symmetry

MDPI

Article

An Efficient Shortest Path Algorithm: Multi-Destinations in an Indoor Environment

Mina Asaduzzaman [1], Tan Kim Geok [2,*], Ferdous Hossain [2,*], Shohel Sayeed [1], Azlan Abdaziz [2], Hin-Yong Wong [3], C. P. Tso [2], Sharif Ahmed [2] and Md Ahsanul Bari [1]

1 Faculty of Information Science and Technology, Multimedia University, Melaka 75450, Malaysia; 1181402835@student.mmu.edu.my (M.A.); shohel.sayeed@mmu.edu.my (S.S.); 1141123630@student.mmu.edu.my (M.A.B.)
2 Faculty of Engineering and Technology, Multimedia University, Melaka 75450, Malaysia; azlan.abdaziz@mmu.edu.my (A.A.); cptso2014@gmail.com (C.P.T.); 1171402467@student.mmu.edu (S.A.)
3 Faculty of Engineering, Multimedia University, Cyberjaya 63100, Malaysia; hywong@mmu.edu.my
* Correspondence: kgtan@mmu.edu.my (T.K.G.); ferdous.mbstu.cse@gmail.com (F.H.); Tel.: +60-013-613-6138 (T.K.G.); +60-112-108-6919 (F.H.)

Abstract: The shortest path-searching with the minimal weight for multiple destinations is a crucial need in an indoor applications, especially in supermarkets, warehouses, libraries, etc. However, when it is used for multiple item searches, its weight becomes higher as it searches only the shortest path between the single sources to each destination item separately. If the conventional Dijkstra algorithm is modified to multi-destination mode then the weight is decreased, but the output path is not considered as the real shortest path among multiple destinations items. Our proposed algorithm is more efficient for finding the shortest path among multiple destination items with minimum weight, compared to the single source single destination and modified multi-destinations of Dijkstra's algorithm. In this research, our proposed method has been validated by real-world data as well as by simulated random solutions. Our advancement is more applicable in indoor environment applications based on multiple items or destinations searching.

Keywords: graph; multi-destination; road network; shortest path; indoor

Citation: Asaduzzaman, M.; Geok, T.K.; Hossain, F.; Sayeed, S.; Abdaziz, A.; Wong, H.-Y.; Tso, C.P.; Ahmed, S.; Bari, M.A. An Efficient Shortest Path Algorithm: Multi-Destinations in an Indoor Environment. *Symmetry* **2021**, *13*, 421. https://doi.org/10.3390/sym13030421

Academic Editor: Peng-Yeng Yin

Received: 28 December 2020
Accepted: 31 January 2021
Published: 5 March 2021

1. Introduction

In computer science, finding the shortest path in the complex indoor environment is a fundamental issue [1]. The solution to this issue is to use a bi-directional graph or road network to find the path with the minimal summation of edge weights among all reachable paths from start to end nodes [2]. The bi-directional graph considered in this paper is either directed or undirected, finite, simple, as well as connected [3]. The definition of a single pair shortest path is a path that contains only one destination. In conventional problems, the distances among the nodes are ascertained. The goal of the conventional single-source SPP is to get the minimal cost path from the source node to the destination node. However, the fuzzy number technique can be utilized instead in precarious environments [4]. The problem arises when the graph contains more than one destination node [5]. The set of destination nodes is a subset of all network nodes and it becomes a multi-destination shortest path problem [6], which has many real-life applications, such as finding the shortest path among multi-destination items in supermarkets, warehouses, libraries, road networks [7], robotics industries [8], service compositions [9,10], and multicast routes [11]. The different techniques found in the literature are calculating either a multi-destination or single-pair shortest path to solve these issues [12]. A^* is an ordinary BFS algorithm that depends on a heuristic function to demonstrate the search towards finding the shortest path from a source node to a destination node in a grid [13]. As of now, Dijkstra's is a renowned path searching algorithm that is mainly used for searching the shortest path from

a single source to a single destination introduced by Edsger W. Dijkstra in 1956. A typical single-source single-destination, Dijkstra's algorithm, is used for single-pair shortest path problems [14]. It keeps up a set S of unraveled nodes, consisting of those hubs or nodes whose ultimate shortest path between the starting node verses has ascertained as well as labels t(i), conserving the upward bound of the smallest pathway distance among v_s and v_i [15]. The node $v_m \in V/S$ with the minimal t(i) is continually chosen by the algorithm, adding v_m to S, as well as updating t(i) as the node v_m [16]:

In the first step. Set $t(v_s) = 0$ and $t(v_j) = w_{sj.}$
For other nodes $S = \{v_s\}$, $L = V/S$
In the second step.
Chose a hub v_m from L, $t(v_m) = min_{v_j \in L} t(v_j)$,
if $t(v_m) = 1$, then stop, else move to third step.
In the third step set.

$$S = S \cup \{v_m\}, L = L \backslash \{v_m\}$$

if $L = \emptyset$, then stop, else move to forth step.
In the fourth step.
For each $v_j \in L$, update

$$t(v_j) = \min\{t(v_j), t(v_m) + w_{mj} \quad (1)$$

Move to first step.

In the above steps, $V = \{v_{1,} v_1, \ldots, v_n\}$ is the set of nodes, S is the set of the source nodes, L is a list, and $W = w_{ij}$ is the weight matrix. The above steps demonstrate the shortest path calculation following the steps of Dijkstra's algorithm and the final output is expressed in Equation (1).

The Modified Dijkstra's Multi-Source Multi-Destination (MDMSMD) algorithm is applied for the multi-destination shortest path problem [17]. For the single-source multi-destination problem, the conventional Dijkstra's single-source single-destination (CDSSSD) algorithm can be used. Although, in this network, the starting point is one node and destinations are a set of multiple nodes as a source node (s) and multiple destination nodes (w, x, y, z) as shown in Figure 1. As a result, it is necessary to calculate the shortest path between s to w, s to x, s to y, and s to z.

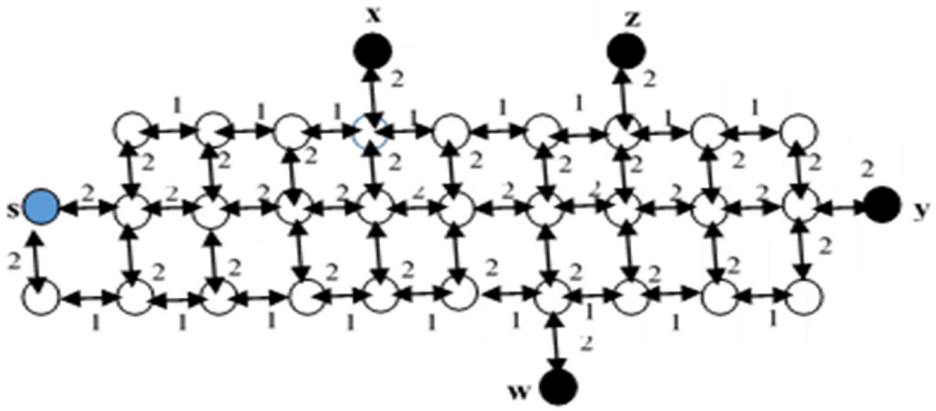


Figure 1. Bi-directional road network.

From the sample road network Figure 1, the steps of CDSSSD in Figure 2 is displayed. Figure 2a is the first step that calculates the shortest path between s and w from all accessible paths. Figure 2b is the second step that calculates the shortest path between s and x. Figure 2c shows the third step that calculates the shortest path between s and y, Figure 2d also shows the shortest path between x and z. In Figure 2e, all the calculated shortest paths among s and the destination nodes w, x, y, and z [18]. Here, the possible number of shortest paths is four. However, the number of paths can be converted to a single

path from a source to the last destination node of a set of destination nodes by applying the MDMSMD. For example, a source node is s and the destination nodes are w, x, y, and z as per Figure 1. Hence, the shortest paths are between s and w, s and x, s and y, and s and z. The path of MDMSMD is s → w → x → y → z and has been expressed in Figure 3. In Figure 3a–e are respectively shown the shortest path between s → w is 10, w → x is 9, x → y is 11, y → z is 8 and s → w → x → y → z is 38. The minimum summation of edge weights of MDMSMD is less than CDSSSD. The MDMSMD solved the multi-destination path problem, with some limitations [19]. The MDMSMD calculates the shortest path from each source to each destination node, the previous destination node is the next source node, and calculates the shortest path from the next source to the next destination node until the last destination node. This algorithm can only calculate the shortest path between one pair at a time [20], it cannot calculate the shortest path between a source node and a set of destination nodes at the same time.

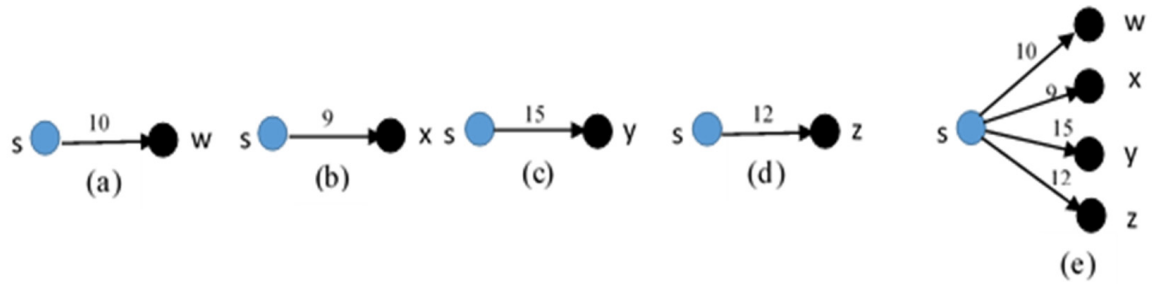


Figure 2. CDSSSD.

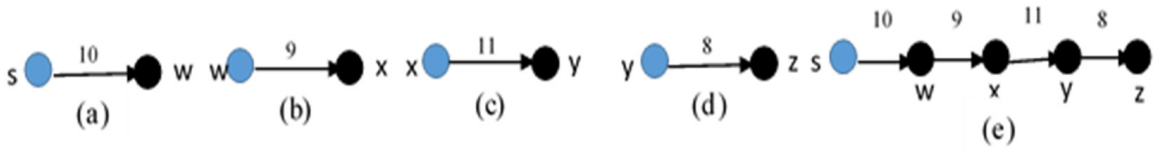


Figure 3. MDMSMD.

The objective of this research is to develop an efficient algorithm for searching the smallest path between a given source and the multi-destination nodes. The name of this proposed algorithm is the efficient algorithm for multi-destination shortest path problem (EAMDSP)—it accentuates the searching area and enhances the algorithm performance. The new concept is based on considering the outcome of the edge weights of the input graph on searching speed to acquire the goal. Using this concept, the EAMDSP finds the nearest destination node among the multiple destination nodes from the source node. After getting the first nearest destination node, it becomes the next source node and again calculates the next nearest destination node among the remaining multiple destination nodes until the last or rest destination node. Finally, the shortest path between a source node and the last destination node that decreases the summation of weights is achieved, which enhances the speed of searching for the desired shortest path. The proposed algorithm acquires the benefits of simplicity as well as the effectiveness of the computations. The outcomes of this algorithm using a real dataset and the simulated random solution will show that the proposed algorithm is better than the existing algorithm.

In this article, the following aims are discussed: Section 2, for the multi-destination shortest path problem, the existing method and solution have been reviewed as well as discussed. We propose an algorithm for searching the multi-destination shortest path in indoor environments in Section 3. In Section 4, the performance of the proposed algorithm is compared to the existing method and the conclusion is in Section 5.

2. Related Work

Nowadays, many researchers are working on graphical solutions and pathfinding to resolve crucial application issues [21–27]. Dijkstra's algorithm is used for finding the solution to the shortest path problem in multi-destination nodes [28]. The Fibonacci heap [29,30] has been utilized in the multi-destination Dijkstra's algorithm and is considered an innovative improvement. Searching the shortest paths by the node combination method is introduced by Lu et al. [31] by continually integrating with the source node's contiguous neighbors. However, multi-destination Dijkstra's algorithm is faster than their method. A quicker strategy with a small number of positive integer weights is introduced by Orlin et al. [32]. Thorup [33,34] invents an effective algorithm to solve the limitation of the Orlin et al. method which is an integral edge weight in every direction, as well as an undirected graph. In reality, no method has been utilized as an input of both undirected and directed graphs [35]. The breadth-first search, as well as the Bellman-Ford algorithm, are proposed respectively in the literature, while the graph contains all equal edge positive/negative weights. Searching the shortest path via a topological sequence [16] in linear time is feasible for a directed graph. An algorithm is introduced by Xu et al. [36] to effectively give a solution to the issue, particularly for infrequent networks.

In an article by [37], calculating the path as a usual job, so for every application [38,39] accelerate procedures perform based on the kind of data. A multi-destination feature of Dijkstra's algorithm is becoming compatible with recent applications [40,41]. By utilizing an amended version of multiple destinations, Dijkstra's algorithm is able to solve the bi-objective multi-destination shortest path problem in the pathway network which is introduced by Ticha et al. [7]. Their searching technique at each repetition of a process is to choose some parts of the path for a non-dominated path to arrive at one of the destination nodes. In the field of time schedule networks, Jin et al. [42] modified the multi-destination method for searching for the shortest path. First, they developed a path calculation method, then added it to the modified algorithm. Mainly, the shortest path calculation depends on the network constraints on its edges. By considering both node weights and edge weights, the multi-destination Dijkstra's method enhancement is proposed by Ananta et al. [43]. For the software-defined networks, this method transmits data packets to all destination nodes. The limitations of multi-destination Dijkstra's algorithm have been discussed in Section 1. This section also discussed the related algorithms, but they have some limitations such as one is designed only for a directed graph or only works on positive edge weight and another contains both positive and negative edge weight. The proposed algorithm has been designed for removing their short-comings and can perform on a bi-directional graph or road network, as well as finding the shortest path from the starting node to the multiple destination nodes, as is required in indoor environments.

3. Algorithms

Algorithm 1 (CDSSSD) and Algorithm 2 (MDMSMD) are used as reference algorithms to compare the performance of our proposed Algorithm 3. The steps of CDSSSD, MDMSMD, and EAMDSP are provided in this section to demonstrate a clear overview.

3.1. Algorithm: CDSSSD

Algorithm 1 Input: Graph: G (V, E, W), source node: S, set of multiple destination nodes: T
Output: Paths from source to destination nodes

```
1.    Declare: L
2.        For (i = 0; I < T.Length; i++)
3.            Declare: nodes_path
4.            n: = G.GetLength(0);
5.            distance: = new int[n];
6.            For (int k = 0; k < n; i++)
7.              distance[i]: = int.MaxValue;
8.            End For
9.            distance[S]: = 0;
10.           used: = new bool[n];
11.           Previous: = new int?[n];
12.           While (true)
13.             minDistance: = int.MaxValue;
14.             minNode: = 0;
15.             For (int m = 0; m < n; m++)
16.               If (!used[m] && minDistance > distance[m])
17.                 minDistance: = distance[m];
18.                 minNode: = m;
19.               End If
20.             End For
21.             If (minDistance == int.MaxValue)
22.               break;
23.             End If
24.             used[minNode]: = true;
25.             For (int l = 0; l < n; l++)
26.               If (G[minNode, l] > 0)
27.                 shortestToMinNode: = distance[minNode];
28.                 distanceToNextNode: = G[minNode, l];
29.                 totalDistance: = shortestToMinNode + distanceToNextNode;
30.                 If (totalDistance < distance[l])
31.                   distance[l]: = totalDistance;
32.                   previous[l]: = minNode;
33.                 End If
34.               End If
35.             End For
36.           End While
37.           If (distance[T[i]] :== int.MaxValue)
38.             return null;
39.           End If
40.           currentNode: = T[i];
41.           While (currentNode ! = null)
42.             Add nodes_path->col: = currentNode.Value;
43.             currentNode: = previous[currentNode.Value];
44.           End While
45.           col1: = S;
46.           For(j = 0;j < nodes_path.Length;j++)
47.             Add L->col: = nodes_path[j]
48.             col1: = nodes_path[j];
49.         End For
50.     End For
51.   Return L;
```

3.2. Algorithm: MDMSMD

Algorithm 2 Input: Graph: G (V, E, W), source node: S, set of multiple destination nodes: T
Output: A path from source to destination nodes and shortest path among destination nodes

```
Declare: L
    For (i = 0;I < T.Length;i++)
        Declare: nodes_path
        n: = G.GetLength(0);
        distance: = new int[n];
        For (int k = 0; k < n; i++)
            distance[i]: = int.MaxValue;
        End For
        distance[S]: = 0;
        used: = new bool[n];
        Previous: = new int?[n];
      While (true)
            minDistance: = int.MaxValue;
            minNode: = 0;
            For (int m = 0; m < n; m++)
                If (!used[m] && minDistance > distance[m])
                  minDistance: = distance[m];
                  minNode: = m;
                End If
            End For
            If (minDistance == int.MaxValue)
                break;
            End If
            used[minNode]: = true;
            For (int l = 0; l < n; l++)
              If (G[minNode, l] > 0)
                  shortestToMinNode: = distance[minNode];
                  distanceToNextNode: = G[minNode, l];
                  totalDistance: = shortestToMinNode + distanceToNextNode;
                  If (totalDistance < distance[l])
                    distance[l]: = totalDistance;
                    previous[l]: = minNode;
                  End If
                End If
            End For
        End While
        If (distance[T[i]] :== int.MaxValue)
            return null;
        End If
        currentNode: = T[i];
        While (currentNode ! = null)
            Add nodes_path->col: = currentNode.Value;
            currentNode: = previous[currentNode.Value];
        End While
        col1: = S;
        For(j = 0;j < nodes_path.Length;j++)
            Add L->col: = nodes_path[j]
            col1: = nodes_path[j];
        End For
      S: = T[i];
    End For
  Return L;
```

3.3. The Proposed EAMDSP

In algorithm 3, we consider that G = (V, E, W) is a graph containing N nodes (vertices), a set of edges, and a set of weights. The vertex set V has $n = |V|$ collection of nodes, E contains $m = |E|$ and is the collection of edges, as well as W, the collection of weights.

Let $_{We} \in W$ be the weight of edge $e = (u, v) \in E$ as $u \to v$ for some $u, v \in V$.

The objective is to find a collection of shortest paths among vertices s ∈ V as an assigned origin node to the collection of multi-destination nodes $d_i \in D \subseteq V$, $i \in I = \{1, 2,..., |D|\}$. For instance, $w_{path(s,\, d_i)} = \sum_j^{|path(s,\, d_i)|} w_{ej}$ where the $path(s, d_i)$ is the weight of path and pd_i ($i \in I$) is the shortest path containing the minimal weight of $w_{path(s,\, d_i)}$ among all accessible paths from s to d_i. Here, if there is no path from s to d_i then $\delta pd_i = \infty$ else $\delta pd_i = \delta path(s,d_i) = \min\{w_{path(s,di)}\}$. Figure 1 explains the details of this scenario.

The proposed algorithm, EAMDSP, provides the shortest path from source node $s \in V$ to all destination nodes or multi-destination nodes $d_i \in T \subseteq V$, $i \in I$ in the graph G = (V, E, W).

Firstly, find the nearest destination node from the set of destination nodes by calculating the shortest path among all possible paths using Dijkstra's single source-single destination algorithm.

Secondly, the nearest destination node from the source node is identified by minimum visited nodes and a minimum sum of weights between the source and each destination node.

Thirdly, after getting the first nearest destination node, it is considered as the next source node and starts the calculation of the nearest destination node from the rest of the destination nodes. This procedure is repeated until the last rest destination node. Getting the visited nodes from the source to the first nearest node combines the visited nodes, and goes for the next sources and continues until the last shortest path. The minimum sum of weights also adds as the above-visited nodes. Finally, the path that is found from the source to the last destination is the shortest path between the source to multi-destination nodes with minimum visited nodes and the minimal sum of weights.

For instance, in Figure 1, the source node is s and destination nodes are sequentially w, x, y, and z. In Figure 4, the steps of EAMDSP have been demonstrated in Figure 1, with a sample bi-directional road network with weights and nodes. In step (a) of Figure 4, the sum of weights calculated from s to w is 10, s to x is 9, s to y is 15, and s to z is 12, and are found after calculating the shortest path among possible accessible paths.

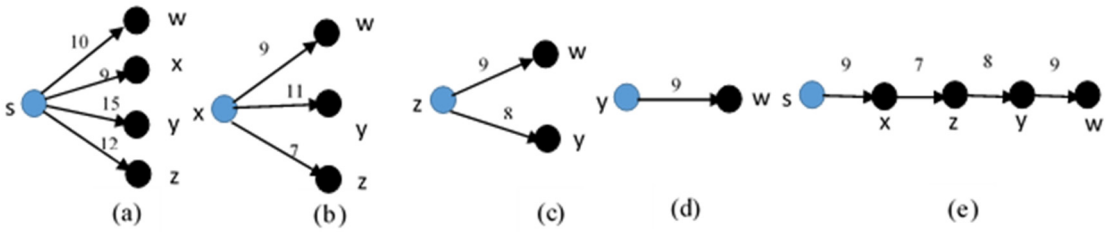


Figure 4. Graphical representation of EAMDSP.

First, each shortest path is s → x then x becomes the source node in the next step (b) of Figure 4. Similarly, the rest (w, y, z) are destination nodes, and again calculation on the sum of the weights from x to w is 9, x to y is 11, and x to z is 7. Secondly, the shortest path is x → z and combined with the minimum visited nodes between x → z. Furthermore, the previous minimum visited nodes between s > x are combined with the minimum sum of weights from s → x to x → z.

Now in step (c) of Figure 4, z is the source and w and y are destination nodes. Again, the calculation is done on the shortest path between z → y and z → w. The minimum sum of weights between z → y is eight and z → w is nine. the shortest path is z → y and combined with the minimum visited nodes between z > y. The previous minimum visited nodes between z > y are combined with the minimum sum of weights from s > x > z to z > y.

In step (d) of Figure 4, calculate the shortest path between y to w and get the sum of the weight values. Finally, the shortest path is s → x → z → y → w, and the total sum of weights is 33.

The MDMSMD working has been discussed in Section 1. The EAMDSP calculates the nearest destination node from the set of destination nodes that is the main difference from the MDMSMD. In the above example, the MDMSMD path will be s → w → x-y-z and the total sum of weight is 38. The CDSSSD total sum of weights is 46 from the above example. The number of total visited nodes and the sum of minimum weights of EAMDSP is less than the MDMSMD and the CDSSSD. As a result, EAMDSP is more efficient than MDMSMD and CDSSSD.

Algorithm 3. Raised algorithm: EAMDSP

Input: Graph: G (V, E, W), source node: S, set of multiple destination nodes: T
Output: A path from source to destination nodes and shortest path among destination nodes

```
1.    Declare: L1, L2, L3
2.    While(T.Length>0)
3.        For ( I = 0;I < T.Length;i++)
4.            nodes_path: = singleSourceSingleDestinationAlgorithm(G,S,T[i])
5.            For(j = 0;j < nodes_path.Length;j++)
6.                length_path: = length_path + G[nodes_path [j], nodes_path[j + 1]]
7.                Add L2->col1: = nodes_path[j], L2->col2: = T[i]
8.            End For
9.            Add L1->col1: = T[i], L1->col2: = length_path
10.       End For
11.       L1 sorting according to length_path
12.       e_Dest_Node: = L1->col1
13.       For (m = 0; m < L2.Length; m++)
14.         IF L2[m]->col2== e_Dest_Node
15.           Add L3->col: = L2[m]->col1
16.         End IF
17.       End For
18.       S: = e_Dest_Node
19.       Remove the e_Dest_Node value from T
20.     L1: = new initialize
21.     L2: = new initialize
22.     End While
23.     Return L3;
```

In the above algorithm, the variables L1 and L2 are declared as a two-dimensional array and L3 is a one-dimensional array for data storage. The first "while" loop runs until the last destination nodes come. The first "for" loop runs until the number of destination nodes are exhausted. The single-source single-destination algorithm is called to find the shortest path for visiting nodes among the source nodes to each destination node. The second "for" loop runs until the total number of shortest paths finding is completed for visiting nodes. Then the total path length or weight is calculated among the shortest path of visited nodes. Store each visited node into col1 of L2 and each destination node into col2 of L2. After that, store each destination node into col1 of L1 and path length or weight from a source to each destination node into col2 of L1. Sorting L1 according to path length or weight (smallest to biggest). Top each destination node from L1 → col1 first index (the smallest distance). Another new "for" loop is started for looping until L2 length or several destination nodes are covered. IF condition L2 [m]—>col2 == e_Dest_Node, then store the visiting nodes from a source to a destination in L3. S: = e_Dest_Node, replace current destination node value to source node value like the previous destination node is the next source node. Remove each destination node value from D. Then, L1: = new initialize or

empty and L2: = new initialize or empty. Finally return L3, where L3 contains the source to all destination visiting nodes with the shortest path.

In Figure 5, ph_u is the shortest path and $p'h_u$ is an alternative path s $\rightarrow u$ via hubs i and j. S is a set (shady zone) of all hubs whose ultimate smallest pathway from the source s has been ascertained.

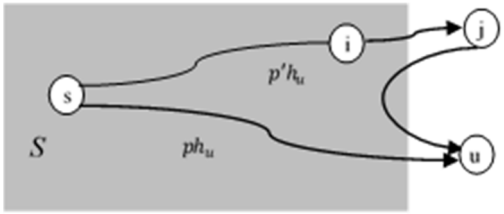


Figure 5. Source and destination shortest pathfinding in graphical representation.

Theorem 1. *(Exactness) The ultimate smallest path among destination nodes $d_i \in D$ is constantly provided by EAMDSP, for example, $d_i.t = \delta path\ (s, sd_i)$, $i \in I$. The penultimate set T_i contains several nodes, which are linked to the destination node d_i.*

Proof. For each $u \in V$ and $t \in T$ assuming the algorithm found the label $u.t$, as well as the smallest path's weight, is δpath(s,u) *for* $s \rightarrow u$. It aims to prove $u.t = \delta$path(s,u) where each vertex u returned at the end of the EAMDSP. □

Initial case, $s.t = \delta$path(s,s) = 0, when S = {s} for every $v \in S$ which considers $v.t$ shows the real smallest path's weight among s and v,

$$v.t = \delta\text{path}(s,v) \tag{2}$$

In this algorithm, -1 is used for u as a subsequent vertex to set S via path ph_u, therefore, $u.t = \delta ph_u$. It will prove this path to be the minimum distance,

$$u.t = \delta\text{path}(s,u) \tag{3}$$

Assume the minimum distance s $\rightarrow u$ is the alternate path $p'h_u$ instead of the path ph_u. The cost of the path ph_u is more than the cost of the path $p'h_u$

$$ph_u > p'h_u = u.t \tag{4}$$

Equation (4) is explained in Figure 2, the path $p'h_u$, begins in S as well as at a few points omits S to arrive at the node u that is as of now not in S. Assume (m, n) is the 1st edge toward $p'h_u$ that departs from S. Evidently, $w_{path(s,m)} + w_{path(m,n)} < \delta_{p'h_u}$. While m $\in$ S, m.t is the cost of the minimum distance $s \rightarrow m$ path by the inductive hypothesis, $m.t = \delta_{path(s,m)} \leq w_{path(s,m)}$, which implies $m.t + w_{(m,n)} \leq \delta_{p'h_u}$. The algorithm must update n.t while n is adjacent to m, which means $n.t \leq m.t + w_{(m,n)}$. The u must have the shortest value due to the algorithm picked u, in other words, $u.t \leq n.t$. To sum up these disparities, we present the outcomes in Section 4.

4. Results and Discussion

In this section, the detailed simulation results of EAMDSP, CDSSSD, and MDMSMD will be presented. To evaluate the proposed EAMDSP simulation performance CDSSSD and MDMSMD simulation results on the same layout were used. The performance was evaluated concerning the number of nodes visited and the weight or cost. Two different scenarios, one was the car spare parts shop and another was the supermarket used for getting the comprehensive results. In the car spare parts shop scenario, the simulations were run in six unique phases as in four, six, eight, ten, twelve, and fifteen items searching layout. Three shortest path algorithms named CDSSSD, MDMSMD, and EAMDSP were used on

both scenarios in different phases for getting a different result. The results comparisons are showed in the layout simulation, the summary table, and the graph.

Figure 6 shows the layout of a spare parts shop containing shelves (nodes) and road networks. The road network also consists of nodes. Every node has a unique number with longitude and latitude. The road network's one node is connected to the neighbor node with edge and weight value. The product/item is placed on the shelve node. The shelf node is connected to the road network's node with an edge and the edge has a weight value. When searching for a product, the product's shelf node number is considered to be the destination node. The shortest path was calculated between the source point (starting point) and the destination node, using the minimum the total nodes visited and their minimum weight or cost.

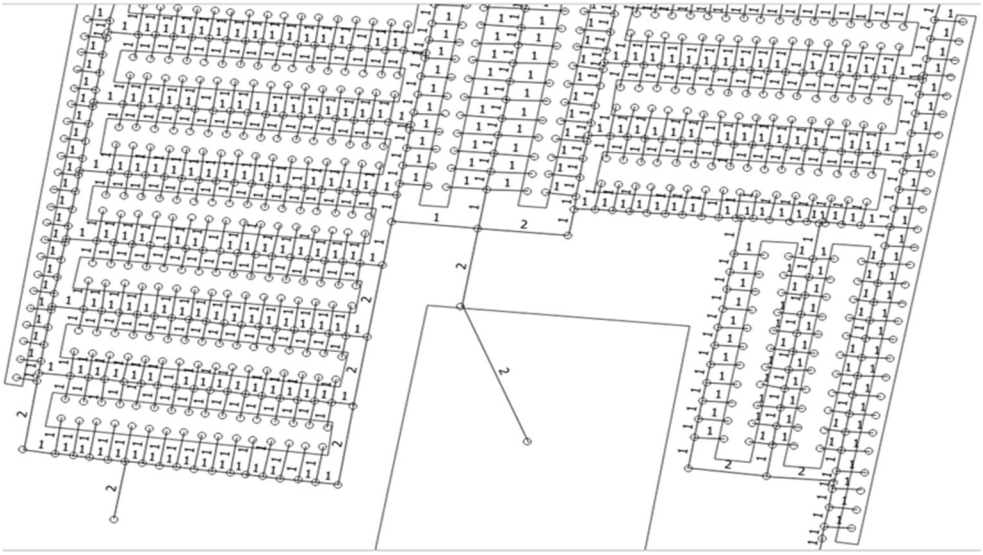


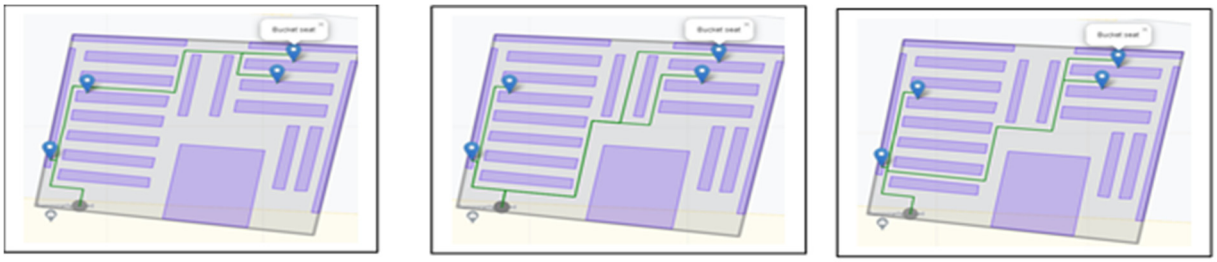
Figure 6. Drawing layout in a car spare part shop with nodes.

Table 1 represents the input which is the randomly selected four items. These input items were used for the same simulation layout of three algorithms such as CDSSSD, MDMSMD, and proposed EAMDSP.

Table 1. Selected four items and categories (input).

Item	Category
Antenna cable	Audio/video devices
Bumper	Body components, including trim
Adjusting mechanism	Braking system
Bucket seat	Car seat

Figure 7a–c are car spare parts shop simulation result of EAMDSP, CDSSSD and MDMSMD respectively, which show the shortest path among four items. Table 2 shows the output summary data for the four-item search. EAMDSP for the shortest path between the above four items visited 93 nodes and had a total weight of 95. However, using CDSSSD and MDMSMD for the same items, the visited nodes were 124 and 120 and the total cost was 136 and 125, respectively. Therefore, the proposed EAMDSP is efficient for visited nodes and total weight. Finally, total visited nodes or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD. The detailed data for the four items are shown in Appendix A.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 7. Simulation on a car spare parts shop scenario using EAMDSP, CDSSSD, and MDMSMD for a four-item search (output scenario).

Table 2. Summary data of EAMDSP, CDSSSD, and MDMSMD for a four-item search.

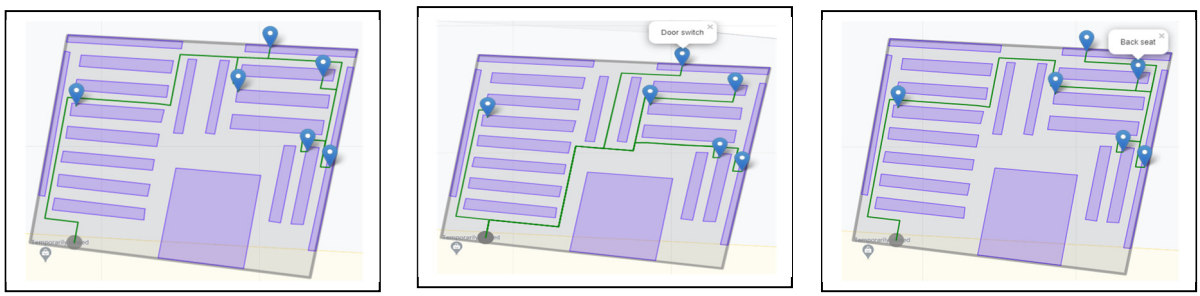
SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	93	95
2	CDSSSD	124	136
3	MDMSMD	120	125

Table 3 represents the input value, which is six items randomly selected. These input items were used on the same simulation layout as the three algorithms CDSSSD, MDMSMD, and EAMDSP.

Table 3. (Input) Selecting six items and category.

Item	Category
Antenna assembly	Audio/video devices
ABS steel pin	Braking system
Back seat	Car seat
Door switch	Electrical switches
Distributor cap	Engine components and parts
Oil gasket	Engine oil systems

Figure 8a–c are car spare parts shop simulation result of EAMDSP, CDSSSD and MDMSMD, respectively which show the shortest path among six items. Table 4 shows the output summary data for the six-item search. EAMDSP for the shortest path between the six items visited 129 nodes and the total weight was 131. However, when using CDSSSD and MDMSMD for the same items, the visited nodes were 220 and 175 and the total cost was 242 and 179, respectively. Therefore, EAMDSP is efficient in terms of visited nodes and total weight. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 8. Simulation on a car spare parts shop scenario using EAMDSP, CDSSSD, and MDMSMD for a six-item search (output scenario).

Table 4. Summary data of EAMDSP, CDSSSD, and MDMSMD for a six-item search.

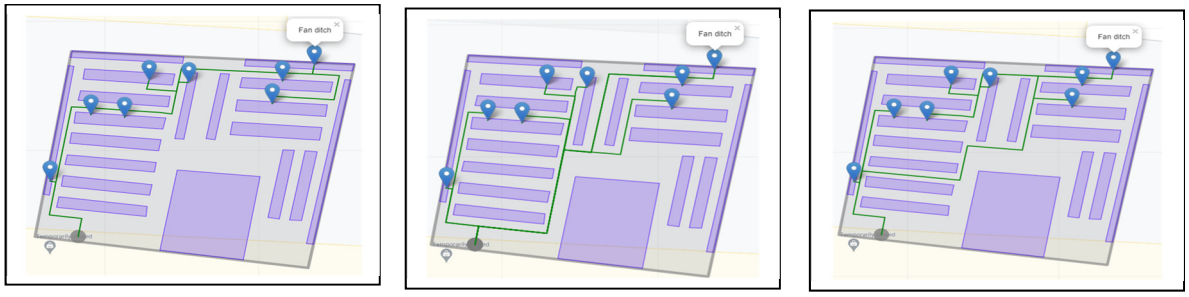
SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	129	131
2	CDSSSD	220	242
3	MDMSMD	175	179

Table 5 represents the input values of the eight items. These input items were used for the same simulation layout using three algorithms CDSSSD, MDMSMD, and EAMDSP.

Table 5. (Input) Selecting eight items and category.

Item	Category
Radio and media player	Audio/video devices
Cowl screen	Body components, including trim
Adjusting mechanism (adjuster star wheel)	Braking system
Dashcam	Cameras
Bench seat	Car seat
Central locking	Doors
Fan ditch	Electrical switches
Engine compartment harness	Wiring harnesses

Figure 9a–c are car spare parts shop simulation result of respectively EAMDSP, CDSSSD and MDMSMD which show the shortest path among eight items. Table 6 shows the output summary data for the eight items searched. When using EAMDSP for the shortest path between the above eight items, 119 nodes were visited, and the total weight or cost was 121. However, when using CDSSSD and MDMSMD for the same items, the visited nodes were 271 and 227 and the total cost was 297 and 239, respectively. So, EAMDSP is efficient for visited nodes and the total weight or cost. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 9. Simulation on a car spare parts shop scenario using EAMDSP, CDSSSD, and MDMSMD for an eight-item search (output scenario).

Table 6. Summary data of EAMDSP, CDSSSD, and MDMSMD for an eight-item search.

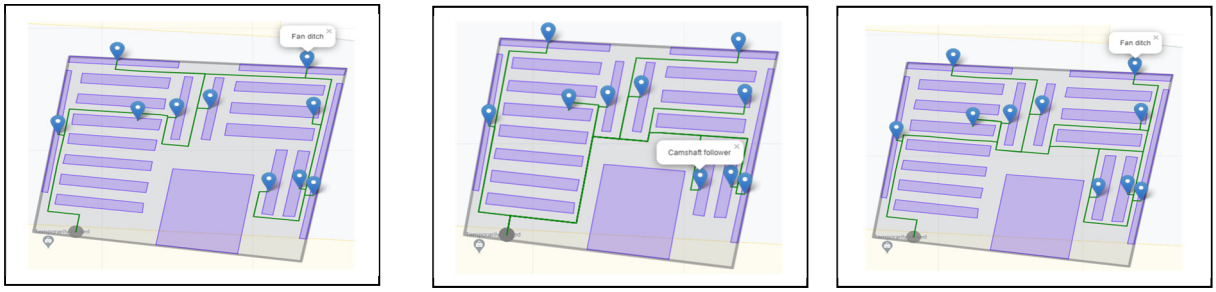
SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	119	121
2	CDSSSD	271	297
3	MDMSMD	227	239

Table 7 represents the input value of ten items. These input items were used for the same simulation layout of three algorithms CDSSSD, MDMSMD, and EAMDSP.

Table 7. (Input) Selecting ten items and category.

Item	Category
Radiator core support	Body components
Brake pad	Braking system
Fan ditch	Electrical switches
Camshaft follower	Engine components and parts
PCV valve	Engine components and parts
Oil suction filter	Engine oil systems
Battery box	Low voltage electrical supply system
Alarm and siren	Miscellaneous
Wiring connector	Miscellaneous
Glowplug	Starting system

Figure 10a–c are car spare parts shop simulation result of EAMDSP, CDSSSD and MDMSMD respectively, which show the shortest path among ten items. Table 8 shows the output summary data for the ten-item search. When using EAMDSP for the shortest path between the above 10 items, 164 nodes were visited and the total weight or cost was 170. However, when using CDSSSD and MDMSMD for the same items, the visited nodes were 377 and 227, and the total cost was 414 and 236, respectively. Therefore, EAMDSP is efficient for visited nodes and total weight or cost. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 10. Simulation on a car spare parts shop scenario using EAMDSP, CDSSSD, and MDMSMD for a ten-item search (output scenario).

Table 8. Summary data of EAMDSP, CDSSSD, and MDMSMD for a ten-item search.

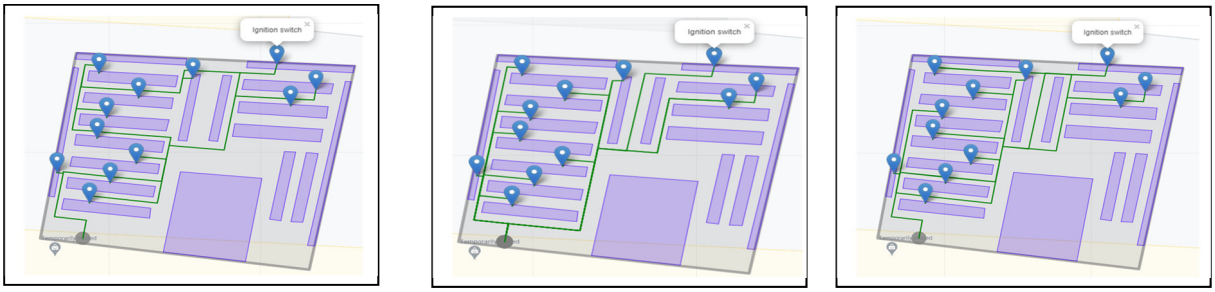
SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	164	170
2	CDSSSD	377	414
3	MDMSMD	227	236

Table 9 represents the input value of twelve items. These input items were used for the same simulation layout of three algorithms CDSSSD, MDMSMD, and EAMDSP.

Figure 11a–c are car spare parts shop simulation result of EAMDSP, CDSSSD and MDMSMD respectively, which show the shortest path among twelve items. Table 10 shows the output summary data for the twelve items searched. When using EAMDSP for the shortest path between the above twelve items, 213 nodes were visited and the total weight or cost was 220. However, when using CDSSSD and MDMSMD for the same items, the visited nodes were 378 and 315 and the total cost was 408 and 327, respectively. Therefore, EAMDSP is efficient for visited nodes and total weight or cost. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.

Table 9. (Input) Selecting twelve items and category.

Item	Category
Tuner	Audio/video devices
Fascia rear and support	Body components
Brake backing plate	Braking system
Front seat	Car seat
Front Right Outer door handle	Doors
Ignition switch	Electrical switches
Fuel gauge	Gauges and meters
Ignition magneto	Ignition system
Side lighting	Lighting and signaling system
Coolant temperature sensor	Sensors
Sunroof glass	Windows
Air conditioning harness	Wiring harnesses



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 11. Simulation on a car spare parts shop scenario using EAMDSP, CDSSSD, and MDMSMD for a twelve-item search (output scenario).

Table 10. Summary data of EAMDSP, CDSSSD, and MDMSMD for a twelve-item search.

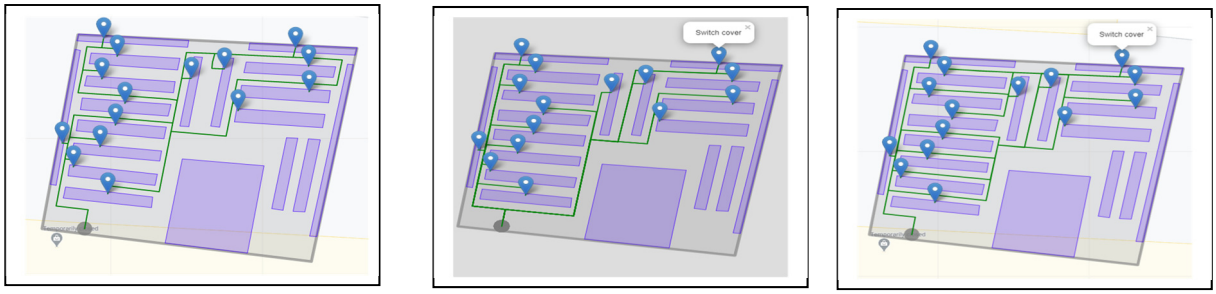
SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	213	220
2	CDSSSD	378	408
3	MDMSMD	315	327

Table 11 represents the input values of the selected fifteen items. These input items were used for the same simulation layout of three algorithms CDSSSD, MDMSMD, and EAMDSP.

Table 11. (Input) Selecting fifteen items and categories.

Item	Category
Front fascia and header panel	Body components
Brake disc	Braking system
Brake booster hose	Braking system
Backup camera	Cameras
Headrest	Car seat
Rear left side outer door handle	Doors
Switch cover	Electrical switches
Odometer	Gauges and meters
Distributor	Ignition system
Engine bay lighting	Lighting and signaling system
Shift improver	Miscellaneous
Knock sensor	Sensors
Starter drive	Starting system
Front left side door glass	Windows
Floor harness	Wiring harnesses

Figure 12a–c are car spare parts shop simulation result of EAMDSP, CDSSSD and MDMSMD respectively, which show the shortest path among fifteen items. Table 12 shows the output summary data for the fifteen-item search. When using EAMDSP for the shortest path among the above fifteen items, 225 nodes were visited and the total weight or cost was 233. However, when using CDSSSD and MDMSMD for the same items, the visited nodes were 487 and 393 and the total cost was 525 and 409, respectively. Therefore, our proposed shortest path algorithm EAMDSP is efficient for visited nodes and total weight or cost. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 12. Simulation on a car spare parts shop scenario using EAMDSP, CDSSSD, and MDMSMD for a fifteen-item search (output scenario).

Table 12. Summary data of EAMDSP, CDSSSD, and MDMSMD for a fifteen-item search.

SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	225	233
2	CDSSSD	487	525
3	MDMSMD	393	409

Figure 13 shows the cost-wise comparison among EAMDSP, CDSSSD and MDMSMD as per Table 13.

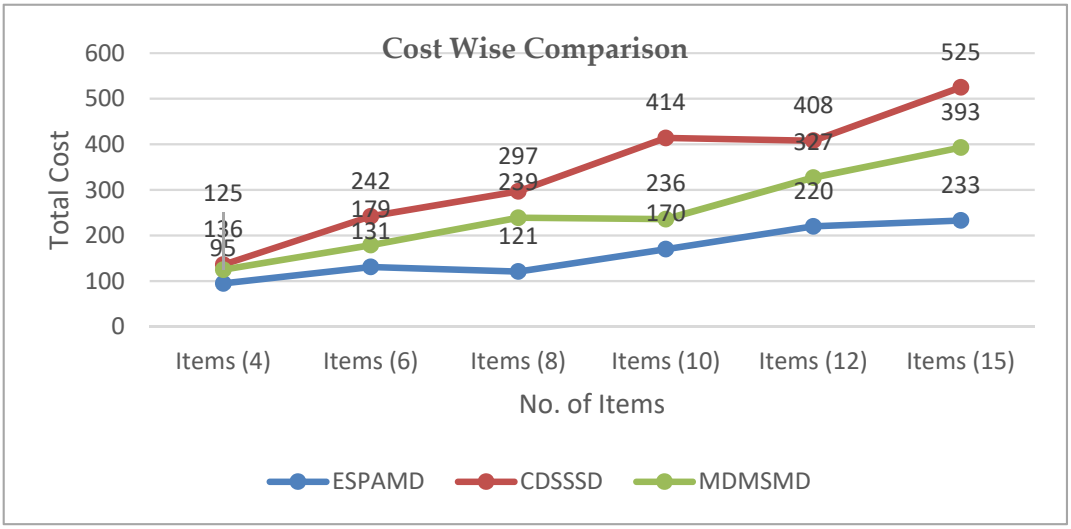


Figure 13. Cost-wise comparison among EAMDSP, CDSSSD, and MDMSMD.

Table 13. Total cost-wise comparison among EAMDSP, CDSSSD, and MDMSMD.

SL No.	Algorithm Name	Items (4)	Items (6)	Items (8)	Items (10)	Items (12)	Items (15)
1	EAMDSP	95	131	121	170	220	233
2	CDSSSD	136	242	297	414	408	525
3	MDMSMD	125	179	239	236	327	409

Figure 14 shows the visited-wise comparison among EAMDSP, CDSSSD and MDMSMD as per Table 14.

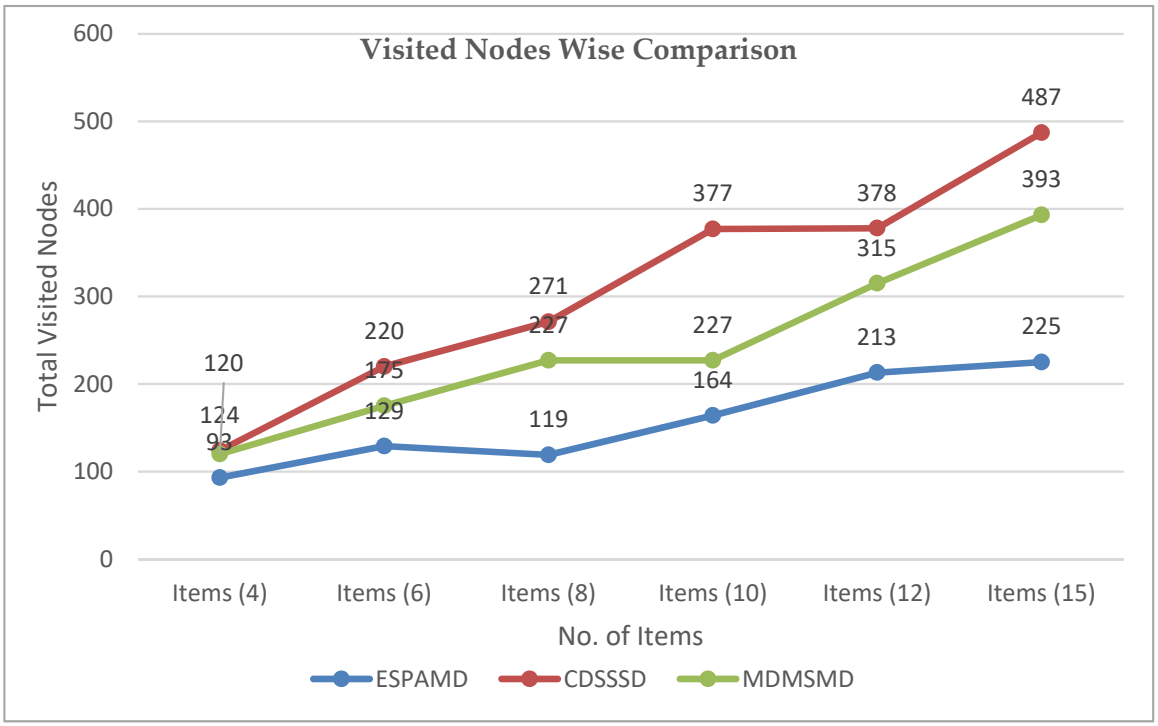


Figure 14. Visited nodes-wise comparison between EAMDSP, CDSSSD, and MDMSMD.

Table 14. Total visited node-wise comparison among EAMDSP, CDSSSD, and MDMSMD.

SL No.	Algorithm Name	Items (4)	Items (6)	Items (8)	Items (10)	Items (12)	Items (15)
1	EAMDSP	93	129	119	164	213	225
2	CDSSSD	124	220	271	377	378	487
3	MDMSMD	120	175	227	227	315	393

In Figure 15, we present another scenario similar to a supermarket simulation scenario, the simulations were run in three unique phases as a five, eight, and eleven-item searching layout. Three shortest path algorithms CDSSSD, MDMSMD, and our proposed EAMDSP were used in this scenario's different phases for getting the different results and comparing results shown in the layout simulation. Summary data is presented in tables and graphs.

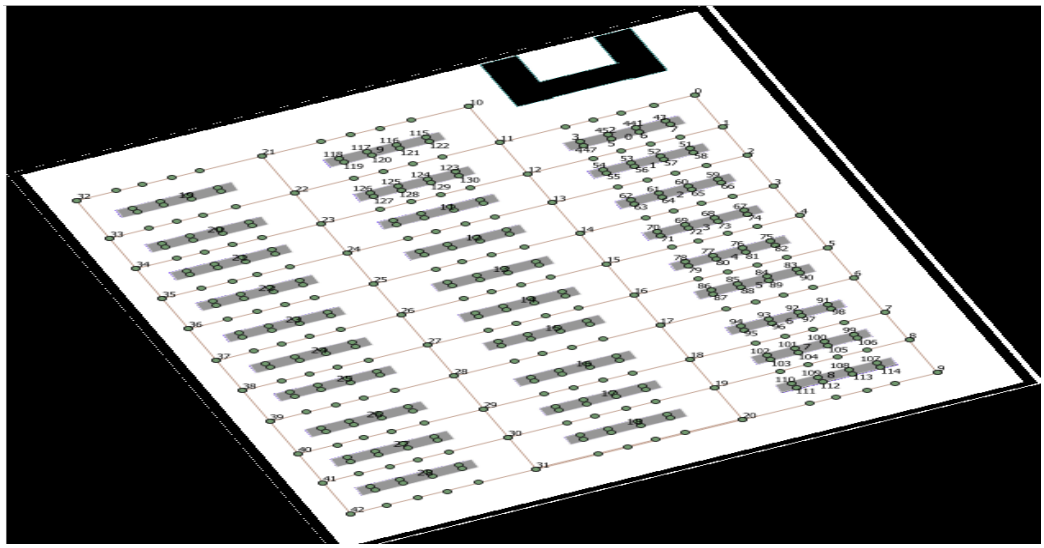


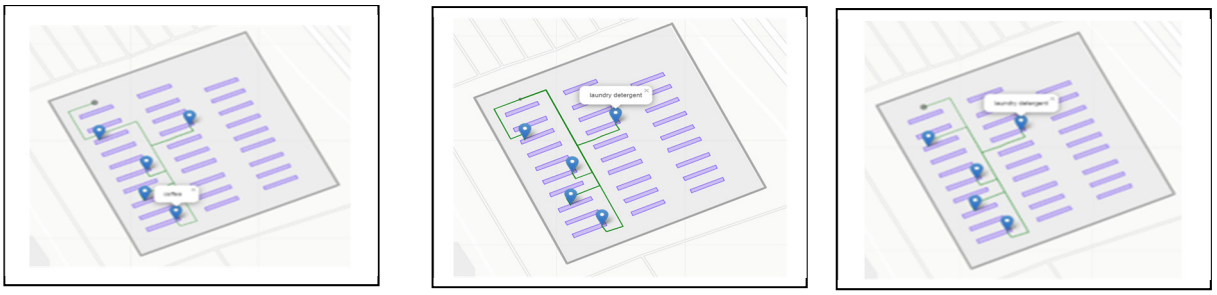
Figure 15. Drawing layout in the supermarket with nodes.

Table 15 represents the input value, which was randomly selected for five items. These input items were used for the same simulation layout of three algorithms CDSSSD, MDMSMD, and EAMDSP.

Table 15. (Input) Selecting five items and categories.

Item	Category
Coffee	Beverages
Dinner rolls	Bread/Bakery
Vegetables	Canned/Jarred goods
Laundry detergent	Cleaners
Yogurt	Dairy

Figure 16a–c are supermarket simulation result of respectively EAMDSP, CDSSSD and MDMSMD which show the shortest path among five items. Table 16 shows the output summary data for the five-item search. When using EAMDSP for the shortest path between the above the five items, 45 nodes were visited and the total weight was 54. However, when using CDSSSD and MDMSMD for the same items, the visited nodes were 60 and 52 and the total cost was 84 and 69, respectively. Therefore, our proposed shortest path algorithm EAMDSP is efficient for visited nodes and total cost. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 16. Simulation on supermarket scenario using EAMDSP, CDSSSD, and MDMSMD for a five-item search (output scenario).

Table 16. The output summary data of EAMDSP, CDSSSD, and MDMSMD for a five-item search.

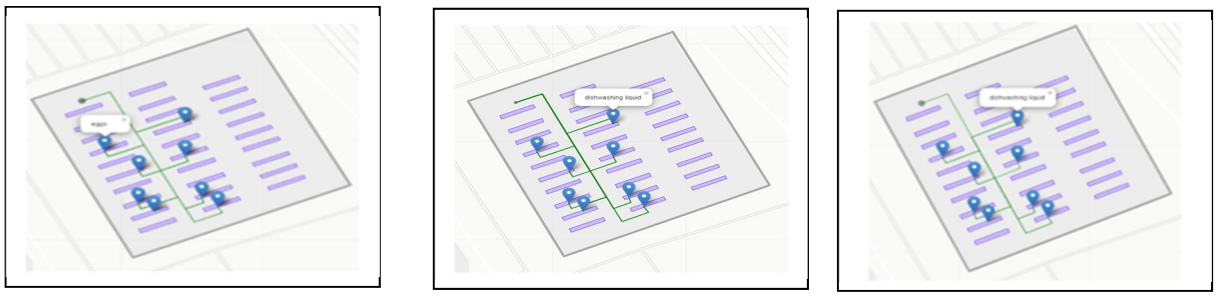
SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	45	54
2	CDSSSD	60	84
3	MDMSMD	52	69

Table 17 represents the input values of the selected eight items. These input items were used for the same simulation layout of three algorithms CDSSSD, MDMSMD, and EAMDSP.

Table 17. (Input) Selecting eight items and categories.

Item	Category
Soda	Beverages
Sandwich loaves	Bread/Bakery
Vegetables	Canned/Jarred goods
Dishwashing liquid	Cleaners
Eggs	Dairy
Mixes	Dry/Baking goods
Vegetables	Frozen foods
Pork	Meat

Figure 17a–c are supermarket simulation result of respectively EAMDSP, CDSSSD and MDMSMD which show the shortest path among eight items. Table 18 shows the output summary data for the eight-item search. When using EAMDSP for the shortest path between the above eight items, 58 nodes were visited and the total weight was 67. However, when using CDSSSD and MDMSMD for the same items, visited nodes were 104 and 72 and the total cost was 150 and 95, respectively. Therefore, EAMDSP is efficient for visited nodes and total weight. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 17. Simulation on supermarket scenario using EAMDSP, CDSSSD, and MDMSMD for an eight-item search (output scenario).

Table 18. The output summary data of EAMDSP, CDSSSD, and MDMSMD for an eight-item search.

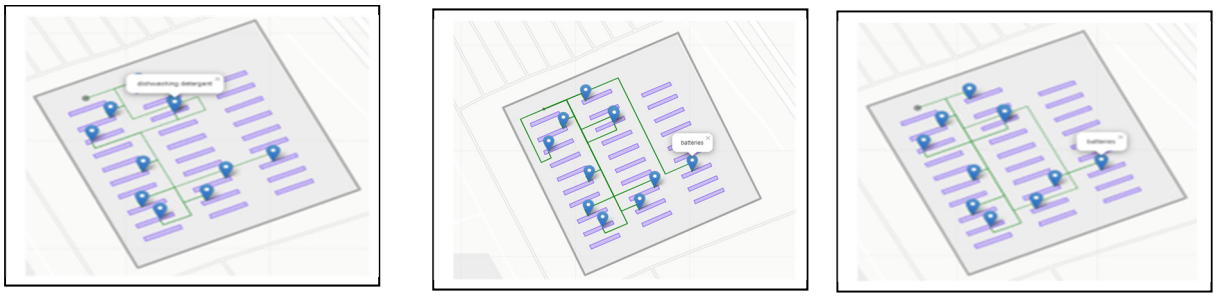
SL No.	Algorithm Name	Total Visited Nodes	Total Cost
1	EAMDSP	58	67
2	CDSSSD	104	150
3	MDMSMD	72	95

Table 19 represents the input values of the selected eleven items. These input items were used for the same simulation layout of three algorithms CDSSSD, MDMSMD, and EAMDSP.

Table 19. (Input) Selecting eleven items and categories.

Item	Category
Juice	Beverages
Dinner rolls	Bread/Bakery
Spaghetti sauce	Canned/Jarred goods
Dishwashing detergent	Cleaners
Cheeses	Dairy
Flour	Dry/Baking goods
Individual meals	Frozen foods
Lunch meat	Meat
Batteries	Others
Paper towels	Paper goods
Shampoo	Personal cares

Figure 18a–c are supermarket simulation result of respectively EAMDSP, CDSSSD and MDMSMD which show the shortest path among eleven items. Table 20 shows the output summary data for an eleven-item search. When using EAMDSP for the shortest path between the above eleven items, 78 nodes were visited and the total weight was 88. However, when using CDSSSD and MDMSMD for the same items the visited nodes were 124 and 101 and the total cost was 165 and 132 respectively. Therefore, EAMDSP is efficient for visited nodes and total weight. Finally, total visited nodes, or total cost can be expressed as EAMDSP < MDMSMD < CDSSSD.



(**a**) EAMDSP (**b**) CDSSSD (**c**) MDMSMD

Figure 18. Simulation on supermarket scenario using EAMDSP, CDSSSD, and MDMSMD for an eleven-item search (output scenario).

Table 20. Summary data of EAMDSP, CDSSSD, and MDMSMD for eleven items search.

SL No.	Algorithm Name	Total Nodes	Total Cost
1	EAMDSP	78	88
2	CDSSSD	124	165
3	MDMSMD	101	132

Figure 19 shows the cost-wise comparison among EAMDSP, CDSSSD and MDMSMD as per Table 21.

Table 21. Total cost-wise comparison between EAMDSP, CDSSSD, and MDMSMD.

SL No.	Algorithm Name	Items (5)	Items (8)	Items (11)
1	EAMDSP	54	67	88
2	CDSSSD	84	150	165
3	MDMSMD	69	95	132

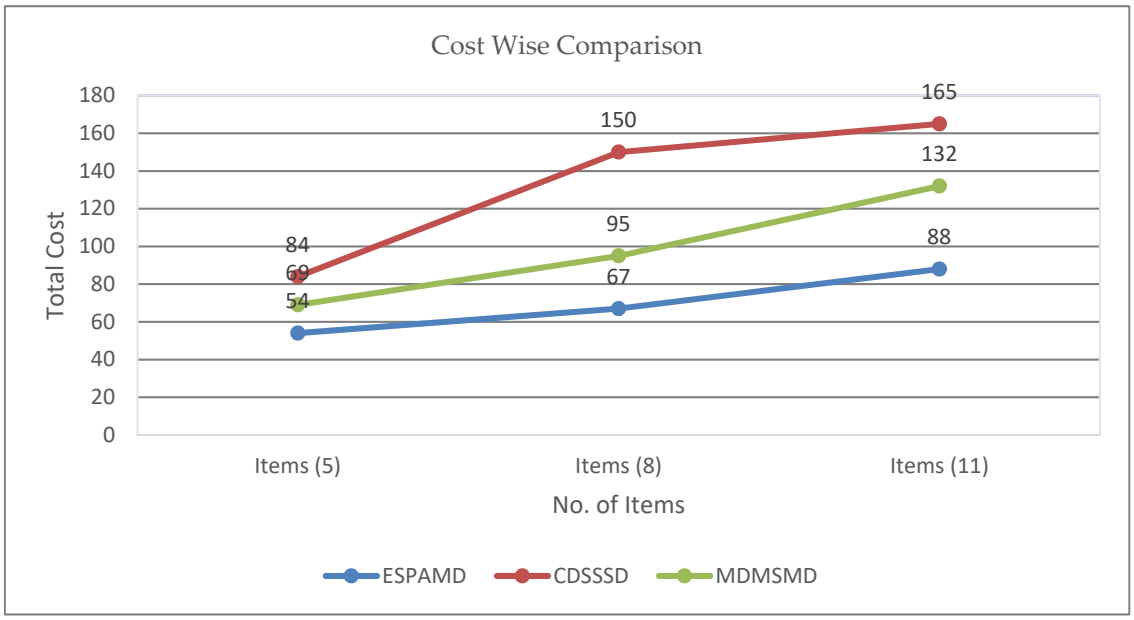


Figure 19. Cost-wise comparison between EAMDSP, CDSSSD, and MDMSMD.

Figure 20 shows the visited-wise comparison among EAMDSP, CDSSSD and MDMSMD as per Table 22.

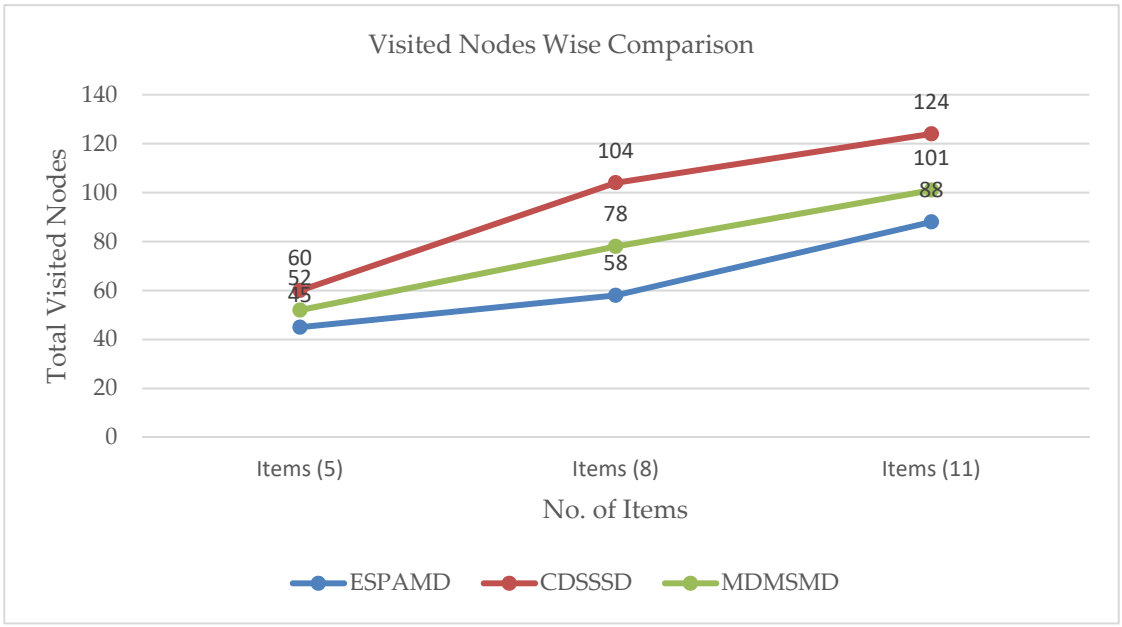


Figure 20. Visited node-wise comparison between EAMDSP, CDSSSD, and MDMSMD.

Table 22. Total visited node-wise comparison between EAMDSP, CDSSSD, and MDMSMD.

SL No.	Algorithm Name	Items (5)	Items (8)	Items (11)
1	EAMDSP	45	58	88
2	CDSSSD	60	104	124
3	MDMSMD	52	78	101

Finally, based on Figures 13, 14, 19 and 20, and Tables 13, 14, 21 and 22 it is reported that the proposed EAMDSP demonstrates the outstanding performance on cost and number of visited nodes compared to CDSSSD and MDMSMD.

5. Conclusions

The proposed EAMDSP algorithm was developed and implemented in indoor applications and found to be more efficient for a multi-destination search. It can be used on any type of road network with multi-destinations for shortest path searching. In this paper, we evaluated the EAMDSP by comparing it with CDSSSD and MDMSMD using two different simulations of several items of searching. For a search with a large number of products, EAMDSP demonstrated an outstanding performance compared to CDSSSD and MDMSMD, based on the total visited nodes and weights. The output of these simulations based on visited nodes and the minimum sum of weights is presented in tables and graphs. This research can be a useful guide for indoor shortest-path researchers. In future work, we might extend the application of EAMDSP from an indoor to an outdoor environment.

Author Contributions: Conceptualization, M.A., F.H. and T.K.G.; methodology, M.A., F.H. and T.K.G.; validation, F.H. and T.K.G.; formal analysis, M.A. and F.H.; writing—original draft preparation, M.A. and F.H.; writing—review and editing, F.H., C.P.T. and S.S.; visualization, M.A.B. and S.A.; supervision, T.K.G. and S.S.; project administration, F.H.; funding acquisition, T.K.G., A.A. and H.-Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research is also funded by the "Indoor Internet of Things (IOT) Tracking Algorithm Development based on Radio Signal Characterization" and "Mobile IOT: Location Aware" project bearing grant no. [FRGS/1/2018/TK08/MMU/02/1] and [MMUE/180025].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: Special thanks to MMU RMC for providing comprehensive financial assistance to this research.

Conflicts of Interest: The authors declare no conflict of interest. The funding sponsors had no role in the design of the study, in the collection, analyses, or interpretation of data, in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Result Detail Data Tables

Table A1. Simulation detail data for car spare parts scenario using EAMDSP for four items search.

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name	
1	342	102.27257	2.2528471	342	384	0		1st source
2	208	102.27257	2.252859	342	384	2		
3	209	102.27257	2.2528595	342	384	1		
4	210	102.27257	2.2528599	342	384	1		
5	211	102.27256	2.2528603	342	384	1		
6	212	102.27256	2.2528607	342	384	1		
7	213	102.27256	2.2528615	342	384	1		
8	214	102.27256	2.2528757	342	384	2		
9	215	102.27256	2.2528774	342	384	1		
10	232	102.27256	2.2528798	342	384	1		
11	384	102.27256	2.2528801	342	384	1	Bumper	1st destination
12	232	102.27256	2.2528798	384	472	1		
13	233	102.27256	2.2528833	384	472	1		
14	234	102.27256	2.2528868	384	472	1		
15	235	102.27256	2.2528903	384	472	1		

Table A1. *Cont.*

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name	
16	236	102.27256	2.2528911	384	472	1		
17	253	102.27256	2.2528938	384	472	1		
18	254	102.27256	2.2528971	384	472	1		
19	255	102.27256	2.2529008	384	472	1		
20	256	102.27256	2.2529042	384	472	1		
21	257	102.27256	2.2529051	384	472	1		
22	274	102.27256	2.2529077	384	472	1		
23	275	102.27256	2.2529111	384	472	1		
24	276	102.27256	2.2529148	384	472	1		
25	277	102.27257	2.2529182	384	472	1		
26	278	102.27257	2.2529192	384	472	1		
27	295	102.27257	2.2529217	384	472	1		
28	296	102.27257	2.2529252	384	472	1		
29	297	102.27257	2.2529287	384	472	1		
30	298	102.27257	2.2529323	384	472	1		
31	299	102.27257	2.2529332	384	472	1		
32	300	102.27257	2.2529328	384	472	1		
33	301	102.27258	2.2529326	384	472	1		
34	472	102.27258	2.252928	384	472	1	Antenna cable	2nd destination
35	301	102.27258	2.2529326	472	683	1		
36	302	102.27258	2.2529323	472	683	1		
37	303	102.27258	2.252932	472	683	1		
38	304	102.27259	2.2529318	472	683	1		
39	305	102.27259	2.2529315	472	683	1		
40	306	102.27259	2.2529314	472	683	1		
41	307	102.2726	2.2529311	472	683	1		
42	308	102.2726	2.2529308	472	683	1		
43	309	102.2726	2.2529306	472	683	1		
44	310	102.2726	2.2529303	472	683	1		
45	311	102.27261	2.2529301	472	683	1		
46	312	102.27261	2.2529298	472	683	1		
47	313	102.27261	2.2529295	472	683	1		
48	314	102.27262	2.2529293	472	683	1		
49	315	102.27262	2.2529291	472	683	1		
50	186	102.27262	2.2529285	472	683	1		
51	185	102.27262	2.252931	472	683	1		
52	184	102.27263	2.2529348	472	683	1		
53	183	102.27263	2.2529386	472	683	1		
54	182	102.27263	2.2529423	472	683	1		
55	181	102.27263	2.2529435	472	683	1		
56	180	102.27263	2.252946	472	683	1		
57	179	102.27263	2.2529495	472	683	1		
58	178	102.27263	2.2529534	472	683	1		
59	22	102.27263	2.2529585	472	683	1		
60	23	102.27265	2.2529578	472	683	1		
61	24	102.27266	2.2529569	472	683	2		
62	25	102.27266	2.2529567	472	683	1		
63	26	102.27267	2.2529567	472	683	1		
64	27	102.27267	2.2529564	472	683	1		
65	28	102.27267	2.2529563	472	683	1		
66	29	102.27268	2.2529562	472	683	1		
67	30	102.27268	2.2529559	472	683	1		
68	31	102.27268	2.2529558	472	683	1		
69	32	102.27269	2.2529557	472	683	1		
70	33	102.27269	2.2529554	472	683	1		
71	683	102.27269	2.2529501	472	683	1	Bucket seat	3rd destination

Table A1. *Cont.*

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name	
72	33	102.27269	2.2529554	683	713	1		
73	32	102.27269	2.2529557	683	713	1		
74	31	102.27268	2.2529558	683	713	1		
75	30	102.27268	2.2529559	683	713	1		
76	29	102.27268	2.2529562	683	713	1		
77	28	102.27267	2.2529563	683	713	1		
78	27	102.27267	2.2529564	683	713	1		
79	26	102.27267	2.2529567	683	713	1		
80	25	102.27266	2.2529567	683	713	1		
81	24	102.27266	2.2529569	683	713	1		
82	164	102.27266	2.2529515	683	713	1		
83	163	102.27266	2.2529477	683	713	1		
84	162	102.27266	2.252944	683	713	1		
85	145	102.27266	2.2529414	683	713	1		
86	146	102.27266	2.2529411	683	713	1		
87	147	102.27267	2.2529409	683	713	1		
88	148	102.27267	2.2529407	683	713	1		
89	149	102.27267	2.2529405	683	713	1		
90	150	102.27267	2.2529404	683	713	1		
91	151	102.27268	2.2529402	683	713	1		
92	152	102.27268	2.25294	683	713	1		
93	713	102.27268	2.2529351	683	713	1	Adjusting mechanism (adjuster star wheel)	4th destination

Total visited nodes = 93, total cost = 95.

Table A2. Simulation detail data for car spare parts Scenario using CDSSSD for four items search.

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name
1	342	102.27257	2.2528471	342	472	0	
2	208	102.27257	2.252859	342	472	2	
3	209	102.27257	2.2528595	342	472	1	
4	210	102.27257	2.2528599	342	472	1	
5	211	102.27256	2.2528603	342	472	1	
6	212	102.27256	2.2528607	342	472	1	
7	213	102.27256	2.2528615	342	472	1	
8	214	102.27256	2.2528757	342	472	2	
9	215	102.27256	2.2528774	342	472	1	
10	232	102.27256	2.2528798	342	472	1	
11	233	102.27256	2.2528833	342	472	1	
12	234	102.27256	2.2528868	342	472	1	
13	235	102.27256	2.2528903	342	472	1	
14	236	102.27256	2.2528911	342	472	1	
15	253	102.27256	2.2528938	342	472	1	
16	254	102.27256	2.2528971	342	472	1	
17	255	102.27256	2.2529008	342	472	1	
18	256	102.27256	2.2529042	342	472	1	
19	257	102.27256	2.2529051	342	472	1	
20	274	102.27256	2.2529077	342	472	1	
21	275	102.27256	2.2529111	342	472	1	
22	276	102.27256	2.2529148	342	472	1	
23	277	102.27257	2.2529182	342	472	1	
24	278	102.27257	2.2529192	342	472	1	

Table A2. *Cont.*

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name
25	295	102.27257	2.2529217	342	472	1	
26	296	102.27257	2.2529252	342	472	1	
27	297	102.27257	2.2529287	342	472	1	
28	298	102.27257	2.2529323	342	472	1	
29	299	102.27257	2.2529332	342	472	1	
30	300	102.27257	2.2529328	342	472	1	
31	301	102.27258	2.2529326	342	472	1	
32	472	102.27258	2.252928	342	472	1	Antenna cable
33	342	102.27257	2.2528471	342	384	0	
34	208	102.27257	2.252859	342	384	2	
35	209	102.27257	2.2528595	342	384	1	
36	210	102.27257	2.2528599	342	384	1	
37	211	102.27256	2.2528603	342	384	1	
38	212	102.27256	2.2528607	342	384	1	
39	213	102.27256	2.2528615	342	384	1	
40	214	102.27256	2.2528757	342	384	2	
41	215	102.27256	2.2528774	342	384	1	
42	232	102.27256	2.2528798	342	384	1	
43	384	102.27256	2.2528801	342	384	1	Bumper
44	342	102.27257	2.2528471	342	713	0	
45	208	102.27257	2.252859	342	713	2	
46	207	102.27258	2.2528588	342	713	1	
47	206	102.27258	2.2528583	342	713	1	
48	205	102.27258	2.252858	342	713	1	
49	204	102.27259	2.2528575	342	713	1	
50	203	102.27259	2.2528572	342	713	1	
51	202	102.27259	2.2528567	342	713	1	
52	201	102.27259	2.2528565	342	713	1	
53	200	102.2726	2.252856	342	713	1	
54	199	102.2726	2.2528555	342	713	1	
55	198	102.2726	2.2528553	342	713	1	
56	197	102.27261	2.2528548	342	713	1	
57	196	102.27261	2.2528543	342	713	1	
58	195	102.27261	2.2528706	342	713	2	
59	194	102.27262	2.2528847	342	713	2	
60	193	102.27262	2.2528996	342	713	2	
61	192	102.27262	2.2529086	342	713	1	
62	166	102.27264	2.2529071	342	713	2	
63	165	102.27265	2.2529057	342	713	2	
64	119	102.27265	2.2529111	342	713	1	
65	120	102.27265	2.2529138	342	713	1	
66	121	102.27265	2.2529178	342	713	1	
67	122	102.27265	2.2529216	342	713	1	
68	123	102.27266	2.2529251	342	713	1	
69	124	102.27266	2.2529265	342	713	1	
70	141	102.27266	2.2529289	342	713	1	
71	142	102.27266	2.2529325	342	713	1	
72	143	102.27266	2.2529367	342	713	1	
73	144	102.27266	2.25294	342	713	1	
74	145	102.27266	2.2529414	342	713	1	
75	146	102.27266	2.2529411	342	713	1	
76	147	102.27267	2.2529409	342	713	1	
77	148	102.27267	2.2529407	342	713	1	
78	149	102.27267	2.2529405	342	713	1	
79	150	102.27267	2.2529404	342	713	1	
80	151	102.27268	2.2529402	342	713	1	
81	152	102.27268	2.25294	342	713	1	

Table A2. *Cont.*

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name
82	713	102.27268	2.2529351	342	713	1	Adjusting mechanism (adjuster star wheel)
83	342	102.27257	2.2528471	342	683	0	
84	208	102.27257	2.252859	342	683	2	
85	207	102.27258	2.2528588	342	683	1	
86	206	102.27258	2.2528583	342	683	1	
87	205	102.27258	2.252858	342	683	1	
88	204	102.27259	2.2528575	342	683	1	
89	203	102.27259	2.2528572	342	683	1	
90	202	102.27259	2.2528567	342	683	1	
91	201	102.27259	2.2528565	342	683	1	
92	200	102.2726	2.252856	342	683	1	
93	199	102.2726	2.2528555	342	683	1	
94	198	102.2726	2.2528553	342	683	1	
95	197	102.27261	2.2528548	342	683	1	
96	196	102.27261	2.2528543	342	683	1	
97	195	102.27261	2.2528706	342	683	2	
98	194	102.27262	2.2528847	342	683	2	
99	193	102.27262	2.2528996	342	683	2	
100	192	102.27262	2.2529086	342	683	1	
101	166	102.27264	2.2529071	342	683	2	
102	167	102.27264	2.2529153	342	683	1	
103	168	102.27264	2.252919	342	683	1	
104	169	102.27264	2.2529226	342	683	1	
105	170	102.27264	2.2529264	342	683	1	
106	171	102.27264	2.2529298	342	683	1	
107	172	102.27264	2.2529338	342	683	1	
108	173	102.27264	2.2529376	342	683	1	
109	174	102.27264	2.2529413	342	683	1	
110	175	102.27264	2.2529451	342	683	1	
111	176	102.27264	2.2529486	342	683	1	
112	177	102.27264	2.2529524	342	683	1	
113	23	102.27265	2.2529578	342	683	1	
114	24	102.27266	2.2529569	342	683	2	
115	25	102.27266	2.2529567	342	683	1	
116	26	102.27267	2.2529567	342	683	1	
117	27	102.27267	2.2529564	342	683	1	
118	28	102.27267	2.2529563	342	683	1	
119	29	102.27268	2.2529562	342	683	1	
120	30	102.27268	2.2529559	342	683	1	
121	31	102.27268	2.2529558	342	683	1	
122	32	102.27269	2.2529557	342	683	1	
123	33	102.27269	2.2529554	342	683	1	
124	683	102.27269	2.2529501	342	683	1	Bucket seat

Total visited nodes = 124, total cost = 136.

Table A3. Simulation detail data for car spare parts scenario using MDMSMD for four items search.

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name	
1	342	102.27257	2.2528471	342	472	0		1st source
2	208	102.27257	2.252859	342	472	2		
3	209	102.27257	2.2528595	342	472	1		
4	210	102.27257	2.2528599	342	472	1		
5	211	102.27256	2.2528603	342	472	1		

Table A3. *Cont.*

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name	
6	212	102.27256	2.2528607	342	472	1		
7	213	102.27256	2.2528615	342	472	1		
8	214	102.27256	2.2528757	342	472	2		
9	215	102.27256	2.2528774	342	472	1		
10	232	102.27256	2.2528798	342	472	1		
11	233	102.27256	2.2528833	342	472	1		
12	234	102.27256	2.2528868	342	472	1		
13	235	102.27256	2.2528903	342	472	1		
14	236	102.27256	2.2528911	342	472	1		
15	253	102.27256	2.2528938	342	472	1		
16	254	102.27256	2.2528971	342	472	1		
17	255	102.27256	2.2529008	342	472	1		
18	256	102.27256	2.2529042	342	472	1		
19	257	102.27256	2.2529051	342	472	1		
20	274	102.27256	2.2529077	342	472	1		
21	275	102.27256	2.2529111	342	472	1		
22	276	102.27256	2.2529148	342	472	1		
23	277	102.27257	2.2529182	342	472	1		
24	278	102.27257	2.2529192	342	472	1		
25	295	102.27257	2.2529217	342	472	1		
26	296	102.27257	2.2529252	342	472	1		
27	297	102.27257	2.2529287	342	472	1		
28	298	102.27257	2.2529323	342	472	1		
29	299	102.27257	2.2529332	342	472	1		
30	300	102.27257	2.2529328	342	472	1		
31	301	102.27258	2.2529326	342	472	1		
32	472	102.27258	2.252928	342	472	1	Antenna cable	1st destination
33	301	102.27258	2.2529326	472	384	1		
34	300	102.27257	2.2529328	472	384	1		
35	299	102.27257	2.2529332	472	384	1		
36	298	102.27257	2.2529323	472	384	1		
37	297	102.27257	2.2529287	472	384	1		
38	296	102.27257	2.2529252	472	384	1		
39	295	102.27257	2.2529217	472	384	1		
40	278	102.27257	2.2529192	472	384	1		
41	277	102.27257	2.2529182	472	384	1		
42	276	102.27256	2.2529148	472	384	1		
43	275	102.27256	2.2529111	472	384	1		
44	274	102.27256	2.2529077	472	384	1		
45	257	102.27256	2.2529051	472	384	1		
46	256	102.27256	2.2529042	472	384	1		
47	255	102.27256	2.2529008	472	384	1		
48	254	102.27256	2.2528971	472	384	1		
49	253	102.27256	2.2528938	472	384	1		
50	236	102.27256	2.2528911	472	384	1		
51	235	102.27256	2.2528903	472	384	1		
52	234	102.27256	2.2528868	472	384	1		
53	233	102.27256	2.2528833	472	384	1		
54	232	102.27256	2.2528798	472	384	1		
55	384	102.27256	2.2528801	472	384	1	Bumper	2nd destination
56	232	102.27256	2.2528798	384	713	1		
57	215	102.27256	2.2528774	384	713	1		
58	216	102.27256	2.2528767	384	713	1		
59	217	102.27257	2.2528764	384	713	1		
60	218	102.27257	2.2528761	384	713	1		

Table A3. *Cont.*

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name	
61	219	102.27257	2.2528755	384	713	1		
62	220	102.27258	2.2528752	384	713	1		
63	221	102.27258	2.2528749	384	713	1		
64	222	102.27258	2.2528744	384	713	1		
65	223	102.27259	2.2528741	384	713	1		
66	224	102.27259	2.2528738	384	713	1		
67	225	102.27259	2.2528735	384	713	1		
68	226	102.27259	2.252873	384	713	1		
69	227	102.2726	2.2528727	384	713	1		
70	228	102.2726	2.2528722	384	713	1		
71	229	102.2726	2.2528718	384	713	1		
72	230	102.27261	2.2528715	384	713	1		
73	231	102.27261	2.2528712	384	713	1		
74	195	102.27261	2.2528706	384	713	1		
75	194	102.27262	2.2528847	384	713	2		
76	193	102.27262	2.2528996	384	713	2		
77	192	102.27262	2.2529086	384	713	1		
78	166	102.27264	2.2529071	384	713	2		
79	165	102.27265	2.2529057	384	713	2		
80	119	102.27265	2.2529111	384	713	1		
81	120	102.27265	2.2529138	384	713	1		
82	121	102.27265	2.2529178	384	713	1		
83	122	102.27265	2.2529216	384	713	1		
84	123	102.27266	2.2529251	384	713	1		
85	124	102.27266	2.2529265	384	713	1		
86	141	102.27266	2.2529289	384	713	1		
87	142	102.27266	2.2529325	384	713	1		
88	143	102.27266	2.2529367	384	713	1		
89	144	102.27266	2.25294	384	713	1		
90	145	102.27266	2.2529414	384	713	1		
91	146	102.27266	2.2529411	384	713	1		
92	147	102.27267	2.2529409	384	713	1		
93	148	102.27267	2.2529407	384	713	1		
94	149	102.27267	2.2529405	384	713	1		
95	150	102.27267	2.2529404	384	713	1		
96	151	102.27268	2.2529402	384	713	1		
97	152	102.27268	2.25294	384	713	1		
98	713	102.27268	2.2529351	384	713	1	Adjusting mechanism (adjuster star wheel)	3rd destination
99	152	102.27268	2.25294	713	683	1		
100	151	102.27268	2.2529402	713	683	1		
101	150	102.27267	2.2529404	713	683	1		
102	149	102.27267	2.2529405	713	683	1		
103	148	102.27267	2.2529407	713	683	1		
104	147	102.27267	2.2529409	713	683	1		
105	146	102.27266	2.2529411	713	683	1		
106	145	102.27266	2.2529414	713	683	1		
107	162	102.27266	2.252944	713	683	1		
108	163	102.27266	2.2529477	713	683	1		
109	164	102.27266	2.2529515	713	683	1		
110	24	102.27266	2.2529569	713	683	1		
111	25	102.27266	2.2529567	713	683	1		
112	26	102.27267	2.2529567	713	683	1		
113	27	102.27267	2.2529564	713	683	1		
114	28	102.27267	2.2529563	713	683	1		

Table A3. *Cont.*

SL No.	Visited Node	Longitude	Latitude	Source Node	Destination Node	Weight	Item Name	
115	29	102.27268	2.2529562	713	683	1		
116	30	102.27268	2.2529559	713	683	1		
117	31	102.27268	2.2529558	713	683	1		
118	32	102.27269	2.2529557	713	683	1		
119	33	102.27269	2.2529554	713	683	1		
120	683	102.27269	2.2529501	713	683	1	Bucket seat	4th destination

Total visited nodes = 120, total cost = 125.

References

1. Adamatzky, A. *Shortest Path Solvers. From Software to Wetware*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 32.
2. Papadopoulos, S.; Kompatsiaris, Y.; Vakali, A.; Spyridonos, P. Community detection in social media, performance and application considerations. *J. Data Min. Knowl. Discov.* **2012**, *24*, 515–554. [CrossRef]
3. Kwon, Y.S.; Sohn, M.Y. Classification of Efficient Total Domination Sets of Circulant Graphs of Degree 5. *Symmetry* **2020**, *12*, 1944. [CrossRef]
4. Yang, L.; Li, D.; Tan, R. Shortest Path Solution of Trapezoidal Fuzzy Neutrosophic Graph Based on Circle-Breaking Algorithm. *Symmetry* **2020**, *12*, 1360. [CrossRef]
5. Kalaitzakis, A. Comparative Study of Community Detection Algorithms in Social Networks. Ph.D. Thesis, Technological Educational Institute of Crete, Heraklion, Greece, 1939.
6. Bharath-Kumar, K.; Jaffe, J.M. Routing to Multiple Destinations in Computer Networks. *IEEE Trans. Commun.* **1983**, *31*, 343–351. [CrossRef]
7. Ben Ticha, H.; Absi, N. *A Solution Method for the Multi-Destination Bi-Objectives Shortest Path Problem*; Elsevier: Amsterdam, The Netherlands, 2017. [CrossRef]
8. Dong, Y.F.; Xia, H.M.; Zhou, Y.C. Disordered and Multiple Destinations Path Planning Methods for Mobile Robot in Dynamic Environment. *J. Electr. Comput. Eng.* **2016**, *2016*, 3620895. [CrossRef]
9. Sepehrifar, M.K.; Zamanifar, K.; Sepehrifar, M.B. An Algorithm to Select the Optimal Composition of the Services. *J. Theor. Appl. Inf. Technol.* **2009**, *8*, 154–161.
10. Wang, W.; Uehara, M.; Ozaki, H. Evaluation of navigation based on system optimal traffic assignment for connected cars. *Int. J. Grid Util. Comput.* **2020**, *11*, 525–532. [CrossRef]
11. Burgaña, J.L. Design and evaluation of a link-state routing protocol for Internet-Wide Geocasting. Master's Thesis, University of Twente, Enschede, The Netherlands, 2017.
12. Sepehrifar, M.K.; Fanian, A.; Sepehrifar, B. Shortest Path Computation in a Network with Multiple Destinations. *Arab. J. Sci. Eng.* **2020**, *45*, 3223–3231. [CrossRef]
13. Jubair, F.; Hawa, M. Exploiting Obstacle Geometry to Reduce Search Time in Grid-Based Pathfinding. *Symmetry* **2020**, *12*, 1186. [CrossRef]
14. Rahman, M.S.; Ahmed, S. A survey on pairwise compatibility graphs. *AKCE Int. J. Graphs Comb.* **2020**, *17*, 788–795. [CrossRef]
15. Easttom, W.; Adda, M. An enhanced view of incidence functions for applying graph theory to modeling network intrusions. *WSEAS Trans. Inf. Sci. Appl.* **2020**, *15*, 102–109.
16. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2009.
17. Arman, N.; Khamayseh, F. A Path-Compression Approach for Improving Shortest-Path Algorithms. *Int. J. Electr. Comput. Eng.* **2015**, *5*, 772–781. [CrossRef]
18. Hakeem, A.; Gehani, N.; Ding, X.; Curtmola, R.; Borcea, C. Multi-destination vehicular route planning with parking and traffic constraints. In Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Houston, TX, USA, 12–14 November 2019; pp. 298–307.
19. Hu, W.-C.; Wu, H.-T.; Cho, H.-H.; Tseng, F.-H. Optimal Route Planning System for Logistics Vehicles Based on Artificial Intelligence. *J. Internet Technol.* **2020**, *21*, 757–764.
20. Li, X.; Yin, H. Optimal Mobile Relays Positions and Resource Allocation for Multi-Relay Multi-Destination Wireless Networks. *IEEE Access* **2020**, *8*, 47993–48004. [CrossRef]
21. Anđelić, M.; Živković, D. Efficient Algorithm for Generating Maximal L-Reflexive Trees. *Symmetry* **2020**, *12*, 809. [CrossRef]
22. Zhang, H.; Zhang, Z. AOA-Based Three-Dimensional Positioning and Tracking Using the Factor Graph Technique. *Symmetry* **2020**, *12*, 1400. [CrossRef]
23. Panić, B.; Kontrec, N.; Vujošević, M.; Panić, S. A Novel Approach for Determination of Reliability of Covering a Node from K Nodes. *Symmetry* **2020**, *12*, 1461. [CrossRef]
24. Slamin, S.; Adiwijaya, N.O.; Hasan, M.A.; Dafik, D.; Wijaya, K. Local Super Antimagic Total Labeling for Vertex Coloring of Graphs. *Symmetry* **2020**, *12*, 1843. [CrossRef]

25. Martínez, A.C.; García, S.C.; García, A.C.; Del Rio, A.M.G. On the Outer-Independent Roman Domination in Graphs. *Symmetry* **2020**, *12*, 1846. [CrossRef]
26. Martínez, A.C.; Estrada-Moreno, A.; Rodríguez-Velázquez, J.A. Secure w-Domination in Graphs. *Symmetry* **2020**, *12*, 1948. [CrossRef]
27. Lv, Y.; Liu, M.; Xiang, Y. Fast Searching Density Peak Clustering Algorithm Based on Shared Nearest Neighbor and Adaptive Clustering Center. *Symmetry* **2020**, *12*, 2014. [CrossRef]
28. Balakrishnan, A.; Banciu, M.; Glowacka, K.; Mirchandani, P. Hierarchical approach for survivable network design. *Eur. J. Oper. Res.* **2013**, *225*, 223–235. [CrossRef]
29. Fredman, M.L.; Tarjan, R.E. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* **1987**, *34*, 596–615. [CrossRef]
30. Qu, T.; Cai, Z. A Fast Isomap Algorithm Based on Fibonacci Heap. In *International Conference in Swarm Intelligence*; Springer: Cham, Switzerland, 2015; pp. 225–231.
31. Lu, X.; Camitz, M. Finding the shortest paths by node combination. *Appl. Math. Comput.* **2011**, *217*, 6401–6408. [CrossRef]
32. Orlin, J.B.; Madduri, K.; Subramani, K.; Williamson, M. A faster algorithm for the single source shortest path problem with few distinct positive lengths. *J. Discret. Algorithms* **2010**, *8*, 189–198. [CrossRef]
33. Thorup, M. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM* **1999**, *46*, 362–394. [CrossRef]
34. Thorup, M. On RAM Priority Queues. *SIAM J. Comput.* **2000**, *30*, 86–109. [CrossRef]
35. MacCormick, J. *What Can Be Computed? A Practical Guide to the Theory of Computation*; Princeton University Press: Princeton, NJ, USA, 2018.
36. Xu, M.; Liu, Y.; Huang, Q.; Zhang, Y.; Luan, G. An improved Dijkstra's shortest path algorithm for sparse network. *Appl. Math. Comput.* **2007**, *185*, 247–254. [CrossRef]
37. Holzer, M.; Schulz, F.; Wagner, D.; Willhalm, T. Combining speed-up techniques for shortest-path computations. *ACM J. Exp. Algorithmics* **2005**, *10*, 2–5. [CrossRef]
38. Chen, Y.-Z.; Shen, S.-F.; Chen, T.; Yang, R. Path Optimization Study for Vehicles Evacuation based on Dijkstra Algorithm. *Procedia Eng.* **2014**, *71*, 159–165. [CrossRef]
39. Madkour, A.; Aref, W.G.; Rehman, F.U.; Rahman, M.A.; Basalamah, S. A survey of shortest-path algorithms. *arXiv* **2017**, arXiv:1705.02044.
40. Okengwu, U.A.; Nwachukwu, E.O.; Osegi, E.N. Modified Dijkstra algorithm with invention hierarchies applied to a conic graph. *arXiv* **2015**, arXiv:1503.02517.
41. Hong, Y.; Li, D.; Wu, Q.; Xu, H. Priority-Oriented Route Network Planning for Evacuation in Constrained Space Scenarios. *J. Optim. Theory Appl.* **2019**, *181*, 279–297. [CrossRef]
42. Jin, W.; Chen, S.; Jiang, H. Finding the K shortest paths in a time-schedule network with constraints on arcs. *Comput. Oper. Res.* **2013**, *40*, 2975–2982. [CrossRef]
43. Ananta, M.T.; Jiang, J.-R.; Muslim, M.A. Multicasting with the extended Dijkstra's shortest path algorithm for software defined networking. *Int. J. Appl. Eng. Res.* **2014**, *9*, 21017–21030.