Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 9661411, 7 pages
https://doi.org/10.1155/2022/9661411

Research Article

A Robust Graph Theoretic Solution of Routing in Intelligent Networks

Muhammad Aasim Qureshi,[1] Mohd Fadzil Hassan,[2] Muhammad Khurram Ehsan,[3] Muhammad Owais Khan,[4] Md YeaKub Ali,[5] and Shafiullah Khan[6]

[1]*Department of Computer Sciences, Bahria University, Lahore Campus, Lahore, Pakistan*
[2]*Computer and Information Science Department, University Teknologi, Petronas, Malaysia*
[3]*Faculty of Engineering Sciences, Bahria University, Lahore Campus, Lahore, Pakistan*
[4]*School of Systems and Technology, University of Management and Technology, Lahore, Pakistan*
[5]*Department of Electronics and Telecommunication Engineering, Rajshahi University of Engineering & Technology (RUET), Rajshahi, Bangladesh*
[6]*Department of Electronics, Islamia College University, Peshawar, Pakistan*

Correspondence should be addressed to Md YeaKub Ali; yeakub@ete.ruet.ac.bd

Received 12 April 2022; Revised 20 May 2022; Accepted 3 June 2022; Published 20 June 2022

Academic Editor: Khaled Maaiuf Rabie

Implementation of robust routing is very critical in network communication. Connecting devices like routers maintain databases for the whole network topology in the routing table. Each router needs to keep these tables updated with the best possible routes so that an efficient communication can always take place in nondelay tolerant intelligent networks that include military and tactical systems, vehicular communication networks, underwater acoustic networks, and intelligent sensor networks. The fast construction of shortest-path tree (SPT) is important to devise an efficient routing in a nondelay tolerant networks. That is why a simple and efficient algorithm is the need of the time. A robust routing solution SPT with $O(V + E)$ time complexity is proposed to supersede the existing landmark.

1. Introduction

Network routing is a backbone of intelligent communication networks specifically nondelay tolerant networks that include military and tactical systems, vehicular communication networks (VCNs), underwater acoustic networks (UANs), and intelligent sensor networks (ISNs) to send the communication data between vertices in a certain network. For the purpose, every router maintain routing tables. These tables manage routes of various network destinations. This network routing process is generally accomplished based on these routing tables and routes defined in them. This makes the formation, written in a memory of these routers, critically important for successful and efficient routing. There exist many graph theoretic solutions of this routing problem. In a network, the vertex of a graph represents a router, and edge/link which connect these vertices represents physical links between the routers.

The needs for broadband Internet applications have increased rapidly in today's Internet. Which makes high speed routing inevitable and has become the key at open shortest-path first (OSPF). As a result of topological changes that take place due to due to an unavoidable circumstances at the OSPF, these algorithms update the routing table to handle the new topological changes. For example, if some link fails in a network, then there is no way but to recalculate these paths [1]. In such scenarios, these shortest paths are regenerated by running the algorithms again [2].

We can formally define shortest-path problem as follows:

Let $G = (V, E)$, where $V = \{v_1, v_2, v_3, v_4, \cdots, v_n\}$ and $E = \{e_1, e_2, e_3, e_4, \cdots, e_m\}$.

G is an undirected nonnegative weighted connected graph (in this scenario, it will be representing the routing network). G has a vertex "s" designated as source and another vertex, "t,", designated as destination, such that $s, t \in V$. We need to find simple path(s) from s to t with minimum incurred cost.

[3] presented a hybrid model with local and global modes. A fixed graph model is used to present the local mode connections. In the paper, they denoted graph by a unit-disk graph UDG(V): for any $u, v \in V$, {u,v} $\in$ UDG(V) if and only if distance between u and v is less than 1. For the local mode, in each round, node u can send a message of $O(\log n)$ bits to node v. For the global mode, they used a variant of the node-capacitated clique (NCC) model presented in [4] that describes salient features of overlay networks. In [5], the authors explain the SPSN approach including route object modeling, capacity-oriented search, and possible route organization procedures. Prototype assessments presented significant support that SPSN is better than the legacy CGR in more than an order of magnitude in contact plans comprising thousands of contacts.

The shortest-path tree (SPT) problem is one of the most well-known problems in graph theory [6]. Since 1959, most of the advances in SPT for general graphs, directed and undirected, are based on Dijkstra's algorithm. There exist lots of applications of SPT problems like computer systems, transportation networks, and vehicle routing [7]. Time complexity of the basic algorithm of Dijkstra [8] is calculated as $O(V^2 + E)$ (where V is number of vertices and E is number of edges) if linear search is applied to find the minimum [9]. [10, 11] introduced a new data structure, heap, to find the minimum. This new data structure improved the time complexity to $O(E \log V)$. Fibonacci heap improved this bound further [12]. Authors of [12] used comparison model for the optimal implementation of Dijkstra's algorithm as Dijkstra's algorithm traverses vertices in ascending order.

In [13], with the use of optimal structure of the data, heap data structure provides the complexity of $O(E \log V)$ with implementation of Dijkstra and Fibonacci heap gives the complexity of $O(E + V \log V)$ with the implementation of Dijkstra. For the stochastic decision-making problem in [14], the stochastic dynamic programming solution is required. The generalized elementary shortest-path problem in [15] shows that two-phase heuristic graphs by including negative cycles can improve the average gap of 0.3% if compared with the best known solutions if number of vertices N are set to 100. The solution in [16] suggests that these algorithms can be implemented in all kind of networks with complexity of Dijkstra $O(V^2 + E)$, Bellman $O(V)^3$, and Floyd $O(VE)$. The paper [17] suggests a solution with $O(kV \log (kV) + kE)$ complexity for a fast delivery problem. The complexity of Bellman is $O(V)^3$, and the complexity of Dijkstra is $O(V)^2$ if the network is fully connected or near to the fully connected [18].

Invent of fusion trees and its application in Dijkstra's algorithm make randomized bound of time complexity, $O(E(\log V)1/2)$. Later, the invention of atomic heaps improved this time bound to $O(E + n \log (v/\log (\log V)))$ bound and $O(E + V(\log V + \varepsilon)1/2)$ [12]. Afterwards, in [6, 19, 20], priority queues improved to $O(E \log (\log (V))$ and $O(E + V \log V^{1+\epsilon})^{1/2}$. These worst-case time bounds are randomized supposing linear space are prerequisite. The researcher of [21] further improved it to $O(E + V (\log V \log \log V)^{1/2})$ which was improved by the authors of [16], next year, using randomized bound to $O (E + V \log V^{1+\epsilon})^{1/3}$.

[2] presented priority queue, which improved single source shortest-path cost to $O(E + V(\log C)^{1/2}$ where C is the cost of the most heavy edge. This bound was further improved by [22] to $O(E + V(3 \log C \log\log C)^{1/3})$ expected time, and [23] presented a further improvement to $O(E + V(\log C)^{1/4+\epsilon})$. The authors of [24] claimed that the algorithm they have presented will outclass Dijkstra algorithm.

In [25], the authors presented an exact adjacency-based primal algorithm to resolve the SPT problem with high resources. Their algorithm explored solution space iteratively using path adjacency-based partition. They claimed optimal convergence with good path in lesser cost. Authors of [26] introduced a multiobjective (time and cost) shortest-path tree problem for connected directed graphs. Every edge was having two cost values. They proved that uncertainty MSPP reduced the total travel time and cost.

[27] used a dual evolutionary algorithm while analyzing the problem of clustered shortest-path tree. Their first algorithm (N-LSE) targeted the minimization of search space with fewer vertices. The second algorithm named M-LSE using N-LSE creates smaller and multiple intercluster edges. [28] solved the dynamic SP problem to solve time-dependent TSP. In logistic planning where traffic data is rich enough, time-dependent path optimization techniques give considerable benefits. Dynamic shortest-path solution contains no length k cycles and resolves the complexity problem when compared to acyclic graph-based methods.

In [29], the author reviewed many popular algorithms for solving the SPT problem in weighted graphs. He proposed two variants of Dijkstra's algorithm. He presented asymptotic comparison different variants of Dijkstra's algorithm on expanded graphs. He presented $O(E' + V' \lg V')$ time algorithm where $E' = E + C^-_{\max} C^+_{\max}$ and $V' = V(C^-_{\max} + C^-_{\max})$. They used color-in and color-out scheme.

2. Concept

This work presents a novel algorithm of finding shortest-path routes in intelligent networks. This algorithm is based on bidirectional search where first search is starting from source vertex, and its target is destination vertex, while second search starts from the destination vertex, and its target is source vertex. It is analogous to two robots looking for each other from two distinct points. First robot, R_A, starts its search from its current position, and let us call it P_A, and second robot, R_B, starts its search from its current position, and let us call it P_B. R_A explores all possible paths, in

BFS fashion, looking for P_B or R_B. In the same way, R_B will be exploring all possible paths, simultaneously, looking for either P_A or R_A. Following are the three possible situations that can rise, in this bidirectional search.

(i) R_A will find P_B

(ii) R_B will find P_A

(iii) R_A and R_B meets at position, P_M

Situation i is the simplest one. Let Path_A be the path that R_A has explored and has vertices $a_1, a_2, a_3, \cdots, a_x$, where a_x is P_B.

$$\text{Required path} = \text{Path}_A = \sum_{i=1}^{x-1} e_{a_i,a_{i+1}}. \tag{1}$$

In situation ii, R_B has found P_A. It means that path is found but in reverse order. Let R_B has explored the vertices in sequence $b_1, b_2, b_3, \cdots, b_y$, where b_y is P_A.

$$\text{Path}_B = \sum_{i=1}^{y-1} e_{b_i,b_{i+1}}. \tag{2}$$

The reverse of this will generate the required path. See the following equation:

$$\text{Required path} = \sum_{i=y}^{1} e_{b_i,b_{i-1}}. \tag{3}$$

In situation iii, it is little complex than the previous two. Two paths—Path_A and Path_B, need to be concatenated. Let Path_A be the path that R_A has explored and has vertices $a_1, a_2, a_3, \cdots, a_x$. Path_B is the path that R_B has explored and has vertices $b_1, b_2, b_3, \cdots, b_y$, where a_x and b_y are the same vertices as P_M and a_1 and b_1 are P_A and P_B, respectively. The required path will be having vertices in the sequence $a_1, a_2, a_3, \cdots, a_x, b_{y-1}, b_{y-2}, b_{y-3}, \cdots, b_1$, as given in equation (3).

$$\text{Required path} = \sum_{i=1}^{x-1} e_{a_i,a_{i+1}} + \sum_{i=y}^{2} e_{b_i,b_{i-1}}. \tag{4}$$

3. Proposed Algorithm

The proposed algorithm targets only those graphs in which on two paths, from source, P_i and P_j, to fulfil the following property:

$$Len(P_i(s,w)) > Len\left(P_j(s,w)\right), \tag{5}$$

$$\sum_{\substack{i=0 \\ \text{for } P_i}}^{k-1} w_{x_i,x_{i+1}} > \sum_{\substack{i=0 \\ \text{for } P_j}}^{l-1} w_{y_i,y_{i+1}}, \tag{6}$$

where $k \neq l$ and $x_0 = y_0 = s$ and $x_k = y_k = w$ and P_i and P_j are paths from s to w such that $P_i = \{x_1, x_2, x_3, \cdots, x_k\}$ and $P_j = \{y_1, y_2, y_3, \cdots, y_l\}$.

The search is executed, parallel, from both ends—source, s, and destination, t. Vertices are explored level by level. Best paths are calculated by concatenating the new edge with previously explored best path, and cost is calculated by adding the cost of all edges in that path from source.

Each vertex during the execution goes through three states, being represented by three colors—green, yellow, and red. The definitions of these colors can be seen in Table 1

Initially, all vertices, i.e., $v_1, v_2, v_3, \cdots, v_n$, are initialized with

$$\begin{cases} \text{SPCost}_{v_i} \longleftarrow \infty, \\ \pi_{v_i} \longleftarrow \varnothing, \\ C_{v_i} \longleftarrow \text{green}. \end{cases} \tag{7}$$

Starting from s and t, independently and simultaneously, the current vertex, let us call it v_{cur}, will explore its neighbors. Every neighbor, i.e., v_{nbr}, is updated with shortest-path cost, and let us call it SPCost, parent, i.e., π and color, i.e., C.

$$\text{SPCost}_{v_{\text{nbr}}} = \min\left(\text{SPCost}_{v_{\text{cur}}} + w_{v_{\text{cur}},v_{\text{nbr}}}, \text{SPCost}_{v_{\text{nbr}}}\right), \tag{8}$$

$$\pi_{v_{\text{nbr}}} = v_{\text{cur}} \text{ if } \text{SPCost}_{v_{\text{nbr}}} \text{ is updated}, \tag{9}$$

$$\text{Clr}_{v_{\text{nbr}}} = \text{yellow if } \text{SPCost}_{v_{\text{nbr}}} \text{ is updated}. \tag{10}$$

Update will be effective if new $\text{SPCost}_{v_{\text{nbr}}}$ reflects improvement over previously stored one.

Initially all vertices have green status. As soon as a vertex is explored during the search, its status is updated to yellow. As soon as any vertex completes its exploration (i.e., all its neighbors are explored, and their status is changed to yellow), its status is updated to red.

Exploration takes place, strictly, level by level. No next level vertex is explored until all the previous level yellow vertices' status is change to red. As soon as one level is completed, the control is switched to the other part of the algorithm to proceed and it also performs the identical steps.

During the search of any side if some red (light or dark) vertex is explored by the counterpart search, then two shortest paths are calculated and stored. If some shortest path has already been stored, then the path will less cost will be kept.

The algorithm terminates when the status of all vertices is updated to red (light or dark). In worst case, depending upon the type of the graph, it finishes in $O(V+E)$ time. If graph is totally disconnected, even then loop in line number 4 of *ShortestPath* will execute for $O(V)$ times. If graph is connected and is having E, then combined effect of line number 4 of *ShortestPath* and line number 2 of *SPfromSource* or *SPfromDestination* would be "all adjacents of all vertices" will cost $O(E)$ which will make it $O(V+E)$. Detailed algorithm can be seen in Algorithm 1.

Table 1: Definitions of colors that represent the state.

Color	Definition
Green	Unseen: the vertex is yet virgin, no neighboring vertex has yet seen/explored it and considered it to be the part of the shortest path
Yellow	Seen: some neighboring vertex has seen/explored it and considered it to the part of shortest path
Red	Visited: this vertex has played its part and has explored all its neighbors, making them yellow

```
ShortestPath(s, t, G)
1. Queue1.enQueue(s)
1. Queue2.enQueue(t)
2. InitializeDoubleSource(SPCost, Pi, Clr, s, t, G)
3.      Flag ← 0
4. While (Queue1 <> Null OR Queue2 <> Null) AND Flag = 0
5.      Do SPfromSource(SPCost, Pi, Clr, G)
6.         SPfromDestination(SPCost, Pi, Clr, G)
SPfromSource(G)
1. while u ← Queue1.deQueue() <> Green
2. for each v belongs to Adj[u]
3.      do if Clr_v = Green OR LightYellow OR LightRed
4.            then Clr_v ← LightYellow
5.                 SPCost_v ← min(SPCost_u +W_(u,v), SPCost_v)
6.                 If SPCost is updated
7.                    Then Pi_v ← u
8.                 If Clr_v = Green OR LightRed
9.                    Then Queue.enQueue(v)
10.             If Clr_v = DarkRed
11.                Then Concatinate two paths and store
12.                     CPCost ← min(CPCost_u + CPCost_v + w_(u,v), CPCost)
13.             If Clr_v = DarkYellow
14.                then skip
15. Clr_u ← LightRed
SPfromDestination(G)
1. while u ← Queue2.deQueue() <> Green
2. for each v belongs to Adj[u]
3.   do if Clr_v = Green OR DarkYellow OR DarkRed
4.      then Clr_v ← DarkYellow
5.      SPCost_v ← min(SPCost_u +W_(u,v), SPCost_v)
6.      If SPCost_v is updated
7.         Then Pi_v ← u
8.      If Clr_v = Green OR DarkRed
9.         Then Queue.enQueue(v)
10.   If Clr_v = LightRed
11.      Then Concatinate two paths and store
12.           SPCost_v ← min(SPCost_u + SPCost_v + w_(u,v), SPCost_v)
13. If Clr_v = LightYellow
14.           then skip
15. Clr_u ← DarkRed
```

Algorithm 1.

4. Working Example of the Algorithm

All vertices are initialized using equation (7) (see Figure 1). Both sources—*A* and *P*, are colored yellow and their SPCosts are marked 0 (see Figure 1(a)). All vertices are enqueued. Main subroutine (named ShortestPath) of the algorithm calls the two replicated subroutines (named SPfromSource and SPfromDestination) one by one until queue has any vertex.

The subroutine SPfromSource dequeues a vertex, it would be source, and here in this example, it is *A*. Algorithm then explores all its neighbors—*B*, *F*, and *E*, one by one.

Equations (8)–(10) are used to update the values of SPCost, π, and Clr. In the same way, SPfromDestination dequeues a vertex, it would be destination, and here in this example, it is *P*. Algorithm then explores all its neighbors—*L* and *K*, one by one. Equations (8)–(10) are

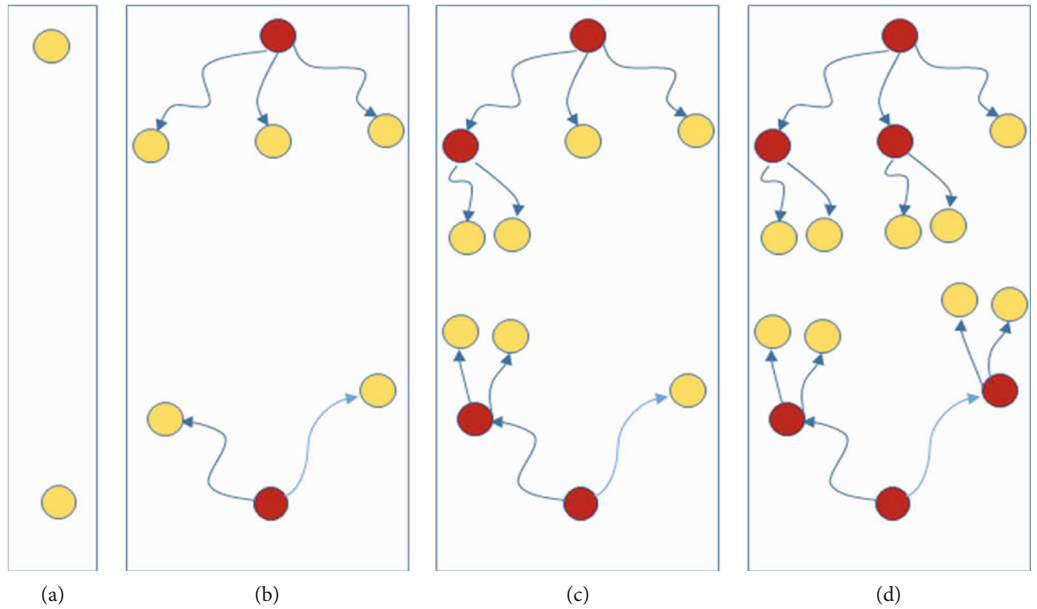

Figure 1: Explaining the process of the algorithm for simplification weights and other details are skipped: (a) initialization: (b) source and destination explored their neighbors; (c) neighbors of neighbors of source and destinations are explored one by one; (d) neighbors of neighbors of source and destinations are explored one by one.

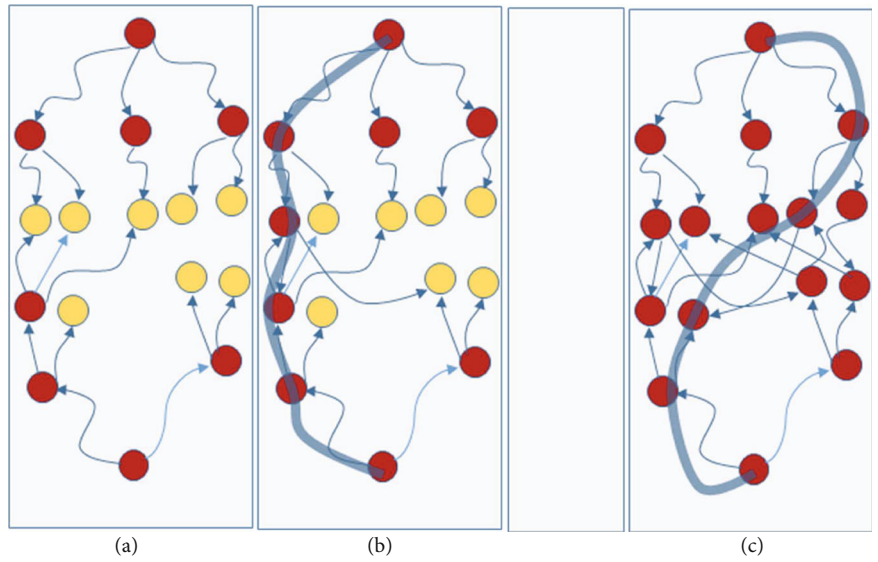

Figure 2: (a) Extract one vertex from queue from both sides and explore their neighbors; (b) explore a red vertex; applying case 3, path is calculated and stored; (c) repeat until all vertices change their color to red, and if new path is shorter, then it is stored and old one is skipped. Finally, the shortest path is obtained.

used to update the values of SPCost, π, and Clr. A and P are colored red after all explorations. New updated status can be seen in Figure 1(b).

In the same way, SPfromDestination dequeues a vertex, and it would be path, equations (9) and (10). Figures 1(c), 1(d), and 2(a) explain these iterations one by one. Figure 2(b) shows that a red vertex is explored from the other side. It generates case 3 (i.e., "R_A and R_B meet at position, P_M"). Path from both sides along with currently updating edge is concatenated to form a path using equation (7). This path is stored.

This process of dequeuing, exploring, and generating new path continues until both queues have no vertices. New paths are generated and compared with the stored one and is discarded if old one is better Figure 2(c). But if new path is better than the old one, it is replaced with the old one. The process ends with the shortest possible path.

5. Discussion

The main advantage of this algorithm is due to the nature of the networks in which it is not possible that indirect distance

can be smaller than direct distance between two vertices. Since Dijkstra invented his SPT algorithm, the research community has worked mostly in the similar direction: choose a node to propagate in nondecreasing order and manage and update neighboring vertices through it. To select vertices in a nondescending order, the half-explored vertices must be sorted, or on each selection, the minimum distance must be calculated from the remaining vertices, which again is equivalent to sort [29].

The main bottleneck of these algorithms is the intermediary sorting algorithm. Most researchers have worked to surmount this bottleneck. New data structures like tree, heap, and priority queue were designed to improve the time bounds. One of these is Fibonacci heap [30] that resulted in the best known asymptotic bound of Dijkstra's algorithm [31]. However, this bound is also criticized by researchers due to the heavy processes of Fibonacci heap [32–37]. This algorithm is beating this bound of Dijkstra's algorithm using Fibonacci heap, where sorting is not being attempted in any form. On one side, it is thrashing this time bound with simple data structure and no sorting algorithm; on the other side, it has parallelization in its nature that is beyond the scope of this paper.

At the same time, this algorithm is also better than the one presented in [29] with worst-case time bound of $O(E' + V' \lg V')$ time algorithm where $E' = E + C^{-}_{\max}C^{+}_{\max}$ and $V' = V(C^{-}_{\max} + C^{-}_{\max})$. The algorithm proposed in this research has worst-case time bound of $O(V + E)$, which is linear in time and hence is better than the one presented in [29] for routing in intelligent networks [38–42].

6. Conclusion

The proposed SPT algorithm executes with $O(V + E)$ time which is relatively better than the state of art work. It is simple to understand and easy to implement and does not require high computational resources. It has additional benefit that after some specific stage, it can always hold a path from source to destination if compromise on quality of path is possible to address the latency issues in nondelay tolerant networks that include military and tactical systems, vehicular communication networks, underwater acoustic networks, and intelligent sensor networks.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] S. K. Lee, J. W. Jang, S. J. Jang, and J. Y. Shin, "—Development and performance analysis of ABR-DBA algorithm for improve network performance‖," *International Journal of Future Generation Communication and Networking*, vol. 1, pp. 1–6, 2008.

[2] R. K. Ahuja, K. Melhorn, J. B. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *Journal of the ACM*, vol. 37, no. 2, pp. 213–223, 1990.

[3] M. S. Sam Coy, A. Czumaj, M. Feldmann et al., "Near-shortest path routing in hybrid communication networks," *Distrib. Parallel, Cluster Computing*, vol. 217, pp. 1–26, 2022.

[4] S. S. J. Augustine, K. Choudhary, A. Cohen, D. Peleg, and S. Sourav, "Distributed graph realizations," in *In Proc. of the 34th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2020)*, pp. 158–167, New Orleans, Louisiana, USA, 2020.

[5] O. De Jonckère and J. A. Fraire, "A shortest-path tree approach for routing in space networks," *China Communications*, vol. 17, no. 7, pp. 52–66, 2020.

[6] A. Andersson, P. B. Miltersen, and M. Thorup, "Fusion trees can be implemented with AC^0 instructions only," *Theoretical Computer Science*, vol. 215, no. 1-2, pp. 337–344, 1999.

[7] B. Zhang, J. Zhang, and L. Qi, *The Shortest Path Improvement Problems under Hamming Distance*, Springer Science+Business Media, LLC, 2006.

[8] E. W. Dijekstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[9] M. Thorup, *Undirected single-source shortest paths with positive integer weights in linear time*, vol. 46, no. 3, 1999, AT&T Labs Research, Florham Park, New Jersey, 1999.

[10] J. W. J. Williams, *Heapsort. Commun*, ACM, 1998.

[11] J. Hershberger, S. Suri, and A. Bhosle, "On the difficulty of some shortest path problems," *ACM Trans. Algorithms*, vol. 3, no. 1, pp. 1–15, 2007.

[12] M. L. Fredman and D. E. Willard, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[13] P. Van Mieghem, "The Shortest Path Problem," in *Performance Analysis of Communications Networks and Systems*, pp. 347–386, Cambridge University Press, Cambridge, 2006.

[14] Z. Y. Xie, Y. R. He, Y. T. Jiang, and C. C. Chen, "Improved and/or tree search algorithm in analysis of stochastic and time-dependent shortest path problem," *Scientific Programming*, vol. 2021, Article ID 9922466, 19 pages, 2021.

[15] W. J. Guerrero, N. Velasco, C. Prodhon, and C. A. Amaya, "On the generalized elementary shortest path problem: a heuristic approach," *Electronic Notes in Discrete Mathematics*, vol. 41, pp. 503–510, 2013.

[16] K. Magzhan and H. Jani, "A review and evaluations of shortest path algorithms," *International Journal of Scientific and Technology Research*, vol. 2, no. 6, pp. 99–104, 2013.

[17] I. A. Carvalho, T. Erlebach, and K. Papadopoulos, "An Efficient Algorithm for the Fast Delivery Problem," in *Fundamentals of Computation Theory*, L. Gąsieniec, J. Jansson, and C. Levcopoulos, Eds., vol. 11651 of Lecture Notes in Computer Science, Springer, Cham, 2019.

[18] D. Medhi and K. Ramasamy, *Routing Algorithms: Shortest Path, Widest Path, and Spanning Tree*, Network Routing, 2018.

[19] M. Therup, "On RAM priority queues," in *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 59–67, New York, 1996.

[20] M. Thorup, "Floats, integers, and single source shortest paths," in *In Proceedings of the 15th Symposium on Theoretical Aspects*

of Computer Science. Lecture Notes on Computer Science, pp. 14–24, 1373. ,Springer-Verlag, New York, 1998.

[21] R. Raman, "Priority queues: small monotone, and trans-dichotomous," in *In Proceedings of the4th Annual European Symposium on Algorithms. Lecture Notes on Computer Science*, pp. 121–137, 1136, Springer-Verlag, New York, 1996.

[22] B. V. Cherkassky, A. V. Goldberg, and C. Silverstein, "Buckets, heaps, lists, and monotone priority queues," in *In Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 83–92, New York, 1997.

[23] R. Raman, "Recent results on the single-source shortest paths problem," *SIGACT News*, vol. 28, no. 2, pp. 81–87, 1997.

[24] S. Pettie, V. Ramachandran, and S. Sridhar, "Experimental evaluation of a new shortest path algorithm_ (extended abstract)," in *ALENEX 2002, LNCS 2409*, D. Mount and C. Stein, Eds., pp. 126–142, Springer-Verlag, Berlin Heidelberg, 2002.

[25] F. S. I. Himmich, H. B. Amor, and I. El Hallaoui, "A primal adjacency-based algorithm for the shortest path problem with resource constraints," *Transportation Science*, vol. 54, no. 5, pp. 1153–1169, 2020.

[26] S. Majumder, M. B. Kar, S. Kar, and T. Pal, "Uncertain programming models for multi-objective shortest path problem with uncertain parameters," *Soft Computing*, vol. 24, no. 12, pp. 8975–8996, 2020.

[27] P. T. Hanh, P. D. Thanh, and H. T. Binh, "Evolutionary algorithm and multifactorial evolutionary algorithm on clustered shortest-path tree problem," *Information Science*, vol. 553, pp. 280–304, 2021.

[28] C. Hansknecht, I. Joormann, and S. Stiller, "Dynamic shortest paths methods for the time-dependent TSP," *Algorithms*, vol. 14, no. 1, p. 21, 2021.

[29] R. Lewis, "Algorithms for finding shortest paths in networks with vertex transfer penalties," *Algorithms*, vol. 13, no. 11, p. 269, 2020.

[30] M. Qureshi, D. Hassan, and B. Fadzil, "AO (E) time shortest path algorithm for non negative weighted undirected graphs," *International Journal of Computer Science and Information Security*, vol. 6, no. 1, pp. 40–46, 2009.

[31] V. S. Pal and Y. R. Devi, "A survey on IP fast rerouting schemes using backup topology," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 4, pp. 213–218, 2013.

[32] I. Qadeer and M. Khurram Ehsan, "Improved channel reciprocity for secure communication in next generation wireless systems," *Computers, Materials & Continua*, vol. 67, no. 2, pp. 2619–2630, 2021.

[33] E. Gutiérrez and A. L. Medaglia, "Labeling algorithm for the shortest path problem with turn prohibitions with application to large-scale road networks," *Ann. Oper. Res.*, vol. 157, no. 1, pp. 169–182, 2007.

[34] M. Weigl, B. Siemiàatkowska, K. A. Sikorski, and A. Borkowski, "Grid-based mapping for autonomous mobile robot," *Robotics and Autonomous Systems*, vol. 11, no. 1, pp. 13–21, 1993.

[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 2nd edition, 2001.

[36] C. S. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT press, 2022.

[37] S. S. Skiena, *The Algorithm Design Manual*, Springer, 2008.

[38] M. Ehsan, "Performance analysis of the probabilistic models of ISM data traffic in cognitive radio enabled radio environments," *IEEE Access*, vol. 8, no. 1, pp. 140–150, 2020.

[39] M. Ehsan, A. A. Shah, M. R. Amirzada et al., "Characterization of sparse WLAN data traffic in indoor opportunistic environments as a prior for coexistence scenarios of modern wireless technologies," *Alexandria Engineering Journal*, vol. 60, no. 1, pp. 347–355, 2021.

[40] M. Ehsan and D. Dahlhaus, "Statistical modeling of ISM data traffic in indoor environments for cognitive radio systems," in *IEEE Digital Information, Networking, and Wireless Communication (DINWC), 2015 Third International Conference*, pp. 88–93, Moscow, Russia, 2015.

[41] A. Mahmood, A. Ahmed, M. Naeem, M. R. Amirzada, and A. Al-Dweik, "Weighted utility aware computational overhead minimization of wireless power mobile edge cloud," *Computer Communications*, vol. 190, pp. 178–189, 2022.

[42] N. Naz, M. Ehsan, M. R. Amirzada, M. Y. Ali, and M. A. Qureshi, "Intelligence of autonomous vehicles: a concise revisit," *Journal of Sensors*, vol. 2022, Article ID 2690164, 11 pages, 2022.