IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

Congestion Control in Spatial Networks in Case of Disasters

Saumya Pandey[1], R. K. Singh[2]

Student, CSE, KNIT, Sultanpur, India

Associate Professor, KNIT, Sultanpur, India

Abstract: Large numbers of algorithms have been proposed to solve shortest path query problems for static or time-dependent spatial networks; however, these algorithms do not perform well to find the nearest shelter with fastest paths in disaster situations. In disasters, path computed through existing algorithms and saved as the fastest might become damaged. ONSC approach provide optimal path in disaster situation but do not deal with congestion control. To solve this problem, this paper proposes a method to reduce the travelling time with an existing dynamic network model, which is called an event-dependent network, to represent a spatial network in a disaster which help the people to choose the optimal path by giving weight-factor(in percentage) of the congestion in the road network.

Index Terms: congestion control, event dependent network, path planning, disaster management.

I. INTRODUCTION

One of the primary and critical information technology tasks in disaster management is helping people escape from danger and, more importantly, reach available shelters for safety. To accomplish this task, flexible path planning that can adapt to unpredictable and varying circumstances during disasters is crucial. General static spatial networks are defined as networks with fixed edge costs. The static fastest-path approaches make the simple assumption that the traveling time for each edge of a road network is constant. In reality, the traveling time on a road segment depends on the level of traffic congestion. Therefore, time-dependent networks are used to model traffic situations in which the cost of traveling on a road varies as a function of time; thus, the traveling time on a road is determined by the arrival time of using the road. Because of the traveling time functions of roads, the solution to the shortest-path planning problem for time-dependent networks is to compute the shortest path between a source point and a destination point. This paper make use of an event-dependent spatial network for modeling disaster situations in which a road becomes blocked because of various unpredictable events such as broken roads, floods, or car accidents. Road changes are unpredictable and frequent; therefore, the fastest paths cannot be effectively precomputed. In addition, the destinations (i.e., the nearest shelter) for people are also unknown and may change when roads become impassable. These types of unpredictable events may also cause the original destination to become unreachable or farther away than other destinations. Therefore, the path planning of event-dependent spatial networks in disasters should resolve the following problems simultaneously: 1) promptly recognizing people who are affected by disaster; 2) locating the newly established nearest shelters (if necessary); and 3) computing the fastest paths from the current position to the destination. This problem is called the event-dependent fastest-path (EDFP) problem.

The procedure of existing ONSC, comprises two phases: a system initialization phase and a running phase. At the system initialization phase, geographic city maps, which are referred to as spatial networks, are transformed into an event-dependent graph (EDG) by the server. An EDG is a graph on which vertices represent geographical coordinates of the city maps, and edges are the roads connecting any two geographical coordinates. The locations of shelters on the maps, such as hospitals, churches, and schools, are designated as vertices called sources in the EDG. The higher the number of vertices is, the more accurate the EDG is. For each vertex in the EDG, the server computes its nearest shelter and the fastest path between them. The computed fastest paths are represented by the NFG (Navigation Forest Graph) structure and stored in the NFG database.

When a mobile client executes its mission, it contacts the server to obtain the required NFG information. A mobile client may have the ability to flexibly determine the range of the NFG to be downloaded but cannot get the full NFG and client can also choose between less congestion and less distance depending on the congestion in the road network. When the spatial network is static or the fastest paths to the nearest shelter are unchanged, the mobile clients can directly follow the fastest paths stored by the NFG to reach the nearest shelters. If the control center receives damage of any node in the NFG from any source like updates from traffic police, the server executes the running phase which provides the new path to mobile client by reconstructing the path according to the newly constructed NFG.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

II. PROPOSED WORK

Existing systems are finding the optimal path to nearest available shelters by using the different algorithms and Path Reconstruction Algorithms and giving the highest performance in comparison to available methods for route finding in case of disaster situation. But there can be one problem with the existing system: In case of large numbers of people on the same route are using this method when a disaster has occurred, then all of them will be suggested the same optimal path that will cause a lot of load on the suggested route. In that case the person should be able to choose between optimal path and load on the path.

So we are extending the existing systems for the shelter computation in case of disaster by giving the load matrix to the existing system and asking the client for the weight percentage of the load factor and also for the distance factor in the form of percent. Client can give zero weight percentage to any of the load or distance factor and get the result based on choice.

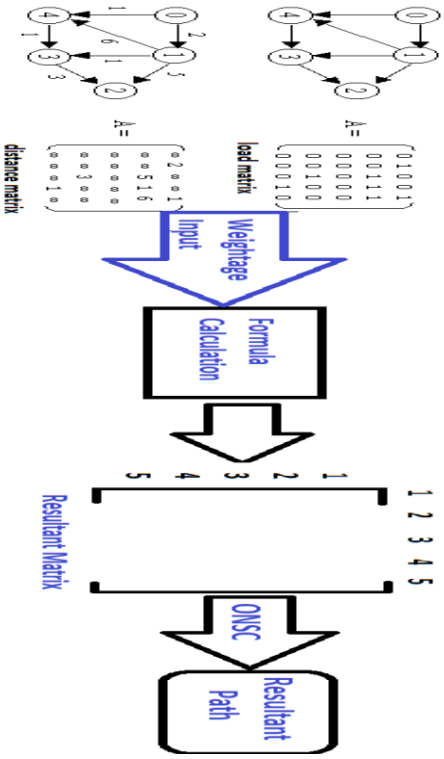

Figure 1 Congestion Control with ONSC

We get the result on the basis of simple formula

(m_adjMatrix[i][j] * ((100.0 - m_loadWeight) / 100.0)) + (m_loadMatrix[i][j] * (m_loadWeight / 100.0))

Where

m_adjMatrix[i][j] is the value of the distance matrix

m_loadWeight is the choice of user(weightage for the load consideration)

m_loadMatrix[i][j] is the value for the load matrix

This change is completely flexible as client has right to decide what he/she want less congestion or less distance by giving the appropriate weight factor(m_loadWeight) as input. This change helps in load balancing and saves time also while maintaining the performance level. The Fig. 1 is representing the way how changes are actually working. Proposed work is a simulation on the existing work in case load is very high in the road network so that client could choose between the shortest path and less load by providing weightage to the load matrix as given in the Fig. 1. Here the load matrix is fixed.

III. ALGORITHMS USED FOR CONGESTION CONTROL IN CASE OF DISATER APPROACH

This section first presents the data structure for recording essential information of a vertex in NFG. This structure is used to compute the new fastest paths. Two algorithms are presented to identify the damage range and the recovery vertices when some of the vertices in the NFG become damages. A damage-range reconstruction method has been to compute the new fastest paths in the damage range.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

A. NFG Vertex Data Structure

Each NFG vertex v_i has four fields : ID, Type, Neighbor, and FastestPath, shown in Fig. 2.

• **ID** is used for vertex identification.
• **Type** is to categorize between the root and general vertex.
• **Neighbor** is used to store the set of adjacent nodes in the EDG and the edge costs between each of them and v_i.
• **FastestPath** is used to store the information of the NT, including the root ID (RID), the previous and the next vertices along the fastest path, and the total path cost.

The NFG vertex data structure has two important properties. One feature is the use of a bidirectional link to acknowledge each vertex with the next (i.e., outgoing) vertex and the previous (i.e., incoming) vertex along the fastest path. The other feature is the attributing of each vertex to only one NT such that no two NTs overlap. Because of these two critical features, when an event occurs, the computation time of reconstructing the NFG can be reduced, and the fastest path can be rapidly determined. We describe how these features facilitate NFG reconstruction later.

NFG vertex v_i									
ID	Type	Neighbor		FastestPath					
i	gen	ID	Edge cost	RID	Path cost	Outgoing vertex	Incoming vertices		
		v_a	$\|(v_a, v_i)\|$	$r_{(j)}$	$\|(r_{(j)}, v_i)\|$	v_a	v_x	...	v_y
		...	...						
		v_y	$\|(v_y, v_i)\|$						

Figure 2 NFG Data Structure

A simple method for constructing the NFG is to use a one-by-one spreading algorithm. Assume that the EDG contains multiple roots. The one-by-one spreading algorithm determines the fastest paths for all vertices from a single root for each root. In other words, for each round, only one root runs the Dijkstra algorithm and stops when all vertices in the EDG have been visited. Values for NFG vertex v_i, such as RID and path cost, are initialized as null and infinity.

They are updated only when the stored path cost is higher than the values generated in new spreading rounds. Finally, the values of all the vertices in the EDG are determined, and each vertex vi is attributed to $NT(r_{(j)})$, where

$$|p[v_i, r_{(j)}]| = \min_{\forall r_{(k)} \in NFG}\{|p\,[vi, r_{(k)}]|\} \qquad (1)$$

Another method is using the simultaneous-spreading algorithm, where multiple Dijkstra algorithms are executed with all roots simultaneously, and the spreading of each root stops when it visits a vertex v_i with a value lower than that which it is going to donate.

B. DRVF Algorithm

As mentioned, in addition to making fastest paths invalid, impassable edges might also cause shelters to become unreachable or lead to other shelters being closer. The objective of the DRVF algorithm is to determine the damage range $D(v_{im(i)})$ and the recovery-vertex set $R(v_{im(i)})$ simultaneously.

Further section reports the use of the recovery-vertex set to determine the fastest paths and the new nearest shelters of vertices in damage range $D(v_{im(i)})$. The pseudo code of DRVF is provided in

Algorithm 1. The inputs of the DRVF algorithm are impassable vertex $v_{im(i)}$ and the NFG. The outputs of the DRVF algorithm are damage range $D(v_{im(i)})$ and recovery vertex set $R(v_{im(i)})$.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

Algorithm 1: Damaged and Recovery Vertex Finding

Input: Impassable vertex $v_{im(i)}$ and NFG
Output: $D(v_{im(i)})$, $R(v_{im(i)})$

1 Put $v_{im(i)}$ into queue Q, insert $v_{im(i)}$ into $D(v_{im(i)})$
2 **while** Q *is not empty* **do**
3 $v_q \leftarrow$ pop Q
4 **for** *all* $v_p \in$ *the set of neighbor vertices of* v_q **do**
5 **if** v_p *is the incoming vertices of* v_q **then**
6 Put v_p into Q, add v_p in $D(v_{im(i)})$
7 **else if** v_p *is NOT the outgoing vertex of* v_q **then**
8 Insert v_p into $R(v_{im(i)})$
9 Remove duplicate copies of vertices in $R(v_{im(i)})$
10 Delete v_i from $R(v_{im(i)})$ if $v_i \in D(v_{im(i)}) \cap R(v_{im(i)})$

Figure 3 DRVF Algorithm

Two crucial properties of the DRFV algorithm are that the parent of a damaged vertex in the NFG must be a damaged vertex and that a neighbor of a damaged vertex in the EDG is either a damaged vertex or a recovery vertex. Based on the first property, the DRVF uses $v_{im(i)}$ as the first damaged vertex (see Step 1) and recursively visits all the neighbors of damaged vertices (see Steps 2–8) until all the descendants of $v_{im(i)}$ in the NFG are checked. A neighbor v_p of a damaged vertex v_q is identified as a damaged vertex if its next outgoing vertex is v_q (see Steps 5 and 6). Otherwise, except for the next outgoing vertex v_q, v_p can be selected as a potential recovery vertex (see Steps 7 and 8) on the basis of the second property. Finally, the recovery vertices are produced by removing duplicate copies and the vertices that are identified as damaged from the potential recovery vertices (Steps 9 and 10).

C. Damage-Range Reconstruction

Damage-range reconstruction is performed to determine the fastest paths for vertices in the damage range. Vertices in the recovery-vertex set contain all the outgoing connections of vertices in the damage range. On the basis of this property, the damage-range reconstruction method first initializes the values of Fastestpath, such as RID and path cost, as null and infinite in the NFG vertex structure for vertices in the damage range. Because the fastest paths of recovery vertices are not deleted, the NFG can be reconstructed by simply re-computing the fastest paths of damaged vertices to their nearest recovery vertices. Therefore, the damage-range reconstruction method considers all the recovery vertices as roots with the beginning costs and runs the simultaneous-spreading algorithm for all the damaged vertices to determine the fastest paths. The beginning weights here are the path costs between the recovery vertices and their nearest roots. It is proven later that damage range reconstruction can obtain the global fastest path from a damaged vertex to its nearest root. Algorithm2 presents the damage-range reconstruction method.

Algorithm : Damage-Range Reconstruction

Input: $D(v_{im(i)}), R(v_{im(i)})$
Output: Updated partial NFG

1 Initialize the values of all the vertices in $D(v_{im(i)})$
2 Set all the vertices in $R(v_{im(i)})$ as roots
3 Use roots to run *Simultaneous spreading* algorithm for all the vertices in $D(v_{im(i)})$
4 **for** *each damaged vertices* **do**
5 Select the new nearest root and the fastest path that yield minimum cost:
6 $v^* = \min_{v \in \mathbf{R}}\{|p[u,v]| + |p_f[v,r]|\}$
7 where **R** is the recovery-vertex set, r is the root of recovery vertex v

Figure 4 Reconstruction Algorithm

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

V. PERFORMANCE COMPUTATION

This section discuss about the experimentation environment. We are the comparing the proposed work with Dijkstra algorithm and simultaneous-spreading algorithms.

A. Experimental Environment

The experiments shown further were implemented in C++ language by using Visual Studio 2010-32 bits and executed on a laptop computer with Pentium Processor 2.13 –GHz processor, 8GB of memory and running Windows 7 32 bits. We have done simulation using taking adjacency matrix as input and also a congestion matrix to reduce the travelling time in case of excess congestion in the road network; we modeled the event occurrence by randomly selecting some of nodes in the NFG as impassable node.

B. Results

Performance of the proposed work is completely dependent on the ONSC procedure with some additional computation cost to calculate the effective matrix for congestion control. To evaluate the Dijkstra, Spreading Algorithms, preprocessing is defined as time to preprocess the data for speeding up online responses, "query" as response time of the online fastest-path query when map is static, and "reconstruction" as the response time of the online fastest-path query when changes in the road network occurs. Table 1 is representing the time complexity of different shortest path computing algorithms. All mentioned algorithm such as Dijkstra, Spreading Algorithm, All pair shostest path computing Algorithm cannot be use in the case of disaster still having higher time complexity. This ONSC approach used to control congestion is having (mlogm+m) time complexity where m is the number of vertices in damaged area when an event has occurred. Maximum value of m can be n/2 where n is the no of nodes in the EDG.

Table 1 Complexity of different Algorithms

Algorithm	Complexity
Dijkstra	O(nlogn)
Spreading	O(nlogn)
APSP	O(n^3)
ONSC	mlogm+m

Table 2 is representing the complexity of different algorithm with different values of n, where n is the no of nodes. Complexity have been computed according to the Table 1. Computation of ONSC is done in worst case in Table 1 by taking m=n/2 as mentioned before.

Table 2 Complexity of algorithms with changes in graph size

Algorithm	Dikastra	APSP	ONSC
n=10	10	1000	3.4948
n=50	84.9485	125000	34.9485
n=100	200	1000000	84.9485
n=500	1349.485	125000000	599.485
n=1000	3000	1000000000	1349.485

As this paper is working on the future scope of the ONSC [6] procedure so it has all the advantage of the existing system. This approach takes less than 3 to 4 ms to compute the nearest shelter and its shortest path which is faster than other existing approaches.

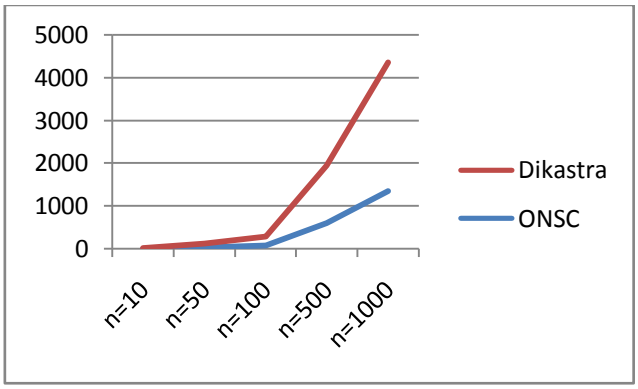

Figure 5 Comparison of dijkstra and ONSC

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

V. CONCLUSION

In this paper, we have used a dynamic network model called the event-dependent network to make simulation for load balancing in unpredictable networks such as spatial networks during disasters to reduce the travelling time. Unlike other spatial networks, the edge weights of this network are unpredictable and change rapidly rather than being static or time varying. To address the fastest-path problem in an event-dependent network, we utilized ONSC approaches to dynamically and promptly respond to queries for the nearest shelter with the fastest paths with the advantage of load balancing. NFG not only stored the fastest paths of the static network but also effectively sped up the calculation of the fastest path when the network changed frequently. ONSC with DRVF algorithms was developed to address various system restrictions such as computing power and memory space. Compared with other fastest-path studies, our experiments with real-world spatial networks and simulated spatial networks show that ONSC approaches substantially outperform competitors in storage and response time. In conclusion, our approaches are applicable to real-world spatial networks during disasters. We next intend to extend this study in two directions. One is to support real-time network recovery such as providing temporary alternative roads or resolving congestion during disasters by taking dynamic load matrix. The other is to extend our work to the global path-planning problem.

REFERENCES

[1] I. Dumitrescu and N. Boland, "Improved preprocessing, labeling and scaling algorithms for the weight constrained shortest path problem," Networks, vol. 42, no. 3, pp. 135–153, Oct. 2003.
[2] A. Efentakis, D. Pfoser, and Y. Vassiliou, "SALT. A Unified Framework for All Shortest-Path Query Variants on Road Networks," arXiv preprint, arXiv:1411.0257, 2014.
[3] M. Thorup,"Undirected single-source shortest paths with positive integer weights in linear time," J. ACM, vol. 46, no. 3, pp. 362–394, May 1999.
[4] L. Zhao and M. Gao, "Node early-fixing: A practical speedup technique for A∗ algorithms," J. Math., Statist. Oper. Res., vol. 2, no. 1, pp. 98–102, May 2013.
[5] R. Bauer et al.,"Combining hierarchical and goal-directed speed-up techniques for Dijkstra algorithm," in Proc. Int. WEA, 2008, vol. 5038, pp. 303–318, ser. LNCS.
[6] Pei-Hsuan Tsai, Chun-Lung Lin, and Jyun-Nan Liu, "On-the-Fly Nearest-Shelter Computation in Event-Dependent Spatial Networks in Disasters." vol. 65, no. 3, pp.1109-1115,March 2016.
[7] Z. Yang and M. Yuan, "Route selection model in emergency evacuation based on quasi-user optimum dynamic traffic assignment," in Proc. Int. Conf. Intell. Comput. Technol. Autom., 2010, vol. 3, pp. 240–243.
[8] S. Chechik, "Approximate distance oracle with constant query time," in Proc. 46th Annu. ACM Symp. Theory Comput., 2014, pp. 654–663.
[9] U. Karisruhe and C. Zaroliagis, "Geometric containers for efficient shortest-path computation," J. Exp. Algorithmics, vol. 10, no. 1.3, pp. 1–30, 2005.