2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME) | 978-1-6654-1262-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICECCME52200.2021.9591017

A Simple Algorithm for finding Steiner Spanning Trees

Rituka Jaiswal
Electrical Engineering & Computer Science
University of Stavanger
Stavanger, Norway
Rituka.Jaiswal@uis.no

Reggie Davidrajuh
Electrical Engineering & Computer Science
University of Stavanger
Stavanger, Norway
Reggie.Davidrajuh@uis.no

***Abstract*—This paper presents a simple algorithm for finding Steiner spanning trees. A Steiner spanning tree connects a set of terminal vertices in a tree with minimal total edge weight. The proposed algorithm uses modified Prim's algorithm to generate the minimum weight spanning tree (MST) as the basis. Then, the non-terminal vertices with 1-degree of connection in the MST are deleted one by one until the resulting spanning tree become free from any non-terminal vertices with 1-degree of connection. Since this algorithm is easy to understand, code, and extend, it is implemented in the GPenSIM software for industrial applications (such as performance analysis of manufacturing systems and smart microgrids).**

***Index Terms*—Steiner Spanning Tree, Minimum-Weight Spanning Tree, Graph Algorithms, GPenSIM.**

I. INTRODUCTION

Graph algorithms are inevitable in the modern world, as graph algorithms power the major ICT applications we use these days. More specifically, some computer and mobile applications (such as Google Maps and social media) use extensions of spanning tree algorithms like Prim's and Kruskal's, which are considered basic graph algorithms. This paper focuses on Steiner spanning tree algorithm.

A Steiner spanning tree is a spanning tree that connects a set of terminal vertices in a tree so that the total edge weight is minimal. This paper proposes a simple algorithm that starts with the well-known Prim's algorithm. Prim's algorithm is modified for efficiency and is used to generate the minimum weight spanning tree (MST) as the basis. Then, the basis MST is refined by eliminating the non-terminal vertices with 1-degree of connection. The proposed algorithm is implemented as a part of GPenSIM and is used to analyse large manufacturing systems and determine smart microgrids.

The structure of this paper: Section-II presents Spanning Trees, including generic minimum spanning trees and Steiner spanning trees. Section-III presents the new algorithm for finding Steiner spanning trees. This section presents the algorithm, its running time, an application example, and the implementation details. Section-IV presents a short discussion.

II. SPANNING TREES

The problem of finding Steiner spanning trees are similar to minimum-weight spanning trees. Hence, the second subsection presents minimum-weight spanning trees (MST, for short). And the third subsection focuses on Steiner spanning trees. The first subsection presents a short literature review.

A. Short Literature Review

Ref. [1] is one of the earliest papers on Steiner spanning tree. The algorithm presented in this paper takes $\mathcal{O}(|V|^3)$ time. Ref. [2] is another earlier work that uses a linear programming approach. However, the running time of the approach is not clear. Ref. [3] focuses on directed graphs and uses a hybrid approach based on Chu-Liu-Edmonds technique [4], and Bilde-Krarup-Erlenkotter ascent algorithm [5]. The running time of this approach also takes $\mathcal{O}(|V|^3)$ time. Ref. [6] is a book that presents different types (classes) of Steiner spanning trees and their (industrial) applications. Ref. [7] presents a heuristic approach; however, the algorithm is more mathematically demanding.

The algorithm presented in this paper takes $\mathcal{O}(|E| + |V|log|V|)$ time, which is equivalent to $\mathcal{O}(|V|^2)$, as $|E| \approx |V|^2$ for dense graphs. However, the novelty of the proposed algorithm is its simplicity, as it is based on a well-known classic algorithm (the Prim's algorithm). Due to its simplicity, it is easy to implement the algorithm as a part of any software. As discussed in section-III-E, the proposed algorithm is implemented as part of the GPenSIM software to model and analyze discrete systems.

B. Minimum-Weight Spanning Trees (MST)

For an undirected graph of V vertices E edges, we need to choose $|E| - 1$ edges to connect all the vertices together in an acyclic network (a 'tree'). More than $|E| - 1$ edges will induce cycles.

The MST problem can be defined as followingly:

Given an undirected graph $G = (V, E)$, in which each edge $(u, v) \in E$, has a weight $w(u, v)$ specifying the cost to connect u and v. The problem is finding an acyclic subset $T \subseteq E$ that connects all of the vertices so that the total weight of the edges is minimal. $w(T) = \Sigma w(u, v), \forall w(u, v) \in T, w(T)$ is minimal.

Since the network of edges T is acyclic and connects all of the vertices, it is a tree. Hence, it is called minimum- weight spanning tree (or "minimum spanning tree (MST)", for short).

Literature provides two simple algorithms for solving MST problem: Kruskal's algorithm [8] and Prim's algorithm [9]. Both Kruskal's algorithm and Prim's algorithm are "greedy algorithms". As greedy algorithms, these two algorithms make one of several choices at every step, assuming the choice taken at that step will provide a globally optimal solution (greedy approach: local optimization leads to global optimization). Usually, greedy algorithms do not guarantee globally optimal solutions. However, Kruskal's algorithm and Prim's algorithm do provide a globally optimal solution for the MST.

Both Kruskal's algorithm and Prim's algorithm implement a "generic" minimum-spanning-tree method that grows a spanning tree by adding one edge at a time [10]. However, the new algorithm proposed in section-III is based on modified Prim's algorithm. This is because Prim's algorithm has the following advantage over Kruskal's algorithm:

- Prim starts with a source vertex (or a 'prime' vertex) and build a tree from this vertex. However, Kruskal looks for any light edge at every step and not necessarily connected with the existing tree, thus builds a set of trees (a forest) during the iterations. This means, if we stop the iterations abruptly, Prim's algorithm will give a rudimentary tree of some connected vertices, whereas Kruskal's algorithm will provide a set of rudimentary trees. Thus, Prim's algorithm strictly follows the "incremental approach" and always has a partial solution at the end of every itertation.

The running time of Kruskal's algorithm takes $\mathcal{O}(|E| \times log|V|)$ [10]. Hence, for dense graphs, Kruskal's algorithm takes $\mathcal{O}(|V|^2 log|V|)$. In contrast, Prims's algorithm take $\mathcal{O}(|E| + |V|log|V|)$ [10], which becomes $\mathcal{O}(|V|^2 + |V|log|V|)$ [10]. Therefore, Prim's algorithm runs (somewhat) faster for dense graphs.

C. Steiner Spanning Trees

A Steiner spanning tree (SST) is a more practical spanning tree, as we usually need not connect every vertex of a graph. We may be interested only in connecting a subset of vertices in a minimum-weight tree of edges. The subset of vertices that are needed to be connected is called the *Terminals*.

There are a large classes of Steiner spanning trees [6], [7], [11]. However, in this paper, we limit the scope to the elementary Steiner spanning trees. The problem of developing Steiner Spanning Trees are defined as followingly:

Given a connected undirected graph $G = (V, E)$, the weight of each edge $w(e)$ is an integer, and a set of vertices N is known as the terminals, $N \subseteq V$, then the problem is to find a subtree $T \subseteq E$ spanning all the terminals and the total weight of T is minimal.

Let $|N|$ be the number of terminals, and $|V|$ be the total number of vertices. We can classify the Steiner spanning tree problem into three cases:

- $|N| = 1$: Trivial case. There is only one terminal vertex.
- $|N| = 2$: Special case in which only two terminal vertices are needed to be connected.
- $|N| = |V|$: Special case in which all the graph's vertices are terminals.
- $2 < |N| < |V|$: The usual (non-trivial) case.

D. Steiner Spanning Trees: Trivial case, $|N| = 1$

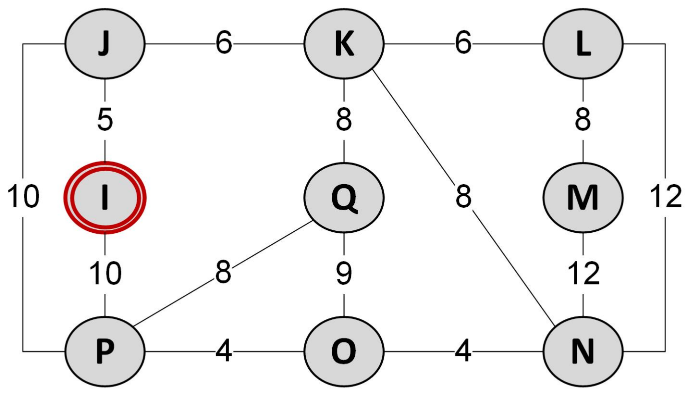

Fig. 1. Steiner Spanning Trees: Trivial case, |N| = 1.

As shown in Fig. 1, we are only interested in a single terminal vertex ('I'). Thus, there is no need to establish a tree. The single vertex 'I' becomes the tree.

E. Steiner Spanning Trees: Special case, $|N| = 2$

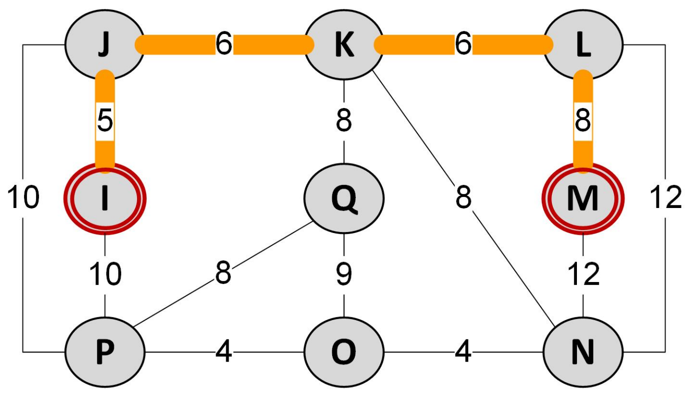

Fig. 2. Steiner Spanning Trees: Special case, |N| = 2.

As shown in Fig. 2, we need to establish a tree that connects two terminal vertices ('I' and 'M') only. This is straightforward, as we can run "Single source shortest path" (e.g., Bellman-Ford's algorithm [12], [13], or Dijkstra's algorithm [14]) from the source vertex 'I'. Upon completion of the MST, backtracking from the terminal vertex 'M' to the source 'I' will identify the path that becomes the Steiner spanning tree (it is a path than a tree).

In Fig. 2, the Steiner spanning tree that spans the terminals 'I' and 'M' and possesses some other vertices such as 'J', 'K', and 'L'. Though the focus is on the terminals ('I' and 'M'), we need some other vertices without which forming a spanning tree might not be possible. *Steiner vertices* are non-terminal vertices that happen to end up in the spanning tree; without the Steiner vertices, creating a Steiner spanning tree is may not be possible as the terminals vertices are not connected with each other directly.

F. Steiner Spanning Trees: Special case, $|N| = |V|$

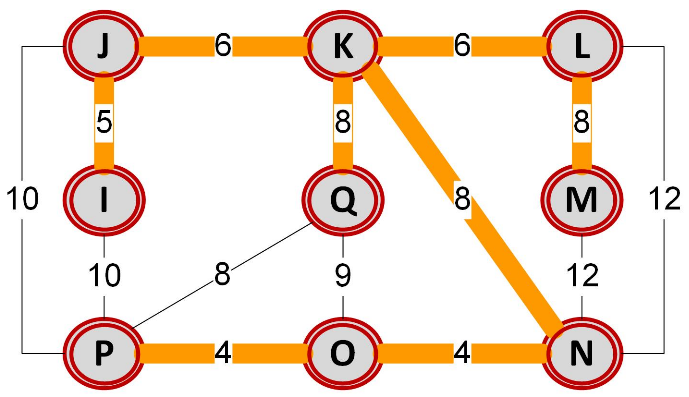

Fig. 3. Steiner Spanning Trees: Special case, |N| = |V |.

As shown in Fig. 3, $|N| == |V|$ means all the vertices are Terminal vertices. Thus, the problem becomes the usual "Minimum Weight Spanning Tree (MST)" problem. Hence, Prim's or Kruskal's algorithm can be used. Fig. 3 shows the Steiner spanning tree, which is the same as the minimum-weight spanning tree.

G. Steiner Spanning Trees: Special case, $2 < |N| < |V|$

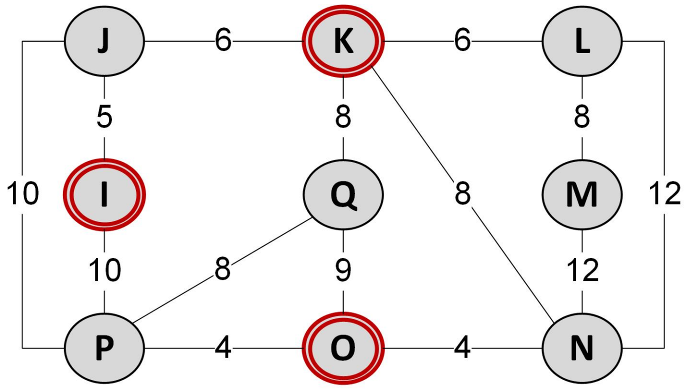

Fig. 4. Steiner Spanning Trees: Non-trivial case, |N| = 3.

Fig. 4 shows a non-trivial case ($|N| = 3$). Providing a solution for this non-trivial case ($2 < |N| < |V|$) is the goal of this paper. The proposed algorithm is given in the following section.

III. THE ALGORITHM

The algorithm, its running time, an application example, and the implementation details are given in the following subsections.

A. The Modified Prim's Algorithm

Prim's algorithm is run from a source vertex. And it will run until all the vertices that can be reached from the source vertex are added to the spanning tree. However, for a Steiner spanning tree, we need not wait until all the possible vertices that are reachable from the source vertex are added. Our primary goal is to add all the possible terminal vertices that are reachable from the source terminal. Hence, Prim's algorithm is modified so that the iterations will stop once all the Terminals are absorbed into the tree. Fig. 5 shows the modified Prim's algorithm, in which line-6 is the modification.

If some of the terminals are not reachable from the source terminal, then the iterations will run until no other vertex can be added to the spanning tree, just as in the case of the (original) Prim's algorithm.

```
PRIM_Modified ($G.V$, $G.E$, $T$)
Input: $G.V$ – set of vertices
       $G.E$ – set of edges
       $T$ – set of terminal vertices
Output: $MST$ - Minimum Spanning Tree
________________________________________
% initialization
1  sort $E$ in nondecreasing order by weight $w$
2  extract $e_1 = (u_1, v_1)$ from $E$ with minimal $w$
3  $A = \{u_1, v_1\}$
4  eliminate $e_1$ from $E$
5     $V = G.V - A$
% the main loop
6  while $V \neq \emptyset$ or $(V \cup T) \neq \emptyset$
7      extract $e = (u_i, v_i) \in E$ with minimal $w$
       such that $u_i \in A$ and $v_i \in (V - A)$
8      $A = A \cup \{v_i\}$
9      $V = V - v_i$
10     eliminate $e$ from $E$
```

Fig. 5. Modified Prim's Algorithm.

B. The Algorithm for finding Steiner Spanning Tree

The algorithm is shown in Fig. 6, and it consists of three steps:

- Step-1: In the first step, one of the terminal vertices are taken as the source vertex.
- Step-2: The modified Prim's algorithm is run from the chosen terminal vertex. Prim's algorithm will be terminated once all the Terminals are absorbed into the tree (no need to wait until all the graph's vertices, including the non-terminal vertices, are included in the tree).
- Step-3: This step is also iterative, in which we will delete the Steiner vertices (non-terminal vertices) that have one degree of connection. This step will continue to delete non-terminal vertices with one degree of connection, one by one, until there exist no more non-terminal vertices with one degree of connection.

```
Steiner_Spanning_Tree (G.V, G.E, T)
Input: G.V – set of vertices
       G.E – set of edges
       T – set of terminal vertices
Output: SST – Steiner Spanning Tree
-----------------------------------------------
% step-1: set T(1) as the source terminal vertex
1   set the source terminal vertex $T(1) = t_S$

% step-2: run Prim's modified algorithm
2   MST = Prim_modified(G.V, G.E, T)

% step-3: the main loop
6   while NOT(exist($v_i$ a Steiner_vertex with 1-degree))
7   |   Eliminate $v_i$ from MST

% the result: assign the remaining MST as the
% Steiner spanning tree
8     SST = MST
```

Fig. 6. The Algorithm for finding Steiner spanning tree.

C. Running Time

- Step-1: Takes $\mathcal{O}(1)$ time as this step only involves a simple assignment (assigning one of the terminal vertices as the source vertex).
- Step-2: This step involves modified Prim's algorithm; thus, take $\mathcal{O}(|E| + |V|log|V|)$ time, which is equivalent to $\mathcal{O}(|V|^2)$ for dense graphs.
- Step-3: This step deletes non-terminal vertices that have one degree of connection. There will be a maximum of $|V| - |N|$ number of non-terminal vertices. Thus, this step takes $\mathcal{O}(|V| - |N|)$ time.

Hence, the running time of the algorithm (RT):
$RT = \mathcal{O}(1) + \mathcal{O}(|V|^2) + \mathcal{O}(|V| - |N|) = \mathcal{O}(|V|^2)$.

D. Application Example

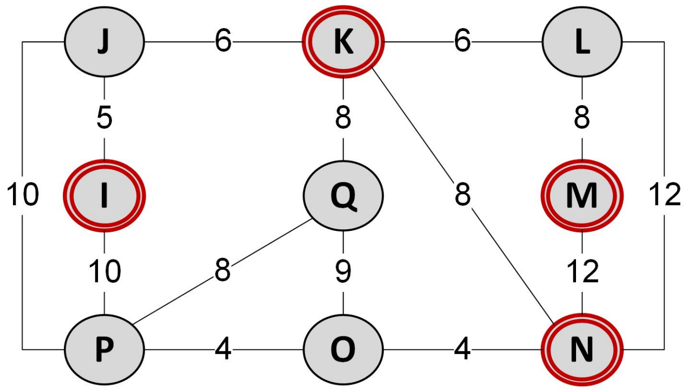

Fig. 7. A graph with four terminals.

We take the graph shown in Fig. 7, in which there are four terminals (such as 'I', 'K', 'M', and'N').

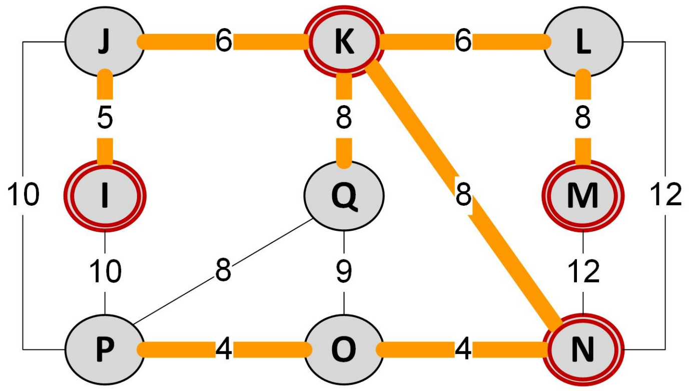

Fig. 8. Resulting spanning tree (MST) after executing the modified P rim's algorithm.

Steps-1 and 2: Fig. 8 shows the spanning tree obtained by the modified Prim's algorithm. Also, terminal 'I' was chosen as the source to start the modified Prim's algorithm. The modified Prim's algorithm did not stop abruptly, as the last terminal 'M' was selected only in the last step, as the following screen dump shows:

```
Printing the edges that are scanned to
make MST ...
Edge-1: J-I Wt: 5
Edge-2: K-J Wt: 6
Edge-3: L-K Wt: 6
Edge-4: N-K Wt: 8
Edge-5: O-N Wt: 4
Edge-6: P-O Wt: 4
Edge-7: Q-K Wt: 8
Edge-8: M-L Wt: 8
Total Weight : 49
```

Step-3: In the MST shown in Fig. 8, there are two Steiner vertices, 'P' and 'Q', with 1-degree of connection. These two Steiner vertices will be deleted. The resulting spanning tree is shown in Fig. 9.

The resulting spanning tree is shown in Fig. 9 now possesses one Steiner vertex (vertex 'O') with 1-degree of connection. This vertex has to be removed too; Fig. 10 shows the resulting Steiner Spanning tree, with a total weight of 33. Note that the resulting Steiner spanning tree shown in Fig. 10 is not optimal. The optimal tree consists of four edges (I-J, J-K, K-N, N-M), with a total weight of 31. However, finding Steiner spanning tree is an NP-hard problem, provides no optimal solutions in polynomial time [15].

E. Implementation

The algorithm is implemented in the MATLAB platform as a function of the tool known as the General-purpose Petri Net Simulator (GPenSIM). GPenSIM is developed by the second author of this paper [16], and it is freely available

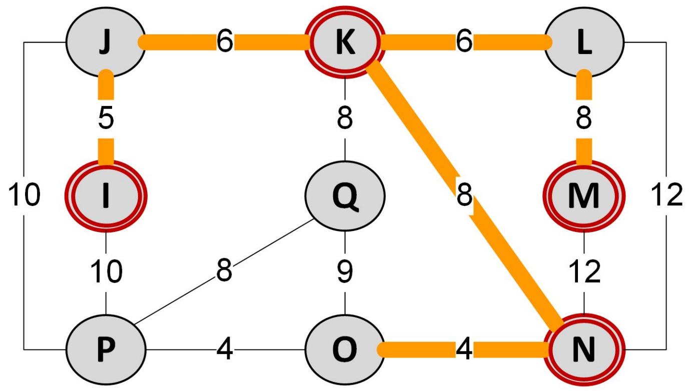

Fig. 9. Resulting spanning tree after removing Steiner vertices ‘P’ and ‘Q’.

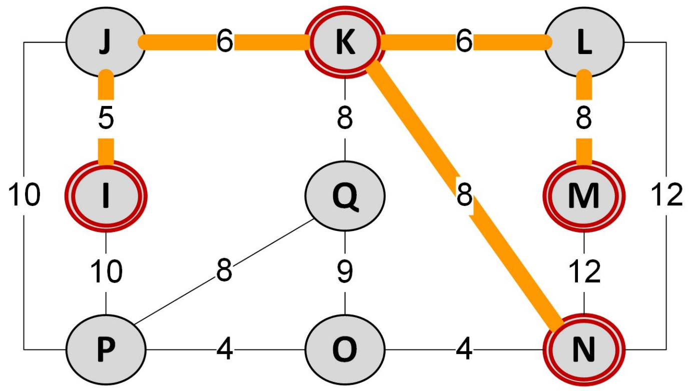

Fig. 10. Resulting spanning tree after removing Steiner vertices ‘P’ and ‘Q’.

[17]. Some universities around the world use GPenSIM for modeling, simulation, and performance analysis of discrete event systems.

IV. Discussion

This paper presents a simple algorithm for finding Steiner spanning trees. Since the algorithm is simple, it was easy to implement this algorithm as a function in the GPenSIM toolbox. Though there are several applications of Steiner spanning trees, our objective is to use Steiner spanning trees in two specific cases:

- Performance of manufacturing systems: Since a modern manufacturing system these days is large, it is not easy to model, simulate, and perform analysis. Hence, [18] proposes decomposing the model as modules and then analyze these modules one by one. Steiner spanning tree could help to decompose a large legacy model into modules. For example, once a small number of manufacturing elements are identified, these elements can be extracted as a subgraph from the overall graph as a Steiner spanning tree [19].
- Electric Grid: Ref. [20] presents an infrastructure for optimal energy usage in smart neighborhoods. When several houses use different types of electric utilities in a neighborhood, we may need to estimate the “smart microgrid” - a group houses and the batteries - so that a stable power supply can be designed. In this case, the group houses and the batteries form a Steiner spanning tree.

References

[1] L. Kou, G. Markowsky, and L. Berman, “A fast algorithm for steiner trees,” *Acta informatica*, vol. 15, no. 2, pp. 141–145, 1981.

[2] L. Gouveia, L. Simonetti, and E. Uchoa, “Modeling hop-constrained and diameter-constrained minimum spanning tree problems as steiner tree problems over layered graphs,” *Mathematical Programming*, vol. 128, no. 1, pp. 123–148, 2011.

[3] R. T. Wong, “A dual ascent approach for steiner tree problems on a directed graph,” *Mathematical programming*, vol. 28, no. 3, pp. 271–287, 1984.

[4] H. Cao, R. Mo, and N. Wan, “3d modelling of a frame assembly using deep learning and the chu–liu–edmonds algorithm,” *Assembly Automation*, 2019.

[5] T. Boffey, *Graph theory in operations research*. Macmillan International Higher Education, 1982.

[6] X. Cheng and D.-Z. Du, *Steiner trees in industry*. Springer Science & Business Media, 2013, vol. 11.

[7] G. Robins and A. Zelikovsky, “Improved steiner tree approximation in graphs.” in *SODA*. Citeseer, 2000, pp. 770–779.

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

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

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

[11] R. Graham and F. Hwang, “Remarks on steiner minimal trees,” *Bull. Inst. Math. Acad. Sinica*, vol. 4, no. 1, pp. 177–182, 1976.

[12] R. Bellman, “On a routing problem,” *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[13] L. R. Ford Jr, “Network flow theory,” Rand Corp Santa Monica Ca, Tech. Rep., 1956.

[14] E. W. Dijkstra *et al.*, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[15] M. R. Garey and D. S. Johnson, “The rectilinear steiner tree problem is np-complete,” *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[16] R. Davidrajuh, *Modeling Discrete-Event Systems with GPenSIM*. Cham: Springer International Publishing, 2018.

[17] GPenSIM, “General-purpose Petri net simulator,” http://www.davidrajuh.net/gpensim, Tech. Rep., 2019, accessed on 20 July 2020.

[18] R. Davidrajuh, B. Skolud, and D. Krenczyk, “Performance evaluation of discrete event systems with gpensim,” *Computers*, vol. 7, no. 1, pp. 1–8, 2018.

[19] ——, “Gpensim for performance evaluation of event graphs,” in *Advances in Manufacturing*, ser. Lecture Notes in Mechanical Engineering, no. 201519. Cham: Springer International Publishing, 2018, pp. 289–299.

[20] R. Jaiswal, R. Davidrajuh, and C. Rong, “Fog computing for realizing smart neighborhoods in smart grids,” *Computers*, vol. 9, no. 3, p. 76, 2020.