modern communication and data transmission, especially in a vital transportation and communication hub like Kanazawa City. However, current methods face limitations in handling complex network topologies and reducing overall costs. This study, through the Minimum Steiner Tree algorithm, particularly the optimized Ant Colony Optimization algorithm, aims to address these challenges and enhance the efficiency and robustness of the network.

2 | Literature Review

This section should be concise and define the background and significance of the research by considering the relevant literature, particularly the most recent publications. When preparing the introduction, please bear in mind that some readers will not be experts in your field of research.

The Steiner Tree Problem (STP) is a classic combinatorial optimization problem and a type of shortest network problem.It is one of Karp's 21 NP-complete problems. Over the past few decades, extensive theoretical research has been devoted to this problem. For instance, current studies show that the STP can be solved in time $O\left(2^{|T|}n^2 \cdot n \cdot m\right)$, where n, m, and $|T|$ represent the number of vertices, edges, and terminals, respectively [1]. The current best approximation ratio is 1.39 [2], and there is no 1.01-approximation algorithm (unless P=NP) [3].

In recent years, many scholars have made efforts in the field of connectivity restoration in wireless sensor networks. Abbasi et al. [4] proposed the Distributed Actor Recovery Algorithm (DARA), which selects existing nodes by considering factors such as distance, node degree, and node labels, and then moves the selected nodes to designated areas to restore connectivity. Senel et al. [5,6] applied biomimetic research based on spider web structures to connectivity restoration in wireless sensor networks, proposing the 1C-SpiderWeb algorithm. This algorithm uses a quasi-mesh topology to restore connectivity in paralyzed areas, but it requires a large number of relay nodes (RNs), increasing the cost of connectivity restoration. Additionally, the complexity of this topology leads to poor fault tolerance. Lee et al. [7] proposed an algorithm that uses the minimum Steiner tree to restore wireless sensor networks experiencing multiple simultaneous failures. The algorithm uses the Steiner tree as the initial topology and studies the placement of RNs, calculating the minimum number of relay nodes needed. Connectivity is restored by moving relay nodes, but the algorithm cannot guarantee redundancy-free movement of RNs or the robustness of the sensor network after connectivity restoration. In 2011, Senel et al. [8] proposed the FeSTA (federate network Segments via triangular Steiner Tree Approximation) algorithm, which uses a triangular Steiner tree structure for network disconnection recovery. The characteristics of the triangular Steiner tree structure effectively reduce the number of deployed relay nodes.

On the other hand, some research has been conducted on approximation algorithms based on the Minimum Spanning Tree (MST). Kou L et al. [9] proposed a simple approximation algorithm based on MST properties. This algorithm constructs a minimum spanning tree and then prunes it

by removing non-terminal leaf nodes to obtain a Steiner tree. However, the approximation ratio of this algorithm is only 2. Robins G et al. applied a compression strategy to this method and achieved an approximation ratio of 1.55. Aragão M P D et al. [10] made some improvements based on the classic Prim and Kruskal MST algorithms, achieving a time complexity of O(km log n), where m is the number of edges in the graph.

Approximations of the minimum Steiner tree based on the shortest path are also a research direction. The TM algorithm, proposed by Takahashi H et al. [11] in 1980, is an approximation algorithm that continuously finds the terminal node closest to the current tree node and adds the points on the path to the tree, repeating this process until all terminal nodes are connected. Therefore, it is also known as the nearest neighbor candidate priority algorithm. This algorithm achieves an approximation ratio of 2(1-1/k), with a time complexity of O(kn2). Initially proposed for the undirected minimum Steiner tree problem, the TM algorithm can also be applied to directed graphs in later research, but with a time complexity of O(n3). Stefan Voß [12] proposed the Prim-SPH algorithm, which sequentially finds the shortest paths from the source node to terminal nodes and finally merges these shortest paths into a Steiner tree. This algorithm generates only one tree during the solving process and is applicable to both undirected and directed Steiner tree problems. The Kruskal-SPH algorithm, proposed by Hwang F K [10], is only suitable for undirected Steiner tree problems. It generates multiple subtrees simultaneously and gradually connects each pair of subtrees with the lowest-cost path.

Additionally, the application of the Ant Colony Optimization (ACO) method to the minimum Steiner tree problem is also a popular research direction. Singh Voß et al. optimized the ACO for the minimum Steiner tree problem, mainly by adjusting the pheromone update mechanism and introducing heuristic information to improve the search efficiency and solution quality. Specifically, the algorithm optimizes the reinforcement and evaporation processes of pheromones, making ants more inclined to explore well-performing paths, while using cost information between nodes as a heuristic factor to guide ants in more intelligently selecting paths. These methods significantly enhance the algorithm's performance in dynamic sensor network environments, effectively reducing the total cost of data transmission.

3 | Description of the Network Design Problem Process

3.1 Graph Creation and Weighting

First, a weighted undirected graph is defined, consisting of a set of nodes and a set of edges. Generally, a given undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, defines $|V|$ and $|E|$ as the number of vertices and edges in the graph, respectively. If there is an edge e in E connecting vertices u and v, then u and v are said to be adjacent in the graph.For any vertex v in the graph, $d(v)$ represents the number of vertices adjacent to v, i.e., the degree of vertex v in the graph. For any two vertices u and v,the distance $d(u, v)$ is defined as the length of the shortest path between these two vertices in the graph.

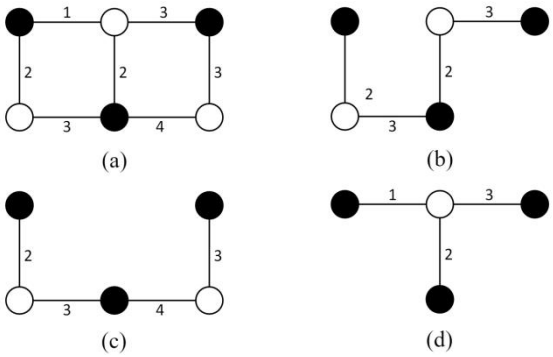

Figure 3: (a) is the input graph, where black points represent terminal vertices and white points represent Steiner vertices; (b)–(d) are three generated solutions with total weights of 10, 12, and 6, respectively. (d) represents the optimal solution.

In the diagram, each node represents a point in the network, and each edge represents a path connecting these points. The weight of the edges indicates the cost or distance of the path, reflecting the connection cost or physical length. The Steiner Tree Problem (STP) is defined as follows: given an undirected weighted graph $G = (V, E)$, the set of points V is divided into two groups: a set of terminal vertices $A \subseteq V$ and a set of Steiner vertices $V \setminus A$. In this context, the number of terminal vertices is denoted byt. Each edge $e \in E$ has an associated cost $c_e \geq 0$.The objective of the STP is to determine a subtree $T = (V(T), E(T))$ (with a vertex set $V(T)$ and an edge set $E(T)$) that covers all terminal vertices and minimizes the total cost $\sum_{e \in E(T)} c_G(e)$.

3.2 Solving the Terminal Set

In the Steiner Tree Problem (STP), solving for the terminal set is crucial because these terminal vertices are the essential set that the Steiner tree must cover. Firstly, it's important to clarify that the terminal set in the Steiner Tree Problem refers to a specific subset of vertices in a given undirected weighted graph that the Steiner tree must cover. These vertices usually have special significance or requirements, such as key nodes in a communication network or important locations in a logistics network.

Before the algorithm begins, it is necessary to initialize and define the terminal set A . Given an undirected weighted graph G, where V is the set of nodes containing all vertices in the graph, and E is the set of edges containing all edges and their weights, the specific vertices that are terminal vertices need to be identified based on the problem requirements or real-world application scenarios. These vertices will form the terminal vertex set A. For example, in communication network design, terminal vertices might be key communication nodes; in a logistics network, they might be distribution centers. It is essential to ensure that the terminal vertex set A is a subset of V , and to confirm that all terminal vertices exist in the graph.

In practical terms, solving for the terminal set may involve operations such as reading input data, marking terminal vertices, and selecting appropriate data structures. First, read the graph G and

the terminal vertex set A from an input file or data source, such as an adjacency matrix or an edge list. Then, mark all terminal vertices in the graph to quickly identify and process these vertices in subsequent algorithm steps. To improve lookup efficiency, a hash set can be used to store the terminal vertex set A.

3.2 Solving the Minimum Steiner Tree

3.2.1 | NetworkX.stenier_tree

Solve the Minimum Steiner Tree problem using the `steiner_tree` function from Python's NetworkX library. This algorithm is based on a greedy strategy, which iteratively selects edges to approximate the minimum weight subgraph that connects all terminal nodes.

To solve the Minimum Steiner Tree problem, follow these steps: First, define the nodes representing key locations and their connections in the graph, specifying the weights of these connections based on their cost or distance. Next, initialize the algorithm by selecting a starting node and constructing a tree to include all terminal nodes with the minimum total weight. This involves iteratively adding edges that connect the tree to the remaining terminal nodes while minimizing the cost. Finally, optimize the solution by evaluating and adjusting the tree to ensure it covers all terminal nodes while minimizing the overall weight.

This approximation method, while not guaranteeing a globally optimal solution, provides an efficient means to obtain an approximate minimum Steiner tree, making it suitable for large-scale network design problems. Finally, we evaluate and validate the generated Steiner tree to ensure that it meets the connectivity and cost-effectiveness requirements of the network design.

3.2.2 | Improved Ant Colony Optimization (IACO)

efficient means to obtain an approximate minimum Steiner tree, making it suitable for large-scale network design problems. Finally, we evaluate and validate the generated Steiner tree to ensure that it meets the connectivity and cost-effectiveness requirements of the network design.

This paper proposes an optimization strategy for the Ant Colony Optimization (ACO) algorithm, aiming to efficiently find a near-optimal solution for the minimum Steiner tree problem. Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behavior of ants in nature. The algorithm solves combinatorial optimization problems by simulating the pheromone trails left by ants as they search for food.

Initially, the graph G(V, E) is constructed, where V represents the set of nodes, and E represents the set of edges. The basic parameters of the ACO algorithm are then established, including the number of ants, the initial pheromone values, the pheromone evaporation coefficient, and the heuristic factor. Next, all nodes and edges in the graph are initialized as inputs to the algorithm, and an initial pheromone value is assigned to each edge. At this stage, the ants are randomly placed at the terminal nodes.

During the solution construction phase, each ant incrementally builds a Steiner tree by selecting paths. When choosing a path, the ant considers two pieces of information: the pheromone concentration on the path and the heuristic value of the path (usually the inverse of the distance). The probability of an ant selecting a particular path is determined by the following formula:

$$P(i,j) = -\frac{[\tau(i,j)]^{\alpha} \times [\eta(i,j)]^{\beta}}{\Sigma[\tau(i,k)]^{\alpha} \times [\eta(i,k)]^{\beta}}$$

Where $\tau(i,j)$ represents the pheromone concentration on the edge (i,j), $\eta(i,j) = \frac{1}{d(i,j)}$ is the heuristic information, and $d(i,j)$ is the distance between nodes i and j. The parameters α and β control the relative importance of the pheromone concentration and the heuristic information, respectively. After each path selection step, the ant performs a local pheromone update on the paths it has traveled. The local update rule simulates the influence of an individual ant's behavior on the overall pheromone concentration. The formula for the local update is as follows:

$$\tau(i,j) = (1 - \rho) * \tau(i,j) + \Delta\tau(i,j)$$

Where ϱ is the pheromone evaporation coefficient, and $\Delta\tau(i,j)$ is the pheromone increment, which is typically inversely proportional to the quality of the path.

Once all ants have constructed a Steiner tree that covers all terminal nodes, the algorithm moves to the solution completion phase. At this stage, the total weight of each ant's constructed Steiner tree is evaluated, and the best solution from the current iteration is selected. The steps above constitute a complete iteration. The algorithm will run multiple iterations until a termination condition is met. The termination condition can be a fixed maximum number of iterations or the absence of an improved solution over several consecutive iterations.

3.2.3 | Core Optimization Points

To further enhance the efficiency of the ant colony algorithm in solving the Minimum Steiner Tree problem, this paper proposes an Improved Ant Colony Optimization (IACO) algorithm. The algorithm incorporates several optimizations, making it more efficient in handling large-scale graphs and achieving better solutions.

First, IACO utilizes Dijkstra's algorithm to precompute the shortest paths between all pairs of nodes, transforming the original graph into a sparse matrix format. This significantly improves computational efficiency by allowing quick look-ups and connections between node pairs during path construction, without the need to build edge-by-edge.

Second, IACO leverages the precomputed shortest paths to rapidly construct connections during the path-building process. This not only speeds up the path construction but also reduces the overhead of redundant calculations. To further enhance computational efficiency, IACO uses NumPy arrays to store pheromones, utilizing matrix operations to significantly improve the efficiency of pheromone updates.

Finally, IACO implements a pruning strategy that removes redundant nodes after each path generation to avoid the formation of cycles. These optimization measures collectively reduce the computational complexity of IACO, with particularly notable performance improvements in large-scale graphs.

3.2.3 | Solving the Minimum Spanning Tree

In the optimization study of network design, to evaluate and compare the efficiency and cost-effectiveness of the Minimum Steiner Tree, we use the solution to the Minimum Spanning Tree (MST) problem as a benchmark. The MST results serve as a reference point for assessing the performance of the Minimum Steiner Tree. Using the `minimum_spanning_tree` function from the NetworkX library, we can compute the minimum weight tree that connects all nodes in the graph.

During the construction of the Minimum Spanning Tree, Prim's algorithm starts from an initial node and gradually expands to the entire graph. In each iteration, the algorithm selects the edge with the minimum weight that connects a visited node with an unvisited node and adds it to the spanning tree. This approach ensures that in each step, the tree expands by adding the edge with the least cost, thereby constructing the Minimum Spanning Tree step by step.

First, initialize the Minimum Spanning Tree by selecting a starting vertex and adding it to the tree. The starting vertex can be chosen randomly or based on a specific strategy, such as selecting the vertex with the highest degree. Initialize a priority queue to select the edge with the current minimum weight. This queue is initially empty.

Next, begin constructing the Minimum Spanning Tree. Add all edges connected to the starting vertex to the priority queue. Then, enter the iteration process: extract the edge with the minimum weight from the priority queue. If one of the edge's endpoints is already in the tree and the other is not, add this endpoint to the tree and add all edges connected to the new vertex to the priority queue. If both endpoints of the edge are already in the tree, discard the edge and continue to extract the edge with the minimum weight from the priority queue.

Repeat this process until the tree includes all vertices in the graph. Since only the edge with the current minimum weight that can extend the tree is selected each time, the resulting tree is the Minimum Spanning Tree with the smallest total weight.

This process not only provides a global view of the entire network but also serves as an ideal benchmark for evaluating Steiner tree solutions that connect only specific terminal nodes.

3 | Empirical Analysis

Kanazawa, located in Ishikawa Prefecture, Japan, is an important city whose central area includes several key zones and facilities such as Kanazawa Station, Kanazawa City Hall, Ishikawa Sports

Center, Kanazawa Castle Park, and Kenrokuen Garden. The central area of Kanazawa covers 916 hectares and includes several major transportation hubs and commercial districts. The landmarks and transportation routes marked in the map highlight the complexity and significance of this region, connecting multiple train stations (including Kanazawa Station, Higashi-Kanazawa Station, and Nishi-Kanazawa Station), national highways, and subway lines. To enhance network connectivity and communication efficiency in this area, an efficient network structure needs to be established to effectively link these key nodes.

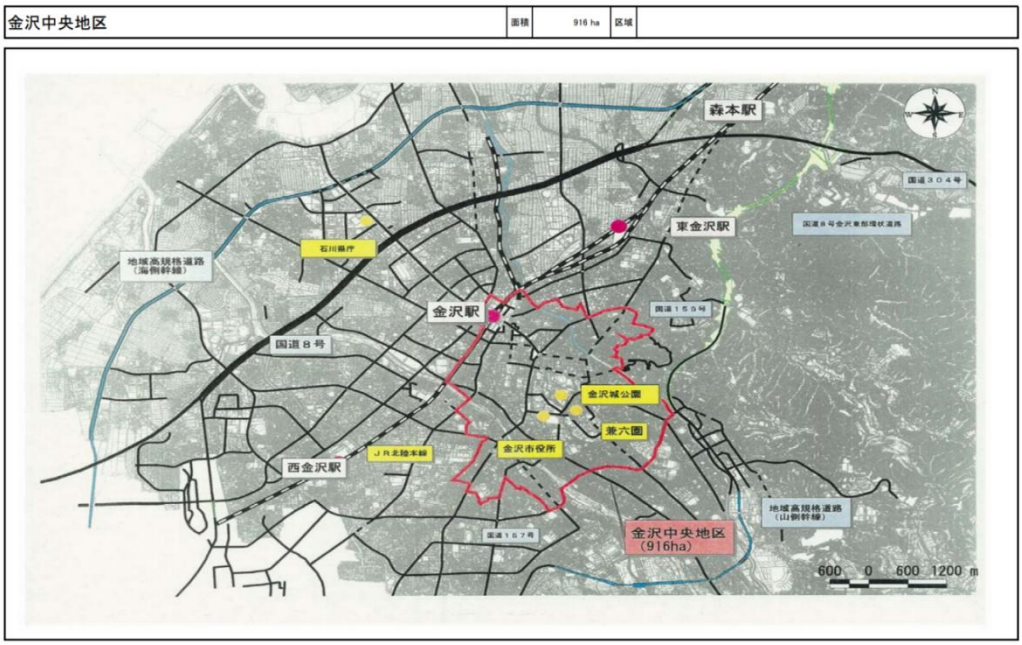

Figure 4: Landmark and Transportation Route Map of Kanazawa City, Ishikawa Prefecture

In this context, we use the Minimum Steiner Tree algorithm to construct an efficient network covering the central area of Kanazawa. The goal of the Minimum Steiner Tree problem is to find a subtree that covers all terminal vertices with the minimum sum of edge weights. We use important landmarks and transportation hubs in the central area of Kanazawa as terminal vertices and optimize the connections between these nodes using the Steiner Tree algorithm.

Step 1. Node Definition

Define important landmarks and transportation hubs on the map as nodes. These nodes include Kanazawa Station, Higashi-Kanazawa Station, Nishi-Kanazawa Station, Kanazawa City Hall, Ishikawa Sports Center, Kanazawa Castle Park, and Kenrokuen Garden. To provide a more intuitive representation, we selected a representative graph from SteinLib [14] Testset B, which includes graphs with different numbers of nodes and edges, to test the algorithm's performance on various scales. The following figure illustrates the relationships between the important landmarks and transportation hubs in Kanazawa City, Ishikawa Prefecture.

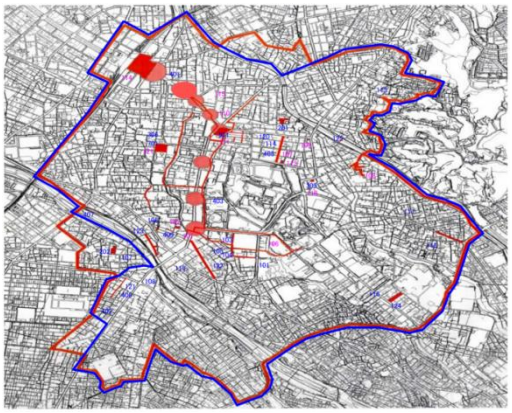

Figure 5: Important Landmarks and Transportation Hubs in Kanazawa City, Ishikawa Prefecture

Step 2. Edge Weight Definition

Next, we define the paths connecting these nodes as edges. The weight of each edge represents the cost or distance of the path, reflecting the connection cost or physical length. The weights of the edges can be set based on the actual geographical or transportation distances on the map. A smaller edge weight indicates a lower cost or distance for that path. In this step, we need to measure and record the weight of each edge in detail to ensure the accuracy and effectiveness of the subsequent algorithms.

Step 3. Initialization

During the initialization phase, we select a starting node and include it in the Steiner tree. Suppose we choose Kanazawa Station as the starting node, as it is a major transportation hub connecting several key areas. To better visualize the nodes, we use Python's NetworkX library and Matplotlib for plotting, employing a configurable layout (defaulting to a circular layout).

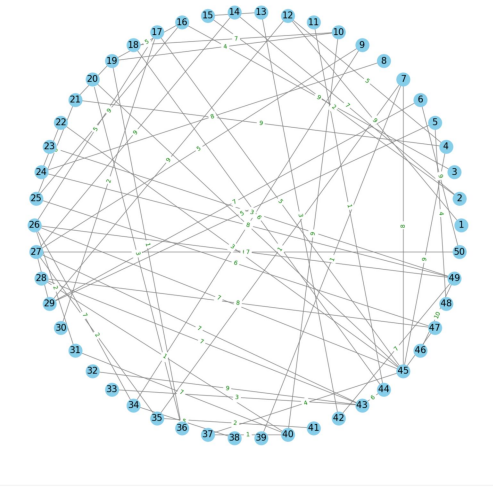

Figure 6: Original Graph Showing All Nodes, Edges, and Edge Weights

Next, we define the paths connecting these nodes as edges. The weight of each edge represents the cost or

distance of the path, reflecting the connection cost or physical length. The weights of the edges can be set based on the actual geographical or transportation distances on the map. A smaller edge weight indicates a lower cost or distance for that path. In this step, we need to measure and record the weight of each edge in detail to ensure the accuracy and effectiveness of the subsequent algorithms.

Step 4. Iterative Construction of the Steiner Tree

1. NetworkX.steiner_tree

During the iterative construction of the Steiner tree, we select the edge with the minimum cost from the priority queue. If one of the nodes connected by this edge is not yet included in the Steiner tree, we add that node to the Steiner tree and include the edge in the Steiner tree. Then, we add all edges connected to the newly added node to the priority queue. This process is repeated until all terminal nodes are included in the Steiner tree. In each iteration, we ensure that the selected edge is the one with the current minimum weight that can extend the Steiner tree, gradually constructing the optimal connecting network.

2. IACO

Given the simplicity of the graph, the specific parameters for the algorithm are set as follows: 1 ant, 1 iteration, pheromone decay rate of 0.1, pheromone importance α of 0.5, heuristic information importance β of 2, and pheromone increment $\Delta\tau(i,j)$ of 100. At the start of the algorithm, a terminal node is randomly selected as the starting point. The ant constructs the solution incrementally based on pheromone concentration and the distances between nodes. By enhancing the path pheromones and optimizing the heuristic information, the algorithm iteratively generates an approximate optimal path. Finally, the pruning process removes unnecessary leaf nodes, resulting in the final approximate solution to the minimum Steiner tree.

Step 5. Output the Steiner Tree

Finally, after completing all iterations, we obtain a Steiner tree that covers all terminal nodes with the minimum total edge weight. This tree represents the efficient network we have constructed, connecting all key nodes in the central area of Kanazawa. The optimized network structure significantly improves communication and transportation efficiency within the region, facilitating the development of Kanazawa City.

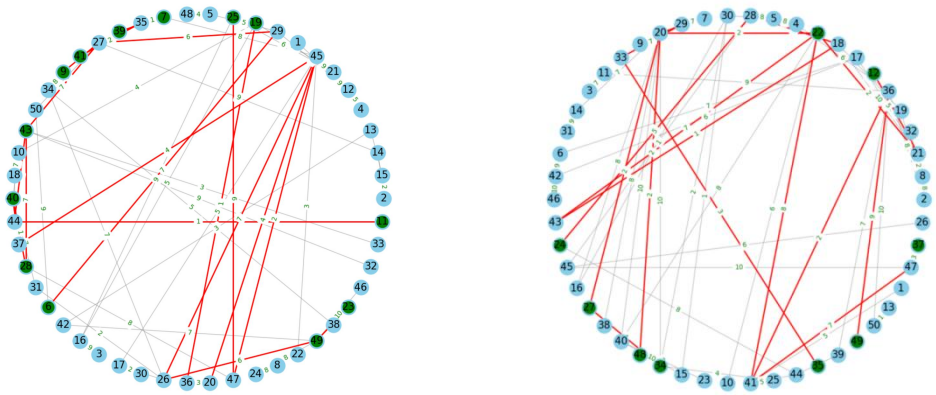

Figure 7: Minimum Steiner Tree Results

IACO - Total Cost 83 (left) and **NetworkX.steiner_tree - Total Cost 100 (right)**

The above figure displays a subgraph that connects only terminal nodes with the minimum total weight, where terminal nodes are highlighted in green, edges of the Steiner tree are shown in red, and edges not included in the Steiner tree are displayed in gray.

Step 6. Calculating the Minimum Spanning Tree

To compare the performance of the Minimum Steiner Tree with the Minimum Spanning Tree (MST), we need to compute the MST. Using Prim's algorithm, the MST connects all nodes while minimizing the total weight without considering specific terminal nodes.

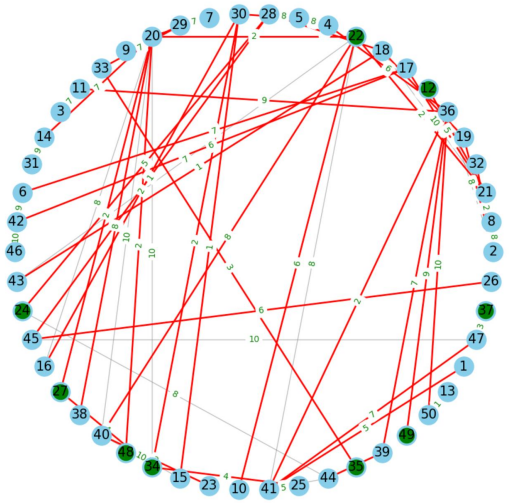

Figure 8: Minimum Spanning Tree Result - Total Cost 238

The above figure displays a tree that connects all nodes with the minimum total weight, where terminal nodes are highlighted in green, edges of the minimum spanning tree are shown in red, and edges not included in the tree are displayed in gray.

Step 7. Comparing Results

After constructing the minimum Steiner tree and the minimum spanning tree, we compare the two results. The original graph, minimum Steiner tree, and minimum spanning tree are displayed side by side to allow users to visually compare their structures and differences. These visual results provide a clearer understanding of the optimization effect of the minimum Steiner tree for connecting specified terminal nodes and allow comparison with the minimum spanning tree to observe the differences between the overall graph structure and specific optimized structures. By comparing the total weights of the two trees, we can assess the optimization effect of the Steiner tree for specific terminal node requirements. The minimum Steiner tree covers the specified terminal nodes with a smaller total weight, while the minimum

spanning tree connects all nodes, potentially including unnecessary paths.

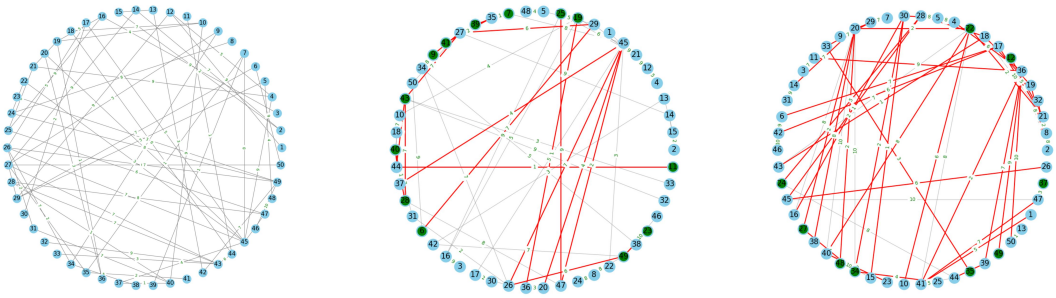

Figure 9: Comparison of the Original Graph (Left), IACO Minimum Steiner Tree (Center), and Minimum Spanning Tree (Right)

Step 8. Efficiency Analysis

This section will evaluate the performance of the improved Ant Colony Optimization (IACO) algorithm and the NetworkX library method in solving the Minimum Steiner Tree problem through comparative experiments. The dataset used in the experiments is based on SteinLib's Testset B, which is widely used to assess the performance of Minimum Steiner Tree algorithms on medium-sized graphs. All algorithms were run in the same hardware and software environment to ensure fairness and consistency of the experimental results. The experimental results include a performance comparison of each method across all selected graphs, with the primary evaluation metrics being:

1. Approximation Ratio = Total Cost of the Algorithm / Known Total Cost

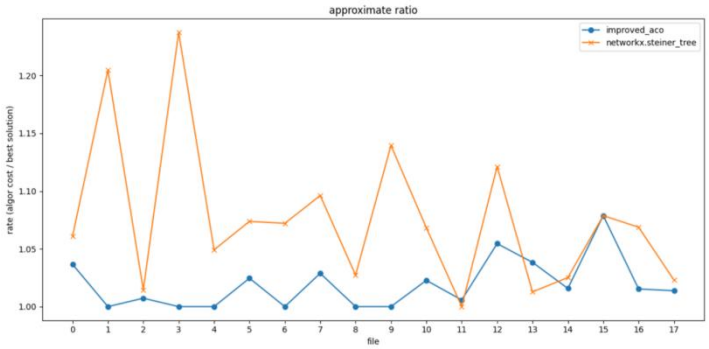

Figure 10: Comparison of Approximation Ratios between IACO and NetworkX.steiner_tree

This figure illustrates the performance comparison between the Improved Ant Colony Optimization (IACO, blue curve) and the Steiner Tree algorithm in the NetworkX library (yellow curve) across different datasets. The X-axis represents dataset numbers, while the Y-axis shows the ratio of the algorithm's cost to the best-known solution (approximation ratio). The graph indicates that the Improved Ant Colony Optimization algorithm provides results significantly closer to the optimal solution across most datasets, demonstrating a lower approximation ratio and thus higher effectiveness on these instances. In contrast, the NetworkX Steiner Tree algorithm exhibits higher approximation ratios in several instances, particularly at data points 1

and 3, showing noticeably poorer accuracy. This suggests that the Improved Ant Colony Optimization algorithm delivers better solution quality for most standard Steiner Tree problems.

2. Runtime: Record the computation time for each method.

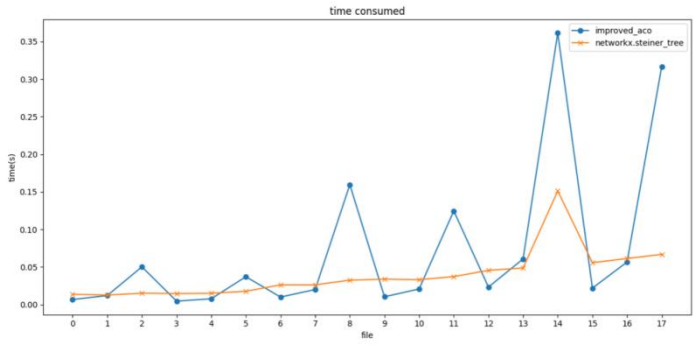

Figure 11: Comparison of Execution Time for IACO and NetworkX.steiner_tree

This figure compares the execution time of the Improved Ant Colony Optimization algorithm (IACO, blue curve) and the Steiner Tree algorithm from the NetworkX library (yellow curve) across different datasets. The X-axis represents the dataset numbers, and the Y-axis shows the time (in seconds) required for each algorithm to solve the problem.

The figure illustrates that while IACO shows higher time consumption on certain datasets (particularly data points 8, 14, and 17), it generally finds solutions more quickly across most datasets, demonstrating better time efficiency. In contrast, the NetworkX Steiner Tree algorithm exhibits relatively stable time consumption, providing a valuable benchmark for potential future improvements or hybrid algorithm research. This suggests that although IACO may face performance bottlenecks when handling specific problem types, it typically achieves faster results across a broad range of samples, highlighting its applicability and efficiency in diverse datasets. Meanwhile, the stability of the NetworkX method also offers potential for developing hybrid algorithms to leverage the strengths of each approach, further optimizing solutions.

4 | Conclusion

This study aims to optimize the network design of the central area of Kanazawa City using the Minimum Steiner Tree (STP) algorithm to enhance communication and traffic efficiency in the region. After a detailed analysis of the classical STP problem, relevant literature, and algorithm implementations, we proposed an Improved Ant Colony Optimization (IACO) algorithm and compared it with Python's NetworkX library in practical applications. The results indicate that the Improved Ant Colony Optimization algorithm not only matches the time efficiency of the NetworkX approach in most standard Steiner Tree problems but also produces superior solutions.

Additionally, through empirical analysis, we found that the Minimum Steiner Tree offers advantages over the traditional Minimum Spanning Tree (MST) when connecting important landmarks and transportation hubs. It reduces unnecessary connections, improves resource utilization, and enhances network robustness. In contrast, while the MST covers all nodes, it includes many redundant paths, increasing the total weight and complexity.

Future research could further refine the algorithms by exploring combinations of the IACO with other intelligent optimization or machine learning algorithms. This would potentially improve computational efficiency and solution quality in large-scale network designs, and investigate its application potential in even larger and more complex networks.

References

[1] Pro.ceedings ofthe thirty-ninth annual ACM symposium on Theory of computing, 2007,67-74

[2] Byrka J, Grandoni F, RothvoB T, et al Steiner tree approximation via iterative randomized round.ing[J]. Journal ofthe ACM, 2013,60(1):1-33

[3] Chlebik M, Chlebikov¡ J. The steiner tree problem on graphs: Inapproximability results[J]. Theoretical Computer Science,2008,406(3):207-214

[4] Abbasi A A,Younis M,Akkaya K.Movement-assisted connectivity restoration in wireless sensor and actor networksUl.lEEE Transactions on Parallel & Distributed Systems,2009,20(9):1366-1379.

[5] Senel fYounis M,Akkaya K.A robust relay node placement heuristic for structurally damaged wireless sensor networks[C]//Proc of lEEE 34th Conference on Local Computer Networks,2009:633-640.

[6] Senel f,Younis M F,Akkaya k.Bio-inspired relay node placement heuristics for repairing damaged wireless sensor networksU].EEE Transactions on Vehicular Technology,2011,60(4):1835-1848.

[7] Lee S,Younis M.Recovery from multiple simultaneous fai- lures in wireless sensor networks using minimumSteiner tree!].Journal of Parallel and Distributed Computing,2010,70(5):525-536.

[8] Senel F,Younis M.Relay node placement in structurally damaged wireless sensor networks via triangular Steiner tree approximation!].Computer Communications,2011,34(16):1932-1941.

[9] Kou L, Markowsky G, Berman L. A fast algorithm for Steiner trees[J]. ActaInformatica, 1981, 15(2):141-145.

[10] Aragão M P D, Werneck R F. On the Implementation of MST-Based Heuristicsfor the Steiner Problem in Graphs[J]. Lecture Notes in Computer Science, 2002,2409:1-15.

[11] Takahashi H. An approximate solution for the Steiner problem in graphs[J].Math. Japonica, 1990, 6: 573-577.

[12] Stefan Voß. Steiner's Problem in Graphs: Heuristic Methods.[J]. DiscreteApplied Mathematics, 1992, 40(92):45-72.

[13] Hwang F K, Richards D S. Steiner tree problems[J]. Algorithmica, 1992,7(1-6):329-332.

[14] Koch, Thorsten. "SteinLib Testdata Collection." Steinlib.zib.de, 2015, steinlib.zib.de/steinlib.php. Accessed 6 Aug. 2024.