World Academy of Science, Engineering and Technology
International Journal of Urban and Civil Engineering
Vol:14, No:10, 2020

Optimal and Critical Path Analysis of State Transportation Network Using Neo4J

Pallavi Bhogaram, Xiaolong Wu, Min He, Onyedikachi Okenwa

***Abstract*—**A transportation network is a realization of a spatial network, describing a structure which permits either vehicular movement or flow of some commodity. Examples include road networks, railways, air routes, pipelines, and many more. The transportation network plays a vital role in maintaining the vigor of the nation's economy. Hence, ensuring the network stays resilient all the time, especially in the face of challenges such as heavy traffic loads and large scale natural disasters, is of utmost importance. In this paper, we used the Neo4j application to develop the graph. Neo4j is the world's leading open-source, NoSQL, a native graph database that implements an ACID-compliant transactional backend to applications. The Southern California network model is developed using the Neo4j application and obtained the most critical and optimal nodes and paths in the network using centrality algorithms. The edge betweenness centrality algorithm calculates the critical or optimal paths using Yen's *k*-shortest paths algorithm, and the node betweenness centrality algorithm calculates the amount of influence a node has over the network. The preliminary study results confirm that the Neo4j application can be a suitable tool to study the important nodes and the critical paths for the major congested metropolitan area.

***Keywords*—**Transportation network, critical path, connectivity reliability, network model, Neo4J application, optimal path, critical path, edge betweenness centrality index, node betweenness centrality index, Yen's *k*-shortest paths.

I. Introduction

THE transportation network is a critical component to provide a more reliable transport service for both people and goods. However, transportation networks are often disturbed by recurrent and non-recurrent perturbations, which result in various socio-economic consequences, e.g., blocked supply chain, increased individual travel costs, loss of life, and many more. Recurrent perturbations [1] such as traffic jams occur periodically in transportation networks and cumulatively cause a decline in the service level of the transportation network by degrading network components. Non-recurrent perturbations are rare and extreme disturbances such as earthquakes, tsunamis, terrorist attacks, etc., can cause failures of network components or interrupt network operations [1]. A reliable transport system can manage these issues to ensure an adequate administration level. Further, the higher the reliability transport networks can produce, the more excellent quality that transportation systems can provide. Meanwhile, connectivity reliability [2], travel time reliability [3], and capacity reliability [4] are another three main measures for evaluating the reliability of a network. These three measures are beyond the scope of this paper. In this paper, we first give a brief introduction of road transport as a network using the Neo4J application and to calculate the most critical and optimal path of that network using centrality and pathfinding algorithms.

Pallavi Bhogaram and Onyedikachi Okenwa are with California State University Long Beach, Long Beach, United States (e-mail: pallavi.bhogaram@student.csulb.edu, onyedikachi.okenwa@student.csulb.edu).

Dr. Xiaolong Wu and Dr. Min He are with Department of Engineering and Computer Science, California State University Long Beach, Long Beach, CA 90840 USA (e-mail: Xiaolong.Wu@csulb.edu, min.he@csulb.edu).

II. Road Transport as a Network

Road transport is an example of connectivity reliability which is the most straightforward measure of reliableness and shows whether the source and destination of a given origin and destination (OD) pair are related to a network matrix. Connectivity reliability is referred to as the probability of maintaining nodes connected in a transport network [2]. Terminal reliability [5] is a case of connectivity reliability that represents the flexibility of a road network where alternative routes are used once certain connections have been broken. During a functional expression, for a specific link, its connectivity can be expressed as a binary variable as shown in (1):

$$Xa = \begin{cases} 1, if\ link\ a\ connected \\ 0, if\ link\ a\ is\ disconnected \end{cases} \quad (1)$$

Transportation systems are commonly represented using networks as an analogy for their structure and flows. The California state has one of the highest numbers of highways and can be considered as an ideal case for our study of reliability. The highway transport system is transformed into a graph/network. Highways connecting the cities are as an edge and cities as nodes of a graph. This paper focuses solely on the road transport system for the case study, but this can apply to any transport and network modes. Finding the critical path [3] among the available roads between any two nodes (cities) and also finding the criticality of a path in the obtained network containing the shortest paths is interesting. The higher the criticality of a path is, the higher is its influence of the network, and any damage to that path might lead to serious havoc. When the California highway network is turned into a graph, the shortest paths were considered among the group. This fact clearly distinguishes the chosen path from other alternatives. Obtaining the shortest path is done by comparing the distance among all and then choosing the shortest one. The southern California network graph is developed using the Neo4j application as shown in Fig. 2.

III. Neo4J Application

Neo4J is the leading graph database. It is a high-performance graph store with all the characteristics a mature and robust database expects, such as a familiar query language and ACID (Atomicity, Consistency, Isolation, Durability) transactions [6]. The programmer works with flexible nodes and relationships of network structure rather than static tables yet enjoys all the advantages of the enterprise-quality database. Script to insert data and obtain the graph is written in cypher query language [7]. The Neo4j tools support plugin to run graph algorithms. Neo4j offers a growing, open library of graph algorithms as shown in Fig. 1 that are optimized for fast results. These algorithms reveal the hidden patterns and structures in the connected data around pathfinding and search, centrality and community detection with a core set of tested and supported algorithms [8]. For example, Neo4j graph algorithm libraries [8] provide a broader view of patterns and structures across all data and relationships are required to discover the overall nature of the networks and model the behavior of complex systems. *Pathfinding and search algorithms* [9] help find the shortest path or evaluate the availability and quality of routes. *Centrality algorithms* [10] determine the importance of distinct nodes in a network. *Community detection* [11] algorithms evaluate how a group is clustered or partitioned, as well as its tendency to strengthen or break apart.

Fig. 1 Neo4J Graph Algorithm Library

In this paper, we briefly discuss pathfinding and centrality algorithms as we use centrality algorithms to find the amount of influence a node has over the flow of information in a graph and pathfinding algorithm to find the critical and optimal paths in the graph. Fig. 2 shows the highway transport system graph generated using the Neo4J application with the table showing the most critical nodes based on the betweenness centrality ranges. The Los Angeles city centrality index range is between 200 and above, which makes it a critical node that matches the reality as LA is the most congested city among all other metropolitan areas in the US. In contrast, there are five cities (Glendale, Victorville, Lompoc, Escondido, Chula Vista) with the centrality index rage of 0, which makes them the optimal nodes as they are less commuted cities when compared to other cities in the graph shown in Fig. 2.

IV. Centrality Algorithm

A. Betweenness Centrality

Betweenness centrality is a way of identifying the amount of influence a particular node has over the flow of information in a graph. Nodes with high betweenness centrality have significant influence within a network by their control over data passing between others. They are also the ones who will be disconnected from the network between other nodes because they lie on the most significant number of paths. It is used to find nodes that serve as a bridge from one point to another. Equation (2) is used to calculate the node betweenness centrality. The breadth-first search algorithm is first used to calculate the shortest path between each pair of nodes in a connected graph. Then dividing the number of shortest paths by the total number of shortest paths in the graph for each node [12] gives:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)} \quad (2)$$

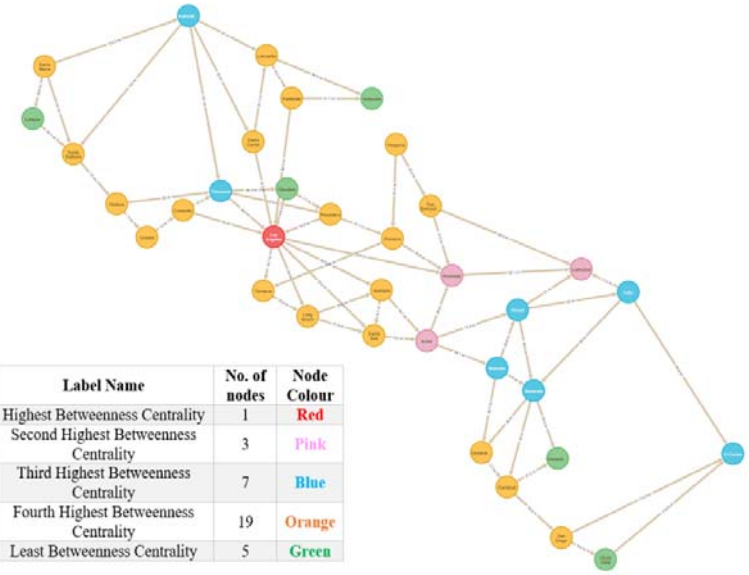
where V is the set of nodes, σ(s, t) is the number of shortest (s, t)-paths, and σ(s, t|v) is the number of those paths passing through node v other than s and t. If s=t, σ(s, t)=1, and if v∈s, t, σ(s, t|v)=0 [12]. Table I presents the node betweenness centrality index of all the nodes or cities mentioned in Fig. 1 which are calculated using the betweenness centrality algorithm supported by the Neo4j tool.

TABLE I
Betweenness Centrality Index

City	Centrality	City	Centrality
Los Angeles	293.52	Ventura	21.07
Riverside	174.23	Pasadena	19.78
Irvine	123.98	San Bernardino	13.63
Cathedral City	110.32	Santa Maria	12.02
Thousand Oaks	95.15	Carlsbad	11.92
Indio	90.84	Lancaster	10.67
Bakersfield	77.72	Oceanside	8.30
Temecula	58.44	Torrance	7.03
Murrieta	54.94	San Diego	6.00
Hemet	53.22	Long Beach	3.71
El Centro	51.58	Hesperia	1.17
Santa Clarita	44.51	Oxnard	1.08
Palmdale	37.41	Glendale	0.00
Anaheim	36.72	Victorville	0.00
Santa Ana	36.72	Lompoc	0.00
Pomona	29.75	Escondido	0.00
Camarillo	26.40	Chula Vista	0.00
Santa Barbara	24.18		

Betweenness centrality can be implemented as described in Algorithm 1 [13]. Betweenness centrality can be computed in O(nm + n2 log n) time and O(n + m) for weighted graphs. For unweighted graphs, running time reduces to O(nm).

It can be observed from the numbers shown in Table I that the vitality of a node in a graph is directly mapped to the critical index (CI). When the CI is higher, that particular node has a more significant influence on the network. Any damage to that specific node has a considerable impact on the entire network and might bring the transportation network to a halt. For instance, if the node 'Los Angeles' is removed from the above graph, many edges/paths connecting the city and different parts of the state via Los Angeles will not have any communication.



Centrality Index Range	Label Name	No. of nodes	Node Colour
200 & above	Highest Betweenness Centrality	1	Red
101 - 200	Second Highest Betweenness Centrality	3	Pink
50 - 100	Third Highest Betweenness Centrality	7	Blue
1 – 49	Fourth Highest Betweenness Centrality	19	Orange
0	Least Betweenness Centrality	5	Green

Fig. 2 California State Network Graph

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V; σ[s] ← 1;
    d[t] ← −1, t ∈ V; d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

B. Path Betweenness Centrality

Betweenness centrality of an edge e is the sum of the fraction of all-pairs shortest paths that pass-through e [13] as shown in (3):

$$C_B(e) = \sum_{s,t \in V} \frac{\sigma(s,t|e)}{\sigma(s,t)} \quad (3)$$

where V is the set of nodes, $\sigma(s, t)$ is the number of shortest (s, t)-paths, and $\sigma(s, t|e)$ is the number of those paths passing through edge e. The edge betweenness centrality index of all the edges is calculated using the edge betweenness centrality algorithm from the python code.

An edge with a high edge betweenness centrality index (BCI) serves as a bridge like a connector between two pieces of a network and the removal of which may influence the correspondence between numerous sets of nodes through the shortest paths between them. Fig. 3 represents a case of eight nodes in a network, and the red edge between two red nodes has the higher edge betweenness centrality. The removal of this edge results in a partition of the network into two densely associated subnetworks. Table II shows the BCI of all the routes mentioned in Fig. 2.

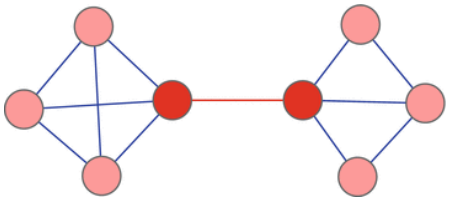


Fig. 3 Critical Path Example

V. Pathfinding Algorithm to Identify the Critical Path

The shortest path becomes the critical one when we analyze a source and destination pair with multiple paths among them. Along with classifying the shortest path, it is of great importance to calculate the criticality of the classified shortest path and to compare the importance of this path to the remainder of the network. In our case, Yen's *k*-shortest paths algorithm [14] is used to identify the shortest path based on the edge betweenness centrality where algorithm computes single-source k-shortest loop-less paths for a graph with non-negative relationship weights, i.e., edge BCI in our case. Yen's *k*-shortest paths algorithm is implemented using the Dijkstra

algorithm to find the shortest path and then proceeds to find the k-1 deviations of the shortest paths.

TABLE II
EDGE BCI OF CALIFORNIA STATE NETWORK GRAPH

Source	Destination	Edge BCI	Source	Destination	Edge BCI
Los Angeles	Riverside	**0.273**	Cathedral City	Hemet	**0.041**
Riverside	Cathedral City	**0.19**	Santa Barbara	Ventura	**0.04**
Cathedral City	Indio	**0.164**	Los Angeles	Glendale	**0.039**
Thousand Oaks	Los Angeles	**0.142**	Los Angeles	Pasadena	**0.036**
Indio	El Centro	**0.137**	Thousand Oaks	Pasadena	**0.035**
Irvine	Murrieta	**0.129**	Santa Barbara	Lompoc	**0.035**
Santa Clarita	Los Angeles	**0.119**	Hesperia	Pomona	**0.035**
Los Angeles	Palmdale	**0.109**	Temecula	Indio	**0.034**
Bakersfield	Thousand Oaks	**0.097**	San Bernardino	Cathedral City	**0.034**
Irvine	Hemet	**0.091**	Los Angeles	Long Beach	**0.032**
Riverside	Irvine	**0.086**	Lancaster	Palmdale	**0.029**
Santa Ana	Irvine	**0.084**	Bakersfield	Lancaster	**0.029**
Anaheim	Irvine	**0.084**	Hemet	Indio	**0.028**
Camarillo	Los Angeles	**0.083**	Hesperia	San Bernardino	**0.027**
Los Angeles	Anaheim	**0.082**	Pomona	Torrance	**0.025**
Los Angeles	Santa Ana	**0.082**	Santa Clarita	Lancaster	**0.024**
Thousand Oaks	Ventura	**0.075**	Carlsbad	San Diego	**0.024**
Santa Maria	Bakersfield	**0.071**	Santa Maria	Lompoc	**0.022**
Hemet	Temecula	**0.068**	Oceanside	Carlsbad	**0.021**
Bakersfield	Santa Clarita	**0.064**	Thousand Oaks	Camarillo	**0.015**
Bakersfield	Santa Barbara	**0.058**	Ventura	Oxnard	**0.013**
Murrieta	Oceanside	**0.056**	Thousand Oaks	Glendale	**0.013**
Pomona	Riverside	**0.051**	Santa Ana	Long Beach	**0.013**
Temecula	Escondido	**0.05**	Anaheim	Long Beach	**0.013**
El Centro	Chula Vista	**0.049**	Lancaster	Victorville	**0.012**
Oxnard	Camarillo	**0.048**	Torrance	Long Beach	**0.011**
Murrieta	Temecula	**0.048**	Temecula	Oceanside	**0.009**
Pasadena	Pomona	**0.047**	Hemet	Murrieta	**0.009**
Temecula	Carlsbad	**0.046**	San Diego	Chula Vista	**0.008**
Palmdale	Victorville	**0.045**	Carlsbad	Escondido	**0.007**
Los Angeles	Torrance	**0.045**	Santa Maria	Santa Barbara	**0.005**
El Centro	San Diego	**0.045**	Glendale	Pasadena	**0.005**
San Bernardino	Riverside	**0.042**	Anaheim	Santa Ana	**0.002**

TABLE III
NODE CENTRALITY INDEX

City	Centrality
A	15.0
F	11.0
B	0.0
C	0.0
D	0.0
E	0.0
G	0.0
H	0.0

The k-shortest paths algorithm is used to study alternative routing on road networks and recommends the top k-paths. Once the centrality index of each path is cumulated to find the total cost of the paths generated from Yen's *k*-shortest paths algorithm, the highest cost is the critical path, and the least cost is the optimal path.

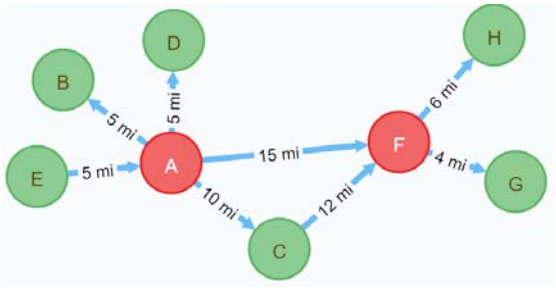


Fig. 4 Sample network graph

VI. CASE STUDY

In this case, we will consider a small dataset with eight cities to calculate the node betweenness centrality, edge betweenness centrality, and both critical and optimal path. The corresponding graph is generated using the Neo4J application shown in Fig. 4. Once the graph is generated the BCI of the node can be calculated, as shown in Table III, to find the critical nodes in the graph.

In Fig. 4, node A and node F are considered to be the most critical and have the highest centrality index. Meanwhile, the edge betweenness centrality is calculated using the Ulrik Brande algorithm [12] to find the most critical and optimal paths. Table IV shows the edge BCI for Fig. 4 using Python code.

TABLE IV
EDGE BCI OF SAMPLE NETWORK GRAPH

Source	Destination	Edge Betweenness Centrality
A	F	0.429
A	B	0.25
A	D	0.25
A	E	0.25
F	G	0.25
F	H	0.25
A	C	0.143
C	F	0.107

Based on the Table IV we can conclude that the edge AF is the most critical one. The Yen's k-shortest paths algorithm is used to find the optimal and critical path. The algorithm computes single-source k-shortest loop-less paths for the graph with non-negative relationship weights. Table V lists all shortest paths based on the edge betweenness centrality. The centrality index of each path is cumulated to find the total cost of the path through which the most critical and optimal path is obtained. From Table V we can conclude that the path E-A-C-F-H is the critical one and the path E-A-F-H is an optimal one.

TABLE V
CRITICAL AND OPTIMAL PATHS

Places	Costs	Total Cost
E, A, C, F, H	0.25, 0.25, 0.10700000077486038, 0.25	0.857
E, A, F, H	0.25, 0.25, 0.25	0.75

Since California state transport is not a simple graph, as shown in Fig. 2, this paper uses the edge betweenness centrality algorithm of Ulrik Brande [12] for every path. The algorithm takes the California state graph as an input i.e., source, destination, and distance values to the algorithm using NetworkX, a Python package for complex networks. The output of the algorithm is a dictionary with a tuple of two cities as key and path betweenness as the value. As shown in Table II, the higher the value of the edge BCI is, the higher is the influence on the network.

We consider Irvine as the source node and Long Beach as the destination node from the California network graph shown in Fig. 2. In order to calculate the critical and optimal path, we first need to list all possible shortest paths from Irvine to Long Beach using Yen's *k*-shortest paths algorithm. Table VI shows the top five results of the algorithm with the cumulated cost of edge BCI. The path Irvine --- Riverside --- Pomona --- Torrance --- Long Beach is the most critical as it has a total cost of 0.395, and the path Irvine --- Santa Ana --- Long Beach is the optimal one with the total cost of 0.099.

VII. CONCLUSION

This study mainly focused on generating a graph for the southern California transportation network using the Neo4J application. Centrality algorithms such as node betweenness centrality and edge betweenness centrality are used to identify the critical nodes in the network. Meanwhile, we were able to identify the critical and optimal paths among the developed transportation graph using Yen's *k*-shortest paths algorithm. The presented case study results clearly support that the Neo4j can be a suitable candidate to identify the important nodes and the critical paths for major congested metropolitan areas.

TABLE VI
TOP 5 SHORTEST PATH

Places	BCI	Total BCI
Irvine, Riverside, Pomona, Torrance, Long Beach	0.086,0.273, 0.025,0.010	0.395
Irvine, Anaheim, Santa Ana, Long Beach	0.086,0.002, 0.013	0.101
Irvine, Santa Ana, Anaheim, Long Beach	0.086,0.002, 0.013	0.101
Irvine, Anaheim, Long Beach	0.086,0.013	0.099
Irvine, Santa Ana, Long Beach	0.086,0.013	0.099

REFERENCES

[1] Yu Gu, Xiao Fu, Zhiyuan Liu, Xiangdong Xu, Anthony Chen, "Performance of transportation network under perturbations: reliability, vulnerability, and resilience" November 2019.

[2] I. M. Obeidat and S. Y. Berkovich, "Reliability of network connectivity," *2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, Ostrava, 2008, pp. 435-441.

[3] I. K. Isukapati and G. F. List, "*Using travel time reliability measures with individual vehicle data*," 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, 2016, pp. 2131-2136.

[4] W. J. Rueger, "*Reliability Analysis of Networks with Capacity-Constraints and Failures at Branches & Nodes*," in IEEE Transactions on Reliability, vol. 35, no. 5, pp. 523-528, Dec. 1986.

[5] C. Tanguy, "*Exact two-terminal reliability of some directed networks*," 2007 6th International Workshop on Design and Reliable Communication Networks, La Rochelle, 2007, pp. 1-8.

[6] "*What Is a Graph Database and Property Graph | Neo4j*", *Neo4j Graph Database Platform*. (Online). Available: https://neo4j.com/developer/graph-database/. (Accessed: 04- Sep- 2019).

[7] "*Cypher Query Language Developer Guides & Tutorials", Neo4j Graph Database Platform*. (Online). Available: https://neo4j.com/developer/cypher-query-language/. (Accessed: 22- Sep- 2019).

[8] M. Needham and A. Hodler, "*Graph Algorithms in Neo4j: Neo4j Graph Analytics*", *Neo4j Graph Database Platform*, 2018. (Online). Available: https://neo4j.com/blog/graph-algorithms-in-neo4j-neo4j-graph-analytics/. (Accessed: 20- Oct- 2019).

[9] Mark Needham, A., n.d. *Graph Algorithms*. (Online). O'Reilly Online Learning. Available at: <https://www.oreilly.com/library/view/graph-algorithms/9781492047674/ch04.html> (Accessed: 6 November 2019).

[10] Amy E. Hodler, M., 2018, "*A Comprehensive Guide to Graph Algorithms in Neo4j*", (ebook) Neo4j, p.34. Available at: <https://go.neo4j.com/rs/710-RRC-335/images/Comprehensive-Guide-to-Graph-Algorithms-in-Neo4j-ebook-EN-US.pdf> (Accessed: 11 February 2020).

[11] "Betweenness_centrality - NetworkX 1.10 documentation", Networkx.github.io, (Online). Available: https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.centrality.betweenness_centrality.html. (Accessed: 19- Oct- 2019).

[12] Ulrik Brandes, "*A Faster Algorithm for Betweenness Centrality*", *Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163-177, 2001.

[13] "Edge_betweenness_centrality — NetworkX 1.10 documentation", *Networkx.github.io*. (Online). Available: https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.centrality.edge_betweenness_centrality.html#networkx.algorithms.centrality.edge_betweenness_centrality. (Accessed: 23- Sep- 2019).

[14] "9.4.6. The Yen's *K*-shortest paths algorithm - 9.4. Path finding algorithms", *Neo4j.com*. (Online). Available: https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/yen-s-k-shortest-path/. (Accessed: 17- Oct- 2019).