Advanced Geomatics

http://publish.mersin.edu.tr/index.php/geomatics/index

e-ISSN: 2791-8637

A Computationally Reproducible Approach to Dijkstra's Shortest Path Algorithm

Muammer Özkan[1], Ayça Tabakoğlu[1], Ekin Gönenç Uygun[1] Berk Anbaroğlu*[1]

[1] *Hacettepe University, Geomatics Engineering, Ankara, Turkey*

Keywords
Shortest path,
Routing Algorithms,
Dijkstra's Algorithm,
NetworkX.

ABSTRACT

One of the common problems in spatial analysis is the Shortest Path Problem (SPP), which aims to determine the shortest path between any given two points on a network structure. Although Shortest Path Problem is a widely used in spatial analysis, there has been lack of online resources to ease learning of the method with a computationally reproducible approach. This article presents how Dijkstra's algorithm works when finding the shortest path. Specifically, the developed online tutorial relied on the openly available road network data of San Francisco, NetworkX a Python package to realize complex graph analysis, and finally QGIS to visualize the shortest paths. All of the discussed material is presented as a Jupyter Notebook to ease computational reproducibility.

1. INTRODUCTION

A graph (network) data structure is commonly used in a Geographical Information System software for various purposes ranging from navigation (Zeng & Church, 2009) to social network analysis (Kumar, Kumar, & Soni, 2021). A graph is composed of nodes and edges, where an edge links two nodes. One of the purposes that utilize graph data structures in geomatics applications is the identification of the least-cost tree connecting all nodes of a graph (i.e. identification of the minimum spanning tree). Such an approach could be used to optimize the structure of an, for example, electricity-distribution network (Çalışkan & Anbaroğlu, 2020) and path planning. Finding the shortest path between two nodes of a graph is widely used not only in our daily lives but also for designing efficient public transportation systems (Yu, Kong, Shao, & Yan, 2018).

The Shortest Path Problem (SPP) consists of determining a path between a beginning (source) location and an end (target) location, such that the distance is minimum compared to alternatives (Rout, Vemireddy, Raul, & Somayajulu, 2020). In general, SPP is represented by a graph with several paths to be evaluated, which represents a computational difficulty. Thus, many types of research from computer science and engineering areas focus on developing efficient algorithms to solve the SPP (Huber & Rust, 2016).

The wide use of SPP in research and practice necessitates the development of openly available solutions that eases learning and experimentation. Amongst the other scripting/programming languages, Python programming language is prominent due to its natural integration to QGIS, a commonly used open-source GIS software. Furthermore, the availability of Jupyter Notebook, which is a computational notebook where a researcher can integrate both code and explanatory text. Utilizing computational notebooks have recently been used in data science and teaching GIS related courses (Kim & Henke, 2021).

This research aims to develop a computationally reproducible approach to Dijkstra's algorithm, one of the first algorithms to solve the SPP, by employing one of the renowned Python packages, *NetworkX*, on an openly available dataset. The developed Jupyter Notebook, and data are shared on GitHub to ease computational reproducibility (*GitHub - Banbar/Shortest_path_Dijkstra*, n.d.).

***Corresponding Author**

(muammerozkan@hacettepe.edu.tr) ORCID ID 0000-0002-7935-5992
(aycatabakoglu@hacettepe.edu.tr) ORCID ID 0000-0003-2317-8357
(eguygun@hacettepe.edu.tr) ORCID ID 0000-0001-8494-2545
*(banbar@hacettepe.edu.tr) ORCID ID 0000-0003-2331-6190

Cite this;

Özkan, M., Tabakoğlu, A., Uygun, E. G. & Anbaroğlu, Berk (2022). A Computationally Reproducible Approach to Dijkstra's Shortest Path Algorithm. Advanced Geomatics, 2(1), 01-06.

2. METHOD

This section first describes the Dijkstra's algorithm and then the NetworkX package.

2.1. Dijkstra's Algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. The algorithm is it has O(nlogn) complexity. Also, this algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph.

Dijkstra's algorithm assumes an infinite weight to the nodes if it does not readily know their weights. Specifically, when moving from node A to node E, as illustrated in Figure 1, the algorithm initially assumes that reaching node E would have an infinite cost. This is because node E is not adjacent to node A. This is represented in the first step of the algorithm illustrated in Table 1.

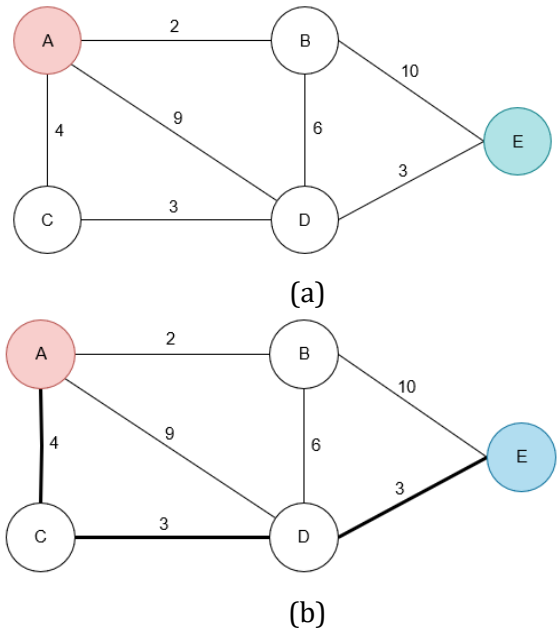

Figure 1. A sample graph (a), the shortest path between nodes A and E with a total cost of 10 (b)

We assume that the route starts at node A and finishes at node E. The execution steps are illustrated in Table 1 consisting of four steps. First step of Dijkstra's algorithm is to identify the weights of the neighbors of the start point. The weights of the neighbors of node A are illustrated in Table 1. After finding the adjacent nodes, the algorithm moves to the node that has the least cost. Specifically, in the second step the algorithm starts searching the graph from node B. Since moving to node B incurs of cost of two, it is also recorded. For example, the cost of edge BD is six, and with the additional cost of two to reaching node B, it has a weight of eight. As this is an improvement, an update operation takes place in terms of reaching node D, which used to be from node A with a weight of nine. Once all the nodes are visited, the algorithm terminates.

The resulting fourth step identifies all the shortest paths from node A. It is straightforward to determine the shortest path, from example to node E, by following the previous nodes. Starting from the target, E, its previous node is D, whose previous node is C, and whose previous node is A (i.e. E → D → C → A). Consequently, the shortest path from node A to node E is the reverse of the outcome, which is A → C → D → E. The algorithm can be generalized to directed graphs, where the cost of an edge AB may differ from BA, by keeping an adjacency list of nodes.

Table 1. Execution steps starting from node A

Step 1: Current Node A (total cost = 0)

To	Weight	Prev. Node
B	2	A
C	4	A
D	9	A
E	∞	-

Step 2: Current Node B (total cost = 2)

To	Weight	Prev. Node
B	2	A
C	4	A
D	8	B
E	12	B

Step 3: Current Node C (total cost = 4)

To	Weight	Prev. Node
B	2	A
C	4	A
D	7	C
E	12	B

Step 4: Current Node D (total cost = 7)

To	Weight	Prev. Node
B	2	A
C	4	A
D	7	C
E	10	D

The algorithm is simple and effective. In order to run Dijkstra's algorithm faster, identification of the next node to visit can be realized by storing the weight values in a *minimum heap* data structure. The min-heap property is that when you take any two nodes of the tree (let's call them X and Y) the value of node X is greater than or equal to the value of node Y if X is a child of Y. In this way, if we move from any node to the root of the tree, the value should never increase. In this way, the weights of the nodes are sorted in ascending order and a route is created by checking the ones with less weight. An exemplar minimum heap structure is provided in Figure 2.

In addition, a heap must be a complete binary tree, which allows storing the values of a heap in a linear list structure as shown in A thorough description of a min-heap is provided (Necaise, 2011). It should be noted that complex data structures such as radix heap, may increase the computational performance of finding the SPP (Ahuja, Mehlhorn, Orlin, & Tarjan, 1990).

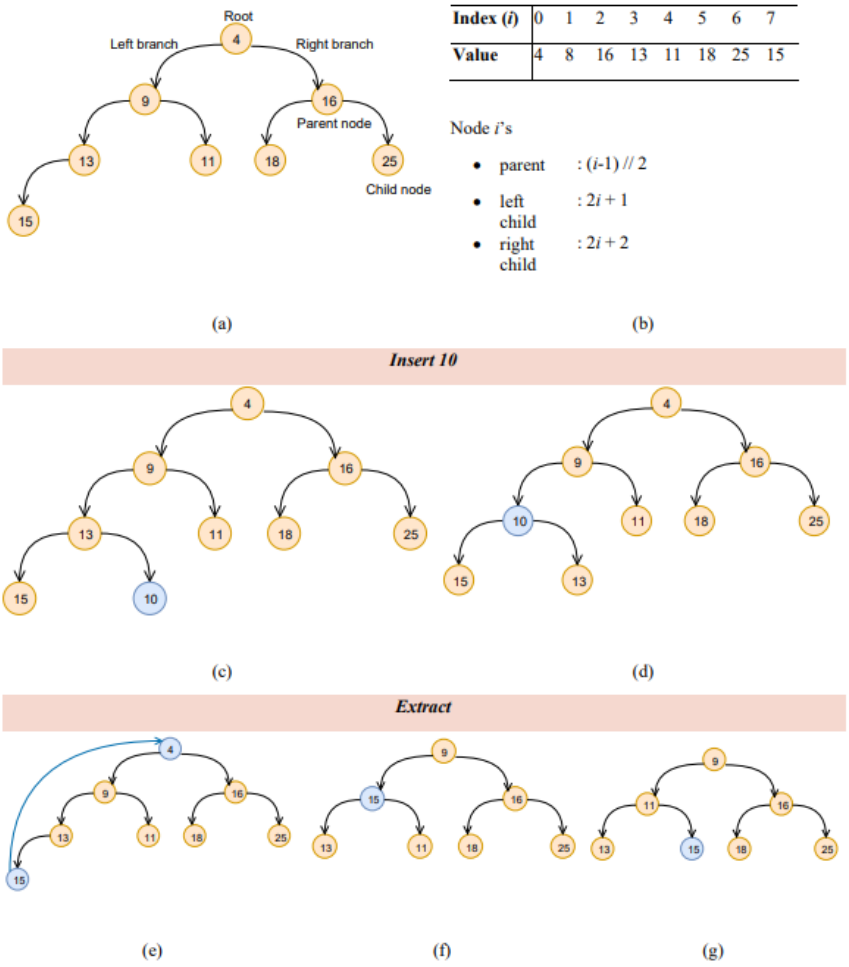

Figure 2. A min-heap (a) and its list representation (b). Inserting ten (c,d) and extraction from the heap (e, f, g)

2.2. NetworkX

NetworkX is a Python package for graph analysis. This package helps to create, manipulate, and investigate the structure, dynamics, and functions of complex networks. It is used to study large complex networks represented in form of graphs with nodes and edges. Using NetworkX, one can load and store complex networks. The potential audience for NetworkX includes mathematicians, physicists, biologists, computer scientists, and social scientists. As of 22 February 2022, 83 papers appear on SCOPUS with the keyword "NetworkX". It allows computational reproducibility as the package is free software, and can be redistributed and/or modified under the terms of the BSD License. ("Overview — NetworkX 1.10 Documentation," n.d.).

3. RESULTS

This section first describes the openly available dataset of San Francisco city. Second the execution steps of the developed Jupyter Notebook are described. The results described in this section are computationally reproducible, which is considered to ease testing on other datasets as well as experimenting on a similar process.

3.1. San Francisco Road Network

This San Francisco Road network data set is obtained from the ACM SIGSPATIAL GIS Cup 2015 competition(*ACM SIGSPATIAL GIS Cup 2015*, n.d.). The reason for choosing this data set is that it is open data source. This road network dataset contains routes and nodes. Each row of the file *sfo_roads.txt* represents an edge and edges are one-way edges. If an edge represents a two-way street, edges are unsigned or with negative sign. For example: 123456, -123456.

Table 2. Properties of the San Francisco Road Data set

Characteristic	Result
Total Length of Roads	9743 (km)
Total Number of Nodes	42408
Total Number of Roads	96850

It should be noted that, there are islands in this dataset. Specifically, some nodes do not have access to other nodes because the start and end points of some paths are written incorrectly. The Figure 3 is serves as a good example of this.

Figure 3. Islands in the data set (Example node id: 48526416)

3.2. Finding the Shortest Path

In order to find the shortest path with the NetworkX package, the following Python packages need to be installed. First, as the road network data is originally recorded as a shapefile (.shp), it must be opened and stored for processing in Python. For this purpose, *GeoPandas* package is used. Second, in order to record the execution time of the whole process, the *time* package is used. Finally, to experiment on random start and end locations, the *random* package is used.

In order to use the "*shortest_path*" function of NetworkX, an array with three elements must be designed: i) the start point, ii) end point, and iii) the weight, which is the cost between the start and end nodes (points). In this paper's context, the weight value is the distance between two points.

At this point, the *shortest_path* function can be executed. If there are routes between these two points, the algorithm will select the least cost path. On the other hand, if there is not a route between the randomly selected two points, it will return "No path between 'Start Point ID' and 'End Point ID'". These steps are summarized in Figure 4.

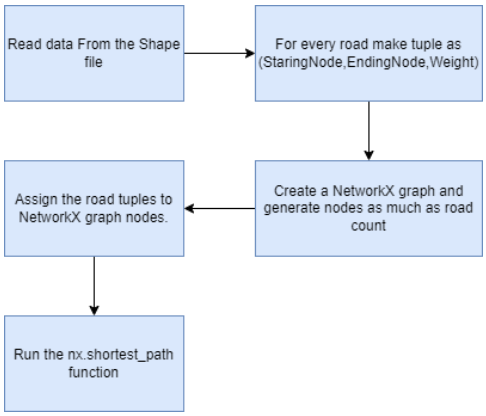

Figure 4. Methodology of using NetworkX shortest_path function with a shapefile.

The developed Jupyter Notebook handled the scenario where the start or end point is located in another island, as illustrated in Figure 3. Specifically, the developed function Random_Dijkstra_NetworkX() runs as intended as shown in Figure 5.

```
Random_Dijkstra_NetworkX()
```

```
' "ID" = 48523524 or  "ID" = 48523666 or  "ID" = 48523669 or  "ID" = 48523679 or  "ID" = 48523682 or  "ID" = 48523691 or  "I
= 48523683 or  "ID" = 48523718 or  "ID" = 48523726 or  "ID" = 48523729 or  "ID" = 48524359 or  "ID" = 48524360 or  "ID" = 48
379 or  "ID" = 312289200 or  "ID" = 312289201 or  "ID" = 48524378 or  "ID" = 811331450 or  "ID" = 863219895 or  "ID" = 96832
8 or  "ID" = 48524445 or  "ID" = 863189477 or  "ID" = 48524451 or  "ID" = 48524450 or  "ID" = 48524448 or  "ID" = 48524449 o
"ID" = 48524686 or  "ID" = 863188342 or  "ID" = 48524668 or  "ID" = 48524666 or  "ID" = 48524667 or  "ID" = 812154015 or  "I
= 48524676 or  "ID" = 48524680 or  "ID" = 48524723 or  "ID" = 312337972 or  "ID" = 48524726 or  "ID" = 48524721 or  "ID" = 4
4722 or  "ID" = 48524737 or  "ID" = 48524736 or  "ID" = 48524738 or  "ID" = 48524739 or  "ID" = 48524651 or  "ID" = 48530532
"ID" = 48530533 or  "ID" = 48530531 or  "ID" = 48530535 or  "ID" = 48530550 or  "ID" = 946460163 or  "ID" = 946460162 or  "I
= 947213705 or  "ID" = 48530547 or  "ID" = 48530556 or  "ID" = 816720638 or  "ID" = 886118424 or  "ID" = 850784614 or  "ID"
68054914 or  "ID" = 48530609 or  "ID" = 847345328 or  "ID" = 847345327 or  "ID" = 855080308 or  "ID" = 48530601 or  "ID" = 8
80309 or  "ID" = 276604275 or  "ID" = 968081241 or  "ID" = 48530628 or  "ID" = 968081242 or  "ID" = 816720639 or  "ID" = 294
512 or  "ID" = 968058482 or  "ID" = 48530664 or  "ID" = 48530669 or  "ID" = 816720715 or  "ID" = 872523076 or  "ID" = 872483
or  "ID" = 849382730 or  "ID" = 816720703 or  "ID" = 816720702 or  "ID" = 48530582 or  "ID" = 968053457 or  "ID" = 48530784
"ID" = 48530797 or  "ID" = 801727302 or  "ID" = 48530832 or  "ID" = 872566158 or  "ID" = 816720654 or  "ID" = 816720653 or
D" = 816720652 or  "ID" = 816720655 or  "ID" = 48531078 or  "ID" = 855119728 or  "ID" = 816720700 or  "ID" = 48531125 or  "I
= 19700537 or  "ID" = 816720698 or  "ID" = 872567356 or  "ID" = 19693746 or  "ID" = 872617564 or  "ID" = 967917458 or  "ID"
8531213 or  "ID" = 872576242 or  "ID" = 48532194 or  "ID" = 847346966 or  "ID" = 967913440 or  "ID" = 108479651 or  "ID" = 4
2221 or  "ID" = 953614894 or  "ID" = 48532418 or  "ID" = 962946926 or  "ID" = 48532426 or  "ID" = 974874185 or  "ID" = 48532
or  "ID" = 48532452 or  "ID" = 48532455 or  "ID" = 94350501 or  "ID" = 872644900 or  "ID" = 872662117 or  "ID" = 48532635 or
"ID" = 970042383 or  "ID" = 48532637 or  "ID" = 961641072 or  "ID" = 968262403 or  "ID" = 968262402'
```

(a)

```
Random_Dijkstra_NetworkX()
```

```
'There is no path between 48527259 and 48525579.'
```

(b)

Figure 5. An exemplar shortest path computation with an SQL output (a), failed execution (b)

In Figure 5, a route exists between the randomly generated start and end nodes. The function the function returns then return the SQL code to select the nodes that can be directly executed in QGIS for selection. On the other hand, there might not be a route between the randomly selected points, which is also handled as shown in Figure 5b. Further explanation is also provided to highlight which nodes are in different islands.

4. DISCUSSION

Although the details regarding a single randomly generated could be obtained, a more comprehensive experimental setup was required in order to have a better understanding on the distribution of run -times. Therefore, the last cell of the developed computational notebook allows a performance analysis on a number of random routes. Specifically, a user can specify the number of random routes to be generated, and their run-times are recorded.

A box-plot illustrates the variation of these run-times and further explanation is provided to reveal further insights to which paths resulted in the fastest / slowest computation. An exemplar run led to the results is illustrated in Figure 5.

Once a route has more nodes to visit, then its computation time increases. The median run time to determine a route is around 0.2 seconds on a computer having Windows 10 with a i7 processor 2.60 GHz and 8 GB RAM.

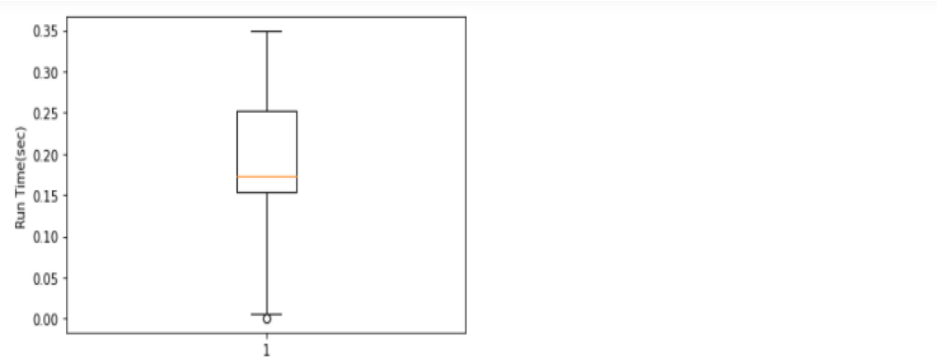

Figure 5. Run time analysis of 30 random routes with additional explanation on fastest and slowest run times

Furthermore, the resulting paths having the min/max run time is also stored as GeoJSON files under the *results/* folder. It is therefore straightforward to visualize these paths on QGIS by dragging and dropping the generated output files. A map produced with this approach is illustrated in Figure 6.

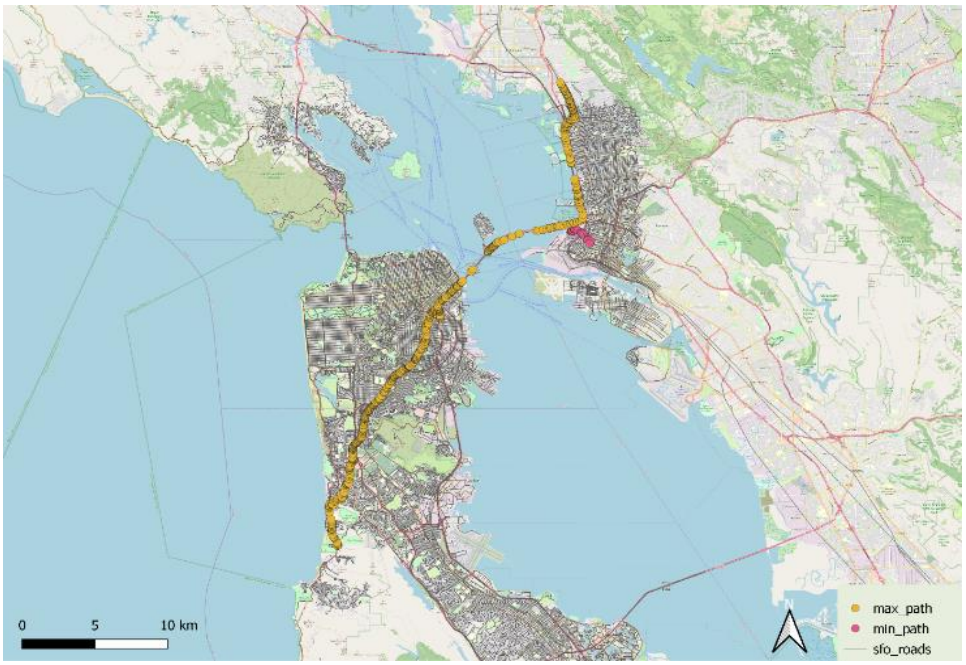

Figure 6. The least and the most time-consuming routes visualized in QGIS.

The ease of visualizing these paths allow a researcher/student to visually verify the outcome. Effective integration of the developed online material in a GIS Programming course should be investigated to further gain insights on its effectiveness (Anbaroğlu, 2021).

5. CONCLUSION

This paper developed an online education material to enable a researcher or analyst to understand how Dijkstra's algorithm works. The experimental setup is designed in a way to ease computational reproducibility. Specifically, open data and open-source software was relied on, and the developed code was implemented as a Jupyter Notebook that is hosted on GitHub. The future work will focus on the utilization of this educational material on a classroom setting, where students would be expected to find their own open dataset, and implement the Dijkstra's algorithm using a heap instead of relying on the readily available functions of NetworkX.

Author contributions

Muammer Özkan developed the code and contributed to the draft of the paper; Ayça Tabakoğlu and Ekin Gönenç Uygun wrote the draft of the paper and contributed to the code, and Berk Anbaroğlu did the supervision and completed the paper.

Conflicts of interest

There is no conflict of interest between the authors.

Statement of Research and Publication Ethics

Research and publication ethics were complied with in the study.

References

ACM SIGSPATIAL GIS Cup 2015. (n.d.). Retrieved from https://research.csc.ncsu.edu/stac/GISCUP2015/index.php

Ahuja, R. K., Mehlhorn, K., Orlin, J. & Tarjan, R. E. (1990). Faster algorithms for the shortest path problem. Journal of the ACM, 37(2), 213–223. https://doi.org/10.1145/77600.77615

Anbaroğlu, B. (2021). A collaborative GIS programming course using GitHub Classroom. Transactions in GIS, 25(6), 3132–3158. https://doi.org/10.1111/tgis.12810

Çalışkan, M. & Anbaroğlu, B. (2020). Geo-MST: A geographical minimum spanning tree plugin for QGIS. SoftwareX, 12, 100553. https://doi.org/10.1016/j.softx.2020.100553

GitHub—Banbar/shortest_path_Dijkstra. (n.d.). Retrieved from https://github.com/banbar/shortest_path_Dijkstra

Huber, S. & Rust, C. (2016). Calculate Travel Time and Distance with Openstreetmap Data Using the Open Source Routing Machine (OSRM). The Stata Journal: Promoting Communications on Statistics and Stata, 16(2), 416–423. https://doi.org/10.1177/1536867X1601600209

Kim, B. & Henke, G. (2021). Easy-to-Use Cloud Computing for Teaching Data Science. Journal of Statistics and Data Science Education, 29(sup1), S103–S111. https://doi.org/10.1080/10691898.2020.1860726

Kumar, R., Kumar, S. & Soni, A. (2021). Election prediction using twitter and GIS. 2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 306–311. https://doi.org/10.1109/ICACITE51222.2021.9404671

Necaise, R. D. (2011). Data structures and algorithms using Python. Hoboken, N.J: Wiley.

Overview—NetworkX 1.10 documentation. (n.d.). Retrieved February 10, 2022, from https://networkx.org/documentation/networkx-1.10/overview.html

Rout, R. R., Vemireddy, S., Raul, S. K. & Somayajulu, D. V. L. N. (2020). Fuzzy logic-based emergency vehicle routing: An IoT system development for smart city applications. Computers & Electrical Engineering, 88, 106839. https://doi.org/10.1016/j.compeleceng.2020.106839

Yu, L., Kong, D., Shao, X. & Yan, X. (2018). A Path Planning and Navigation Control System Design for Driverless Electric Bus. IEEE Access, 6, 53960–53975. https://doi.org/10.1109/ACCESS.2018.2868339

Zeng, W. & Church, R. L. (2009). Finding shortest paths on real road networks: The case for A*. International Journal of Geographical Information Science, 23(4), 531–543. https://doi.org/10.1080/13658810801949850

© Author(s) 2022.
This work is distributed under https://creativecommons.org/licenses/by-sa/4.0/