A WEB APPLICATION FOR MST COMPUTATION IN RANDOM AND USER-DEFINED GRAPHS

K. Troumpounis[1], E. Papaioannou[2], N. Karanikolas[1], C. Kaklamanis[2]

[1]University of Patras (GREECE)
[2]University of Patras and CTI "Diophantus" (GREECE)

Abstract

In Graph Theory, a graph is defined as a collection of vertices and edges, where vertices correspond to entities and an edge between two vertices implies that the corresponding entities are somehow related. Edges can have weights reflecting a kind of cost required for their establishment, such as distance, time or budget. Trees are special cases of graphs which are acyclic and connected. A Spanning Tree for a given graph is a tree containing all vertices of this graph. When weights are assigned to the edges of a graph, a Minimum Spanning Tree (MST) for this graph is a spanning tree of minimum total edge-cost. Given a connected graph, we can efficiently (i.e., fast) compute a Minimum Spanning Tree for this graph using two well-known algorithms from the literature, one suggested initially by Vojtěch Jarník and later by Robert Prim and one suggested by Joseph Kruskal, which return similar results though in a slightly different fashion. The MST problem has various practical applications in real-world instances where we seek for minimum-cost solutions for connecting all involved points of interest.

In this work, we present a web application for the computation of Minimum Spanning Trees (MST) for user-defined graphs which can be either randomly generated or manually drawn on a canvas. Our main intention with this application is to provide an additional tool for students to support their study mainly within courses on Introduction to Algorithms but also in the context of courses utilizing the MST construct. Of course, our MST calculator can be exploited by all interested users when their activities involve MST computations.

Our application contains the main section and a training section. In the main section, users can set-up an instance of the MST problem by first providing the initial graph and then selecting the desired method, i.e., Kruskal's or Prim's algorithm, for the MST computation. The initial graph can be either a user-defined random graph or it can be manually drawn on a canvas. Users can choose to either receive the computed MST or observe the whole computational procedure in a step-by-step fashion. Users can download computed MSTs as .png files. In the training section, users can practice on the application of Kruskal's and Prim's algorithms for the MST problem. In particular, users are presented with a graph automatically generated by the application and the method they are expected to apply. Then users must gradually build an MST by selecting edges until all vertices of the given graph are connected. The computed MST is evaluated by the application and in case of incorrect answers corrections are indicated directly on the user-computed MST.

Compared to other existing relevant applications, the innovative nature of our application mainly stems from its educational focus. More precisely, unlike existing approaches, our application allows users to familiarize with random graphs, a model capturing real-world scenaria, and also enables them to directly draw their desired initial graph. Also, our application enables users to exploit both algorithms in a single interface and run them also in a step-by-step fashion thus letting them experimentally and even visually compare how these algorithms build MSTs. Our application is responsive, currently offered in greek and english and has been developed using React.

Keywords: Educational web app, responsive, Minimum Spanning Tree (MST) problem, Kruskal's algorithm, Prim's algorithm, random graphs, React.

1 INTRODUCTION

Graph theory provides a fundamental framework for modelling and analysing connectivity in diverse real-world scenarios, with the minimum spanning tree (MST) problem standing as a cornerstone concept for network optimization. The ability to find the minimum-cost set of edges connecting all vertices in a weighted, undirected graph is crucial. That is why we believe that mastering the underlying principles and algorithms, primarily those developed by Prim and Kruskal, constitutes a key learning objective in computer science education, offering valuable insights into algorithm design and analysis.

However, our observation within educational settings indicates a persistent challenge: while textbooks and traditional lectures effectively convey the theoretical foundations of MSTs, they often fall short in providing the interactive experiences necessary for students to fully internalize the dynamic, step-by-step nature of these algorithms. A visualized learning and training environment could support students to grasp the decision-making process inherent in Kruskal's edge-sorting approach or Prim's vertex-centric expansion, particularly when applied to various graph structures beyond simple examples. This gap between theoretical knowledge and practical, visual understanding motivates the design and development of innovative digital learning and training tools.

The objective of this work is to bridge this gap by introducing MST Calculator, an interactive, web-based educational application specifically designed to support student learning of MST algorithms. Built using react to leverage its capabilities for creating dynamic and responsive user interfaces, our application aims to provide a flexible and accessible platform for exploration and practice.

While various online tools exist for graph algorithm computation or visualization (e.g. Vercel MST-calculator [20] or VisuAlgo min spanning tree [19], GraphOnline's MST tool [14], or the Technical University of Munich's algorithm visualization for Kruskal's algorithm [17]), many focus primarily on computation output or lack features crucial for targeted learning. Existing applications often neglect dedicated pedagogical components, such as integrated training modules, or fail to offer a simple easy to use option for students in order to grasp the information as easy as possible.

In contrast, our MST Calculator integrates several key features within a single platform specifically designed for learning: it allows graph input via both random generation and manual canvas drawing, provides step-by-step visualizations for both Kruskal's and Prim's algorithms, and includes a Training Mode with immediate user feedback. Furthermore, it offers a responsive bilingual (Greek/English) interface.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the MST problem, the algorithms suggested by Kruskal and Prim and the concept of Erdős–Rényi random graphs. In Section 3, we present the technical framework of the MST Calculator, covering its design aspects, its development, and a short demonstration. We conclude in Section 4 also discussing plans for future enhancements.

2 THEORETICAL FRAMEWORK

2.1 The MST problem

The minimum spanning tree (MST) problem is a fundamental optimization problem within graph theory [5], addressing the challenge of connecting a set of points in the most economical way possible. Formally, consider an undirected, connected graph G=(V,E), where V is the set of vertices (nodes or points) and E is the set of edges (connections between pairs of vertices). Each edge e in E is assigned a non-negative weight w(e), representing a cost, distance, time, or other relevant metrics associated with that connection [5]. Within such a graph, we seek a subset of the edges that connects all the vertices together without forming any cycles. More formally, a subgraph of the original graph is called a spanning tree if it is connected, contains all vertices of the original graph and is acyclic (contains no cycles).

The minimum spanning tree problem is not merely a theoretical construct; its solution provides a powerful tool for optimizing connectivity in numerous practical domains. Finding a structure that connects all points with minimum total cost has direct relevance across various fields. For instance, in network design, minimum spanning trees help determine the least-cost layouts for physical networks like telecommunication lines, computer networks, utility pipelines, and transportation routes [4,7]. Similarly, in electronic circuit design, minimum spanning trees aid in minimizing the total wire length needed to connect elements on circuit boards or within integrated circuits [1]. Furthermore, constructing a minimum spanning tree is often a key step in approximation algorithms for computationally harder problems, such as the traveling salesperson problem [5]. Other applications arise in areas like computational biology, for example in building phylogenetic trees, and in image analysis for understanding network-like structures [5,7].

2.2 Kruskal's algorithm

Kruskal's algorithm is a classic greedy algorithm designed to find a minimum spanning tree for a connected, weighted, undirected graph. It was first published by Joseph Kruskal in 1956 [5] and remains one of the primary methods taught and used for solving the MST problem. The algorithm receives as input a connected, undirected graph where each edge has an associated numerical weight [5]. If the input graph is not connected, the algorithm will produce a minimum spanning forest (a collection of minimum spanning trees, one for each connected component). The primary output, assuming a connected input graph, is a set of edges that constitutes a single minimum spanning tree for that graph.

Kruskal's algorithm builds a minimum spanning tree using a greedy approach, always prioritizing the edge with the lowest weight that doesn't violate the tree structure. An indicative execution of Kruskal's algorithm on a random graph is illustrated in Fig. 1. The process can be described as follows:

1. Initialize an empty set, which will store the edges forming the minimum spanning tree.
2. Sort all the edges (E) available in the input graph according to their weight, typically from smallest to largest.
3. Iterate through the sorted edges, starting with the edge having the lowest weight.
4. For each edge under consideration, check if adding it to the set of edges already selected would create a cycle.
5. If adding the current edge does not form a cycle, add it to the set of selected edges for the minimum spanning tree. If adding the edge would form a cycle, ignore this edge and move to the next one in the sorted list.
6. Continue this process of checking and adding edges until exactly V-1 edges have been selected, where V is the number of vertices in the graph.
7. The final set containing V-1 edges produced by this procedure constitutes a minimum spanning tree for the original connected graph [3].

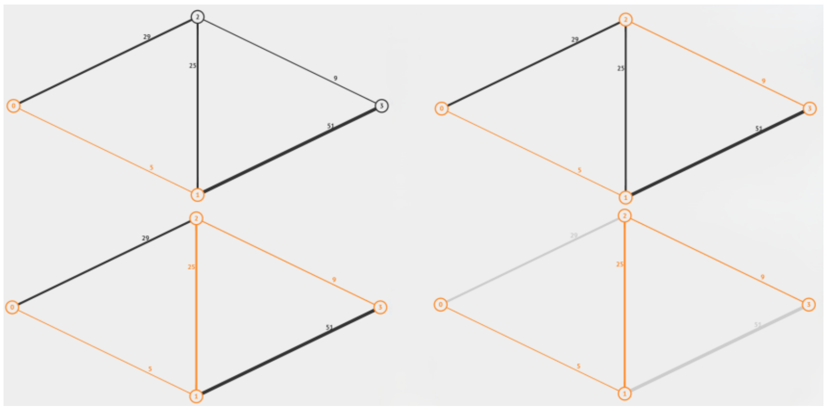


Figure 1: Step-by-step execution of Kruskal's algorithm on a random graph (figure created using Visualgo tool [20]).

The efficiency of Kruskal's algorithm is typically measured by its time complexity, which depends on the number of vertices (V) and edges (E) in the input graph. The algorithm's runtime is largely determined by the initial step where all E edges must be sorted according to their weight. Using efficient comparison-based sorting algorithms, this sorting process requires O(E log E) time [6,8]. The subsequent phase involves iterating through the sorted edges and employing data structures to efficiently track vertex components and prevent the formation of cycles when adding edges to the minimum spanning tree being constructed. While crucial for correctness, the time taken for these component-checking and merging operations, when implemented effectively, is generally less than the time required for the initial edge sort [3]. As a result, the overall performance is typically dominated by the sorting step. Therefore, the standard accepted time complexity for Kruskal's algorithm is O(E log E) [6,2]. For many graphs encountered in practice, this complexity is also equivalent to O(E log V) [3].

2.3 Prim's algorithm

Prim's algorithm provides another effective greedy strategy for determining a minimum spanning tree. While commonly associated with Robert Prim who published it in 1957, it was originally developed earlier by Vojtěch Jarník in 1930 [9]. Like Kruskal's algorithm, it guarantees finding an optimal solution for connected, weighted, undirected graphs. The algorithm takes as input a connected, undirected graph with assigned edge weights [9]. It is also common to specify a starting vertex from which the tree construction will begin, although the choice of start vertex does not affect the final minimum spanning tree's total weight. The output is the set of edges forming a minimum spanning tree for the graph.

Prim's algorithm constructs the minimum spanning tree by progressively growing a single connected tree component. It starts from an initial vertex and, at each step, adds the cheapest possible edge that connects a vertex already in the tree to one that is not yet included. An indicative execution of Prim's algorithm on a random graph is illustrated in Fig. 2. The steps of the algorithm are as follows:

1. Initialize an empty set to store the edges of the minimum spanning tree. Choose an arbitrary vertex in the graph as the starting point of the tree.
2. Maintain information about the minimum-weight edge that connects each vertex outside the current tree to any vertex currently inside the tree. Initially, this information is based on the edges connected to the starting vertex.
3. Repeat the following steps until the tree includes all V vertices in the graph: a. Among all edges that connect a vertex currently inside the tree to a vertex outside the tree, identify the edge with the smallest weight. b. Add this minimum-weight edge to the set of selected edges for the minimum spanning tree. c. Include the vertex reached by this new edge into the set of vertices considered inside the tree. d. Update the connection information for the neighbors of the newly added vertex, as there might now be cheaper ways to connect other outside vertices to the growing tree through this new vertex.
4. Once all V vertices are part of the tree structure, the process terminates. The collected set of V-1 edges forms a minimum spanning tree [3].

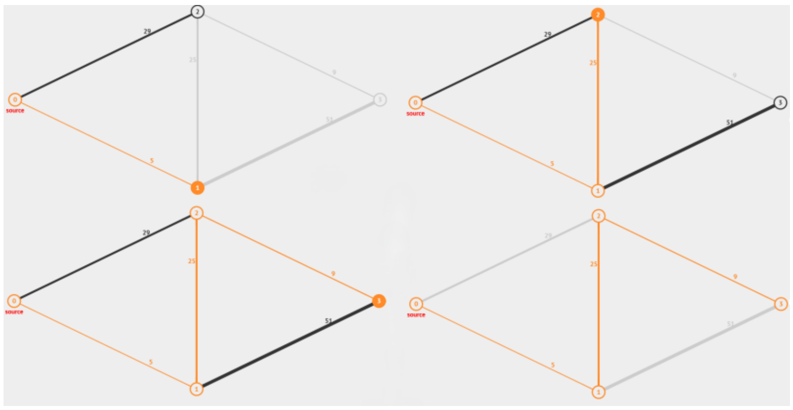


Figure 2: Step-by-step execution of Prim's algorithm on a random graph (figure created using Visualgo tool [20]).

The time complexity of Prim's algorithm depends on the data structures used. A simple implementation can run in O(V^2) time [3], where V is the number of vertices. However, by using more efficient data structures like a priority queue, the time complexity can be reduced to O(E log V) where E is the number of edges. This makes Prim's algorithm efficient for sparse graphs, where the number of edges is relatively small compared to the number of vertices [3].

2.4 Erdős–Rényi random graphs

The Erdős–Rényi model, named after Paul Erdős and Alfréd Rényi, provides a foundational framework for generating graphs based on probabilistic rules [4]. While there are two closely related variants, the commonly used variant relevant to our application considers a graph with a specific number of vertices, where an edge is created between each distinct pair independently according to a specified probability value [4]. Studying such random graphs holds significant importance in graph theory. They serve not only as crucial baseline models for comparing the structure of real-world networks but also for understanding fundamental network phenomena like phase transitions, where properties such as large-scale connectivity emerge abruptly as the edge probability changes. Furthermore, they provide essential test beds for analysing the average-case performance of graph algorithms, offering insights beyond traditional worst-case scenarios, and act as a simple, mathematically tractable starting point for developing more complex network models [8].

In our MST Calculator, this edge probability model drives the random graph feature. Users specify the desired number of vertices and the probability factor for edge inclusion. Based on these inputs, the application constructs the graph's structure by randomly determining the existence of each potential edge. Following the structure generation, random weights, selected uniformly from a user-defined range, are assigned to the created edges. This process enables users to generate varied graph scenaria for practicing minimum spanning tree computations.

3 TECHNICAL FRAMEWORK

This section provides the design and implementation details of the MST Calculator web application, which is publicly accessible at https://konstantinostroumpounis.github.io/calculate-mst/. We will describe the application's design, focusing on the user interface, core functionalities, and responsiveness, followed by an overview of the development process, including the technologies used and specific implementation aspects.

3.1 Design

3.1.1 User Interface

The MST Calculator application is structured with a simple and clear layout that organizes the available options in a logical manner. At the top of the page, a horizontal navigation bar is present, containing the application's logo on the left side, a "How to use" button, a "Training" button, and language selection options for English and Greek. Clicking the "Training" button navigates users to a dedicated module where random graph challenges are presented. This section maintains a similar visual style to the main interface but restricts controls to the interactive MST-building and feedback mechanism. Underneath the navigation bar, the central part of the screen is occupied by a large canvas area, which serves as the main visual workspace. To the left side of the canvas, there is a vertical toolbar offering tools for building, generating, and clearing the graph as well as the options to run the algorithms. This vertical toolbar remains consistently visible and separated from the main canvas area. The design employs a minimal visual style, featuring a clean background and a clear separation between the navigation bar, the canvas, and the toolbar. The arrangement ensures that all key elements are immediately visible and accessible without overlapping or cluttering the workspace. The overall visual organization emphasizes clarity and straightforwardness, allowing users to easily understand the structure and immediately begin experimenting with the application without needing extensive familiarization with the UI.

3.1.2 Functionalities

The MST Calculator application provides core functionalities designed for interactive learning and exploration of minimum spanning tree algorithms. Users have the option to either generate a random graph [4] automatically or manually construct a custom graph by placing vertices and connecting them with weighted edges on the canvas. Once a graph is present, users can select and execute either Kruskal's or Prim's algorithm to compute the corresponding minimum spanning tree. The application allows the minimum spanning tree computation to be presented either as a final result showing the highlighted tree, or through a step-by-step visualization process, enabling users to observe the progressive construction of the tree according to the chosen algorithm's logic. In addition to displaying the minimum spanning tree visually, the application also calculates and clearly displays the total weight (sum of edge weights) of the computed MST. After an algorithm execution, users have the option to download an image file (.png format) of the resulting graph with the highlighted MST for further study. A dedicated button allows users to clear the visual effects of the algorithm's execution (such as highlighted edges or steps), restoring the display to the original graph structure. This enables users to easily run a different algorithm on the same graph or make modifications before further computations. Furthermore, the application includes a training functionality which automatically generates graph problems and challenges users to build the minimum spanning tree interactively by selecting edges. Immediate feedback is provided during this process, indicating whether the user's choices are correct according to the selected algorithm's steps, thereby reinforcing conceptual understanding through active engagement.

3.1.3 Responsiveness

Responsive web design is a crucial aspect of modern web development, ensuring that an application's interface automatically adapts to the user's specific viewing environment [6]. For our MST Calculator, this means the layout and its elements are designed to fluidly change based on the screen size, resolution, and orientation of the device being used. Whether accessed on a large desktop monitor, a laptop, a tablet, or a smartphone, the interface adjusts by resizing content, modifying layouts, or selectively showing/hiding elements to optimize readability and interaction [7]. The fundamental goal of this responsive approach is to provide all users with a consistent, accessible, and user-friendly experience, regardless of the device they choose to use. Our MST Calculator on various devices is depicted in Fig. 3.

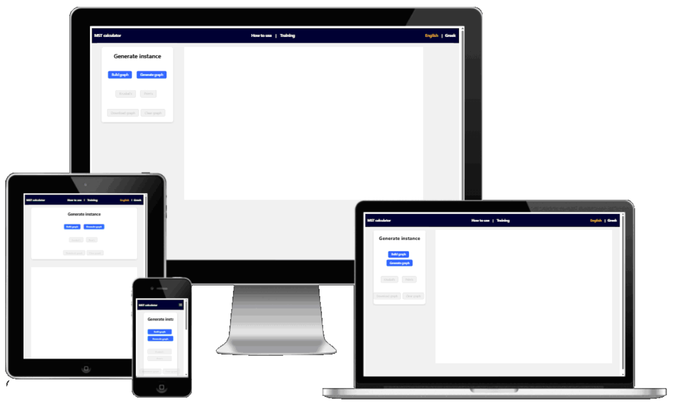


Figure 3: Our MST Calculator on various devices (via amiresponsive tool [18]).

3.2 Development

3.2.1 Software and technologies

The development of the MST Calculator leveraged several modern web technologies to deliver an educational, interactive, and responsive learning platform for learning algorithms commonly applied in graph theory problems. React.js [16] served as the core framework for building the user interface, offering component-based architecture that facilitated modularity, reusability, and efficient state management. In the MST Calculator, React.js was used to structure the entire application interface, including graph visualization controls, training modules, language selection, and feedback notifications. Vite [12] was employed as the development server and build tool. Known for its speed and optimized module bundling, Vite significantly improved the development experience, especially by providing fast refreshing iterative updates of UI components and algorithm animations.

To enable dynamic graph creation and manipulation, Cytoscape.js [11] was integrated as the primary graph visualization library. Cytoscape.js handled the rendering of vertices and edges, the live execution and visualization of MST algorithms (Kruskal’s and Prim’s), and supported the export of completed MSTs as image files for educational documentation. Ant Design [10] was utilized to create a responsive user interface. Ant Design components such as buttons, modals, inputs, sliders, and notification pop-ups were incorporated throughout the application, including the step-by-step visualization panels and the Training Mode interaction dialogs.

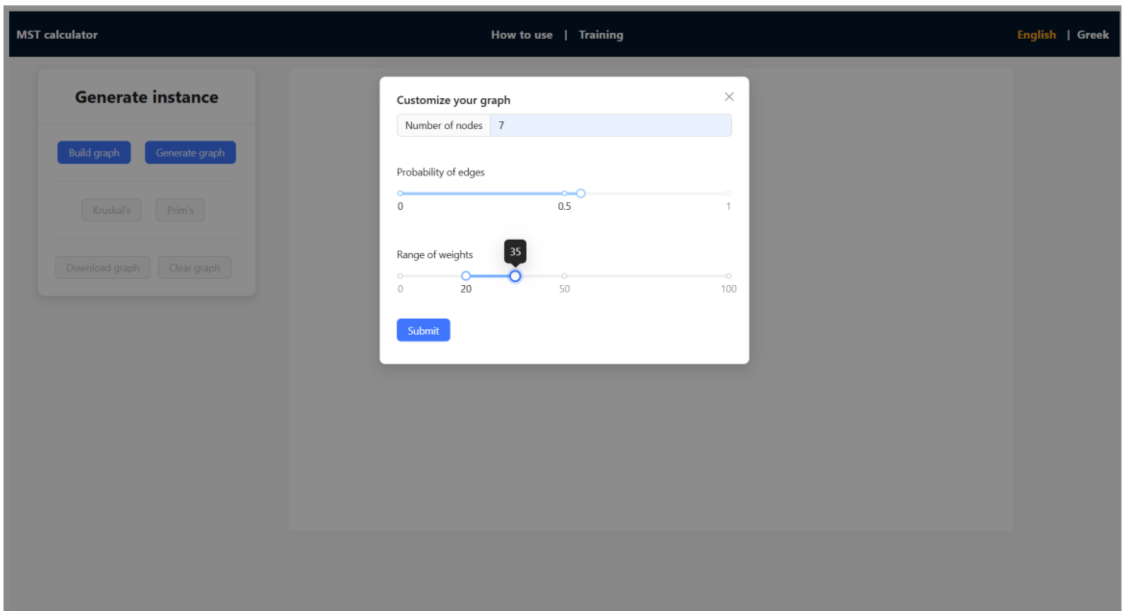
The application also supports multilingual functionality through i18next and react-i18next [15]. These libraries enabled the dynamic switching of the user interface between English and Greek languages. Language selection persists across sessions by utilizing the browser's local storage. Finally, GitHub pages [13] were used to host and deploy the MST Calculator, making the application freely accessible to users without the need for backend infrastructure.

Each of these technologies contributed to enhancing the educational value, ease-of-use, and accessibility of the MST Calculator, aligning the project with the goals of delivering an intuitive and easy learning experience in graph theory and algorithms.

3.2.2 Demonstration

In this section we demonstrate the core features and workflow of the MST Calculator application using a sample scenario. While the application allows users to manually construct graphs from scratch, for this demonstration we will utilize the random graph generation feature. We will showcase the process of generating a random graph instance based on user parameters, followed by the execution of a minimum spanning tree algorithm with step-by-step visualization, and conclude by exploring the application's interactive Training Mode.

We start by specifying the necessary parameters for the generation of a random graph (Fig. 4). In particular, we want the application to generate for us a graph with 7 vertices, edge probability 0.55, and edge-weight ranging from 20 to 35.



Customize your graph

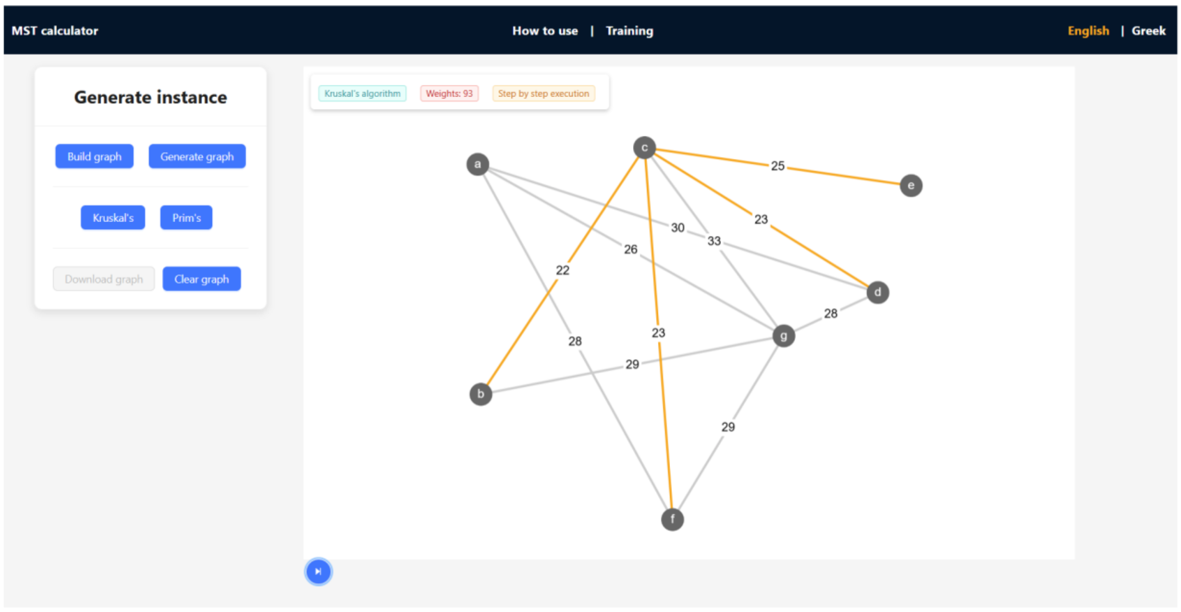
Number of nodes 7

Probability of edges 0 0.5 1

Range of weights 35 0 20 50 100

Figure 4: Specifying parameters for random graph generation within the MST Calculator.

Once the random graph is generated and displayed in the canvas, the user can now proceed to compute a minimum spanning tree for this graph. Kruskal's algorithm is selected from the options available in the vertical toolbar and the step-by-step feature is enabled to visualize the algorithm's process. Fig. 5 illustrates an intermediate stage of the algorithm's execution after the first few steps have completed.



Kruskal's algorithm | Weights: 93 | Step by step execution

Figure 5: Visual state of the MST Calculator canvas after step 4 of Kruskal's algorithm execution.

The step-by-step visualization continues in this manner, evaluating subsequent edges until the minimum spanning tree, containing exactly V-1 edges (6 edges in our 7-vertex example), is fully constructed. Fig. 6 displays the final output presented by the MST Calculator upon the completion of Kruskal's algorithm.

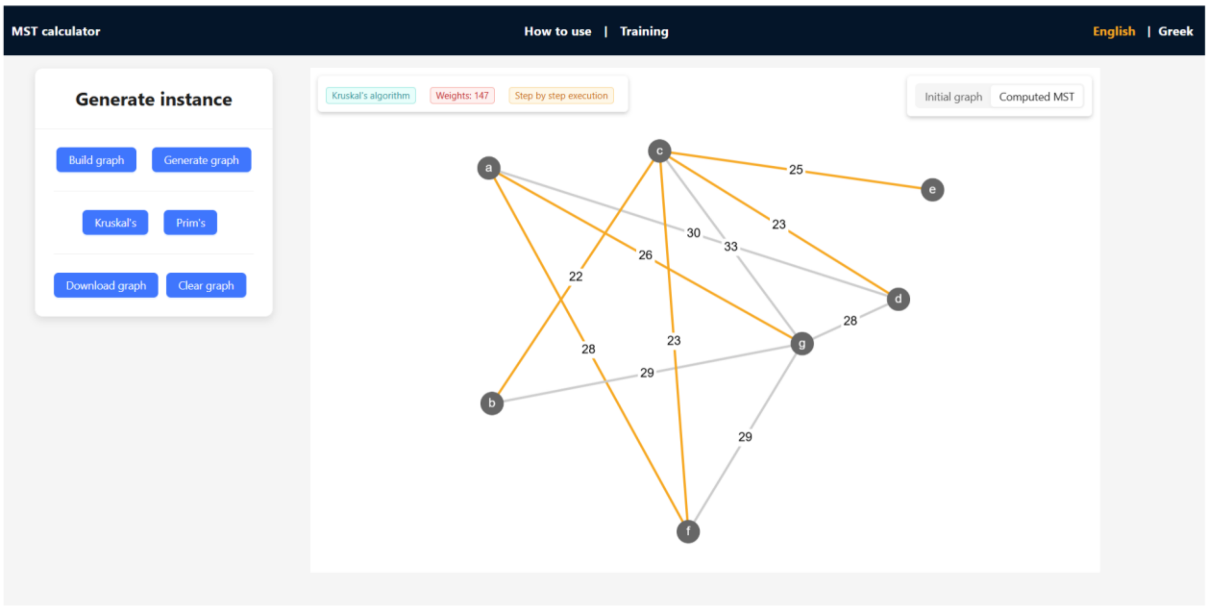


Figure 6: The edges of the computed minimum spanning tree are highlighted on the canvas.

The MST Calculator also features a Training Mode for hands-on practice. This is demonstrated in Fig. 7, which shows the start of an exercise using Prim's algorithm on an automatically generated 5-vertex graph, with vertex b highlighted by the application as the root (starting point) for constructing the minimum spanning tree.

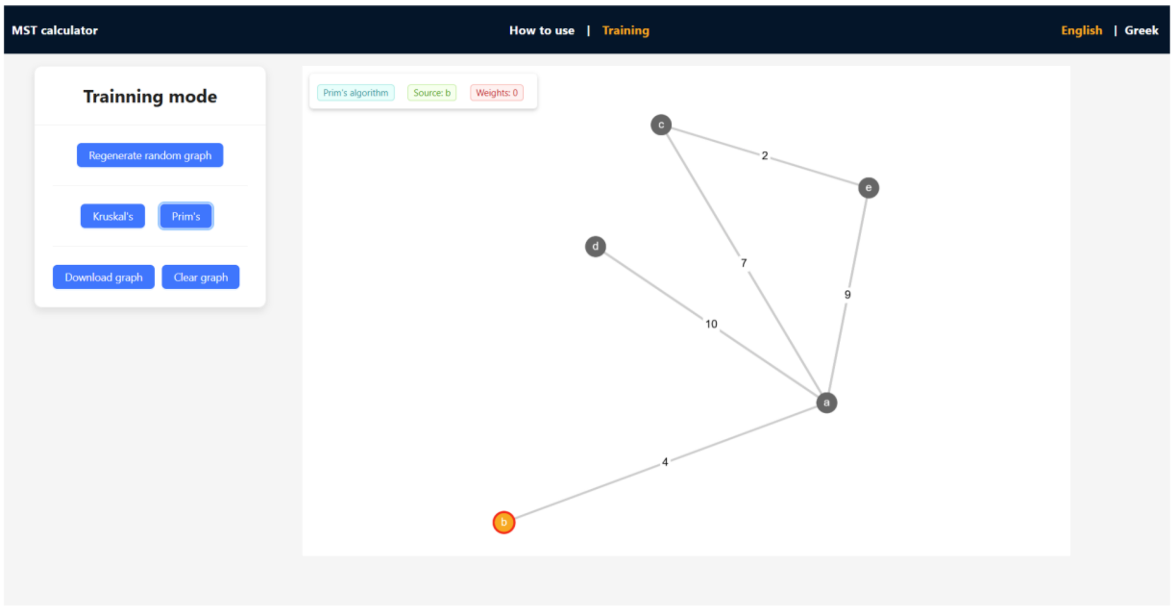


Figure 7: Training mode configured for Prim's algorithm practice.

With the starting vertex identified, users proceed by selecting edges that they believe to belong in the minimum spanning tree according to Prim's algorithm logic (i.e., by choosing the minimum-weight edge connecting the current tree to a vertex outside the tree). The MST Calculator provides immediate visual feedback on each selection. Fig. 8 demonstrates this feedback mechanism, showing the outcome after users have selected two different edges during the exercise. A correctly selected edge is highlighted in green, while an edge chosen incorrectly (not the minimum-weight connecting edge at that step) is marked with blinking red.

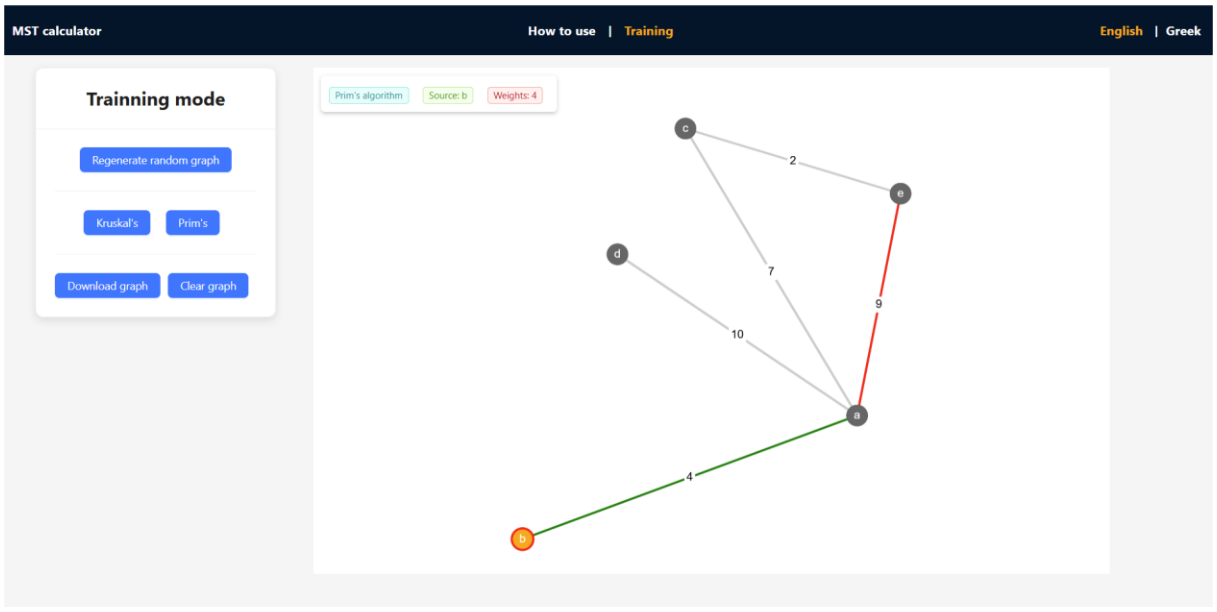


Figure 8: Example of real-time feedback in Training mode while practicing Prim's algorithm.

This walkthrough highlights how the MST Calculator integrates graph generation, algorithm visualization, and interactive practice, thereby providing a comprehensive environment for learning minimum spanning tree concepts.

4 CONCLUSIONS AND FUTURE PLANS

We designed and developed the MST Calculator application with an intention to provide students with an easy-to-use tool to support the learning and visualization of minimum spanning tree algorithms, specifically Kruskal's and Prim's methods. Through a clear user interface, graph construction capabilities, step-by-step algorithm visualizations, and a dedicated Training Mode that simulates real exercises where users are challenged with algorithm-specific tasks and receive instant feedback - a feature designed to reinforce procedural mastery and support self-assessment - the application provides users extensive practical engagement with key graph theory concepts. The inclusion of features such as random graph generation, real-time feedback, total MST weight calculation, and graph export further enhances the educational value of the application.

Future improvements could include the support for directed graphs and additional algorithms. Further responsiveness optimization and accessibility enhancements would also broaden the usability of the application across a wider range of devices and user requirements. Additionally, incorporating user tracking for learning analytics could provide valuable insights into student engagement and performance. Our MST Calculator demonstrates the potential of interactive web applications for enhancing algorithm comprehension, offering a practical resource for both students and educators engaged with fundamental computer science concepts.

REFERENCES

[1] N. P. Akpan and I. A. Iwok, "A Minimum Spanning Tree Approach of Solving a Transportation Problem," *International Journal of Mathematics and Statistics Invention*, vol. 5, no. 3, pp. 09-18, 2017.

[2] D. Cheriton and R. E. Tarjan, "Finding minimum spanning trees," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 724–742, 1976

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge: MIT Press, 2009.

[4] P. Erdős and A. Rényi, "On Random Graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[5] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of American Mathematical Society.*, vol. 7, no. 1, pp. 48–50, 1956.

[6] E. Marcotte, Responsive web design, *A List Apart*, 25 May, 2010, Accessed 28 April, 2025, Retrieved from https://alistapart.com/article/responsive-web-design/.

[7] M. Nebeling and M. C. Norrie, "Responsive design and development: Methods, technologies and current issues," in *Web Engineering. ICWE 2013*. Lecture Notes in Computer Science, vol 7977 (F. Daniel, P. Dolog, and Q. Li, eds.), pp. 510-513, Berlin: Springer, 2013.

[8] M. E. J. Newman, *Networks: An Introduction*. Oxford: *Oxford University Press*, 2010.

[9] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, Nov. 1957.

[10] Ant Financial Open Source Community, "Ant Design – A Design System for Enterprise-Level Products". Retrieved from https://ant.design/

[11] Cytoscape Consortium, "Cytoscape.js – Graph Theory (Network) Library for Visualization and Analysis." Retrieved from https://js.cytoscape.org/

[12] Evan You and Vite Contributors, "Vite – Next Generation Frontend Tooling." Retrieved from https://vitejs.dev/

[13] GitHub, "GitHub Pages – Websites for You and Your Projects," GitHub Pages, 2024. Retrieved from https://pages.github.com/

[14] GraphOnline, "Help - Find minimum spanning tree". Retreived from https://graphonline.ru/en/wiki/Help/FindMinimumSpanningTree

[15] Jan Mühlemann and Contributors, "i18next – Internationalization Framework". Retrieved from https://www.i18next.com/

[16] React Team, "React – A JavaScript library for building user interfaces," Retrieved from https://react.dev/

[17] Technical University of Munich, Department of Informatics, "Minimum Spanning Tree - Kruskal's Algorithm". Retreived from https://algorithms.discrete.ma.tum.de/graph-algorithms/mst-kruskal/index_en.html

[18] ui.dev, "Am I Responsive?," Retreived from https://ui.dev/amiresponsive?

[19] Vercel MST Calculator. Retreived from https://mst-calculator.vercel.app

[20] VisuAlgo, "Visualising data structures and algorithms through animation". Retrieved from https://visualgo.net