A Dijkstra-based algorithm for selecting the Shortest-Safe Evacuation Routes in dynamic environments (SSER)

Angely Oyola, Dennis G. Romero, and Boris X. Vintimilla

Faculty of Electrical and Computer Engineering, Escuela Superior Politécnica del litoral, Km 30.5 via Perimetral, P.O. Box 09-01-5863, Guayaquil-Ecuador
{ajoyola;dgromero;boris.vintimilla}@espol.edu.ec

Abstract. In this work it proposed an approach to addressing the problem to find the shortest-safe routes in buildings with many evacuation doors and where the availability status of internal areas could be changed by different kind of sensors. We present two advantages over the common use of Dijkstra's algorithm, related to the problem of obtaining evacuation routes: 1) Fast search of the shortest-safe evacuation route to multiple exits with a backward approach and 2) Support to dynamic environments (graph with variable vertex availability). Four Dijkstra-based algorithms were considered in order to evaluate the performance of the proposed approach, achieving short times in evacuation to multiples exits.

Keywords: Dijkstra, Dynamic environments, Shortest-Safe

1 Introduction

Determining short routes has been a topic of interest in different scenarios and application areas. Currently different proposals based on *Dijkstra's* algorithm, among others, are intended to determine short routes in applications such as routing protocols, transport routes, evacuation systems, among others [1][2].

In the scope of evacuation systems, algorithms based on *Dijkstra* have been developed to calculate evacuation routes, seeking to optimize resources, simplifying methods and obtaining acceptable response times even with large amounts of data, addressing problems where is necessary to analyze all possible evacuation routes, indicating a single exit as destination. However, traditional implementations of these algorithms are impractical in dynamic environments, where the availability of evacuation routes may vary suddenly, for example, in applications requiring continuous monitoring. These dynamic environments can be found in buildings with several exit doors, where it is common to find some blocked areas for maintenance, cleaning or security concerns. A proper monitoring solution should ensure that, in an emergency situation, people can be driven towards alternative routes but avoiding suggestions that lead people to dangerous zones and also to perform unnecessary calculations for this purpose[3]. The present

work aims to contribute to the solution of calculating the shortest-safe routes, oriented to be applied in dynamic environments.

The manuscript is organized as follows. Related works are presented in Section 2. Then, SSER approach (Shortest-Safe Evacuation Routes) is introduced in Section 3, which consider the backward searching from multiple exits, storing meta-data in vertex in order to select efficiently a shortest-safe evacuation route considering dynamic environments. Experimental results are provided in Section 4. Finally, conclusions are given in Section 5.

2 Related work

Techniques like Dijkstra [4], Floy-Warshall [5] and Bellman Ford [6] are the most commonly used algorithms for evaluating shortest paths, considering their simplicity, being Dijkstra the one with the best run-time on extensive graphs [1] [2]. These algorithms have been used to propose novel approaches for route planning in different application contexts such as buildings, sensor networks, vehicle congestion, among others.

In this sense, we review some studies related to the problem of route planning which includes finding the shortest path, reducing resource consumption, processing times and getting safe solutions.

The work presented by Artmeier et al. in [7] addresses the problem of route planning for electric vehicles, where an important aspect is to minimize energy consumption. Mohring et al. in [8] consider Dijkstra's algorithm for the point-to-point shortest path problem in large and sparse graphs with a given layout. Their work examines partitioning algorithms from computational geometry and compares the impact on the speed-up of the shortest path algorithm. The first contribution of the study is to evaluate whether partitioning algorithms from computational geometry can be used for the arc-flag approach, comparing the results with METIS [9]. As a second contribution, Mohring et al. present a multi-level version of arc-flags that produces the same speed-up with lower consumption.

On the other hand, Rosenthal [10] considers cooperative games for addressing the problem of finding the shortest path from a specific source to all other nodes in the network. This paper applies Dijkstra's algorithm in the development of a rail system for metropolitan areas to bring passengers in outlying stations from a central terminal as fast as possible. The initial contribution of this work is to define the shortest path games in the context of designing commuter rail lines for metropolitan areas. Such design is focused not only on building a network that minimizes travel times but also for reducing capital costs in a fair manner.

In [11] the Dijkstra's approach has served on the development of algorithms to find short routes in buildings with a single evacuation door and with always-available areas. However, nowadays it's not enough to find short routes but also finding routes that take less time to be followed, less effort or resource consumption and safer. There are some situations according to the size and building configuration that requires several exit doors, as well as areas that can

be temporarily or permanently disabled. Many studies have been conducted to develop systems that could exclude unsafe paths and calculate the shortest-safe path from multiple starting points to multiple exit points during an emergent situation [1] [12].

3 Proposed approach

Regarding the problem of determining the shortest evacuation routes, Dijkstra's algorithm is one of the most popular if compared with *Bellman Ford* [2] and *Floyd-Warshall* [1], due to its simplicity and time complexity $O(n^2)$. Dijkstra's algorithm calculates the shortest routes considering a single vertex as source, in the context of this work it is equivalent to a single evacuation route and escape areas always available. However, nowadays many buildings have several evacuation routes and the availability of escape areas varies at different time intervals [13]. To overcome the difficulties associated with dynamic environments (variable availability of escape doors and evacuation routes), the SSER approach (*Safe and Short Evacuation Routes*) uses its own structures to represent different areas of the building, aisles and evacuation doors. In this sense, two main structures are used (Fig. 1): *vertex*, corresponding to escape areas of the building (offices or places with access to the main aisles or escape routes) and the *edges* or aisles connecting these areas (Table 1).

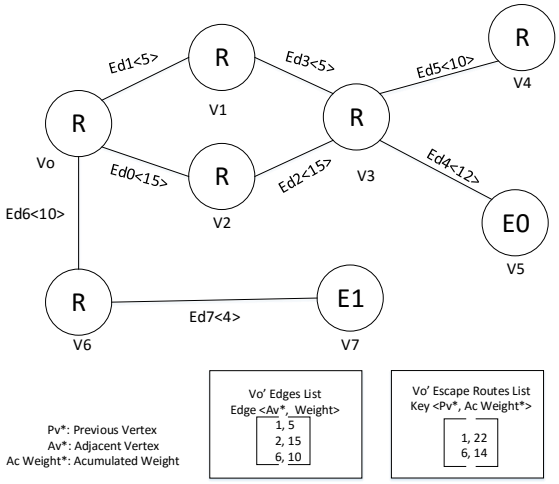

Fig. 1. Graph representing the building's evacuation areas (R), evacuation paths (Ed) and evacuation doors (E).

Each *vertex* has two structures (type *vector*) to represent the list of paths related to each escape area and the list of aisles in direct connection with them, getting as result the adjacency matrix for calculating escape routes (Table 2).

Table 1. Vertex Properties

Property	**Description**
Type	R (room), E (escape)
Access status	A: Active, I: Inactive
Edge list	vector $< Dv*, weight >$

Table 2. Adjacency Matrix corresponding to Fig. 1

Vertex	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	-	5	15	-	-	-	10	-
1	5	-	-	5	-	-	-	-
2	15	-	-	15	-	-	-	-
3	-	5	15	-	10	12	-	-
4	-	-	-	10	-	-	-	-
5	-	-	-	12	-	-	-	-
6	10	-	-	-	-	-	-	4
7	-	-	-	-	-	-	4	-

Thus, the building is represented by a set of vertices interconnected by different adjacent arcs. Considering the structures and properties of the vertices, adjacent arcs, evacuation routes and how the graph is examined, we can mention two main advantages over the common use of Dijkstra's algorithm, related to the problem of obtaining evacuation routes [1]:

1) Fast search of the shortest-safe evacuation route to multiple exits with a *backward* approach: Dijkstra's algorithm seeks the shortest path among two pair of vertices given a source vertex. In the context of evacuation systems, this type of *forward* approach of Dijkstra's algorithm (Fig. 2) does not differentiate between vertices representing available evacuation doors and vertices representing areas that need to be evacuated.

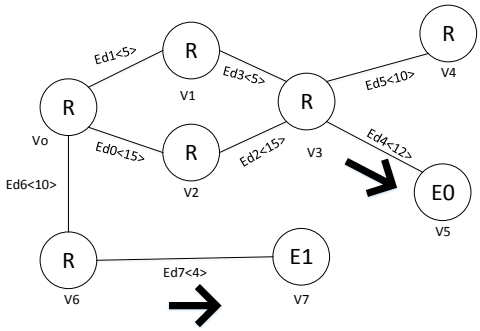

Fig. 2. Dijkstra Forward Search

Considering that most buildings have multiple evacuation doors (target of evacuation routes), the advantages of using SSER approach over common Dijkstra are explained on the base of the following Dijkstra's pseudocode [14].

Algorithm 1 Dijkstra's algorithm

Require: a source vertex a
Require: a target vertex z
Ensure: the weight of edge (i, j) is $w(i, j) > 0$
Require: $L(z)$ the shortest route from a to z
1: **procedure** *dijkstra* (w, a, z, L)
2: $L(a) := 0$
3: **for all** *vertices* $x \neq a$ **do**
4: $L(x) := \infty$
5: **end for**
6: $T :=$ set of all vertices
7: //T is the set of vertices where the shortest distance to a
8: //has not been calculated.
9: **while** $z \in T$ **do**
10: **begin**
11: Choose $v \in T$ with minimum $L(v)$
12: $T := T - V$
13: **for** each $x \in T$ adjacent to v **do**
14: $L(x) := \min L(x), L(v) + w(v, x)$
15: **end for**
16: **end**
17: **end while**
18: **end** *dijkstra*

The 4^{th} line in the pseudocode is invoked $n-times$ considering all vertices in the graph. The 14^{th} line is also invoked n-times to explore the adjacent vertices, it could be n vertices in the worst scenario, that's why Dijkstra's algorithm has a time complexity of $O(n^2)$. In this sense, if we keep the common Dijkstra's approach the algorithm should be iterated n-times for each vertex representing an available door for evacuation, and m-times for each vertex representing an evacuation door, where n is the number of available doors for evacuation and m are all the evacuation doors. Considering this, the time complexity to determine an evacuation route using common Dijkstra is $O(m \times n^3)$, which is inefficient on dynamic environments or real time applications. For this reason, SSER uses a backward approach (Fig. 3), which involves to consider evacuation doors as the starting point of the route, towards the different vertices representing evacuation areas. It is just necessary to invoke Dijkstra $m - times$ in order to calculate the shortest-safe evacuation route per each vertex in the graph. It leads to an increased efficiency for finding evacuation routes with a time complexity of $O(m \times n^2)$.

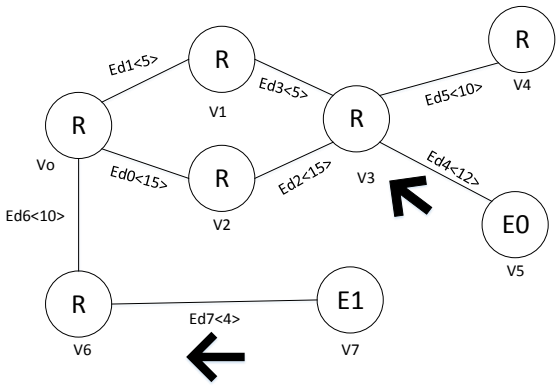

Fig. 3. SSER Backward Search

In order to achieve this, SSER approach uses structures in each of its vertices for storing the short and safe previous neighbor vertex (considering a backward search) that allows to construct the shortest-safe path to each available evacuation doors in the building. Then, among the all available routes it is selected the shortest one. The data type of the structure for the evacuation routes storing is *vector* and consists of a pair: <id, accumulated distance>, where *id* is the previous vertex identifier of the solution route, and the *accumulated distance* is the sum of existing weights to reach the current vertex. This structure supports exceptions that allows to consider another alternative when the shortest route is not safe.

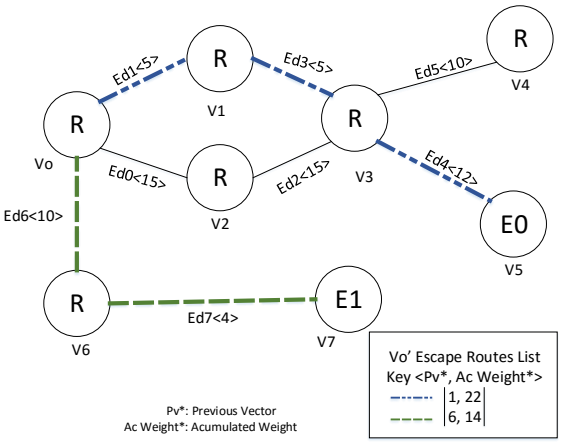

Fig. 4. Illustration of two possible evacuation routes

2) Support to dynamic environments: Dijkstra's common use works with a static graph (where the vertices and edges, established at the beginning, are always available). If the application requires to modify the connections in the graph, it is necessary stop processing, modify the graph, load the graph and execute the process again. All these steps when applied on large graphs, involves waste of time, specially by considering emergency situations. For this, the SSER approach includes the initially defined "Accessibility Status" property in each vertex, in order to label the vertices as available or unavailable. It should be noted that the update of the accessibility status is performed in real time without having to stop processing. The information is sent by sensors housed in different building areas, the kind of sensors used depends on the situation to monitor (fire, smoke, toxic gases, among others). Based on the current status of the vertex, the SSER approach will only consider the available vertices to obtain the evacuation routes in the building. In this way, people are not addressed by escape routes involving dangerous areas.

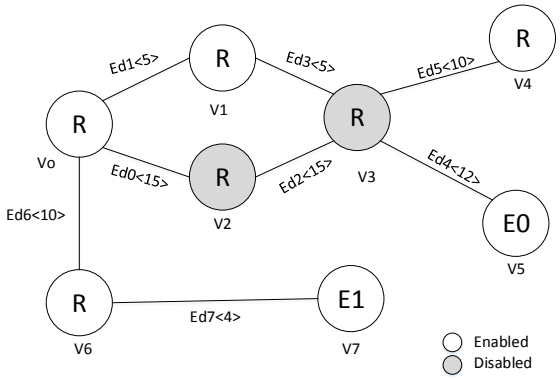

Fig. 5. Graph illustrating vertices with different access status

In this context, the proposed approach SSER is able to find the shortest-safe evacuation route, considering the access availability of vertices.

4 Experimental results

It was compared the SSER algorithm's runtime between other 4 Dijkstra-based algorithms (Fibonacci, BGL, Lemon and OrTool)[15] with focus on graphs analysis using different data structures. One of the main considerations for testing was to use the same programming language in the implementation of the mentioned methods to compare, in order to avoid discrepancies due to compiler or code interpreter. The SSER approach was developed in C++. For testing purposes were considered 3 graphs randomly-created (see Table 3), each one with 10,000 vertices (n), density (d) and different arcs (m). The equation used to

determine the graphs is: $\frac{2m}{n(n-1)}$ [15]. Following this, the algorithm SSER was applied to each graph in Table 3, where were generated randomly 50 evacuation doors (vertices as destination). The tests were performed on a computer Lenovo T440p, Core-i7 with 16GB of RAM. The algorithm analyzed evacuation routes to all vertices in the graph. This was not done with the other 4 methods (Fibonacci, BGL, Lemon and OrTool) because these are focused on determining the shortest path between two vertices.

Table 3. Random Graphs considered from [15]

Graph	**n**	**m**	**d**
Rand1	10000	100000	0.001
Rand2	10000	1000000	0.01
Rand3	10000	10000000	0.1

Subsequently, we proceeded to calculate the execution time of the SSER algorithm. The results are detailed in Table 4, where can be observed the execution time of the SSER algorithm compared with the other four methods, which is positioned in third place among the analyzed algorithms.

Table 4. Comparing runtimes

Graph	**Fibonacci**	**BGL**	**Lemon**	**OrTools**	**SSER**
Rand1	0.052	0.0059	0.0074	1.2722	0.0177
Rand2	0.0134	0.0535	0.0706	1.6128	0.0459
Rand3	0.0705	0.5276	0.7247	4.2535	0.3704

One of the causes of its performance is due to its goals of seeking the safe evacuation of all areas (represented by vertices in the graph), so it focused on finding evacuation routes considering all vertices from each of the 50 evacuation doors randomly selected for the test.

Although SSER did not get the best runtime, the results are enhanced by the selection of safe routes considering all escape doors. With this it could be justified how the SSER algorithm has an effective role in situations involving finding the shortest safe routes, where is required to consider all selected vertices in the graph to specific ones.

After analyzing the efficiency, was addressed the role of the algorithm to be suitable for emergency situations. The study considered a graph with 8 vertices, 2 evacuation doors E_0 and E_1, 8 aisles and 6 areas that need to be evacuated (see Fig. 6). With this graph an emergency was simulated by making unavailable the vertex V_3. The SSER algorithm considered the unavailability status of V_3 and discarded any evacuation route to the escape door E_0 for safety reasons. This

was achieved by checking the availability status of vertices for each iteration and considering only those accessible and safe. In addition, SSER marked V_4 as an blocked area since the only aisle $Ed_5 < 10 >$ suggests pass through a high risk area.

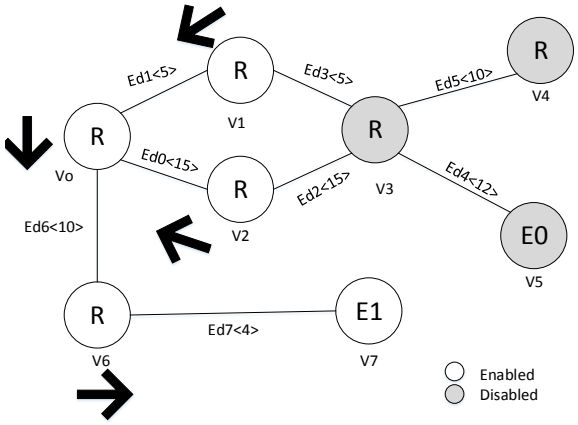

Fig. 6. Illustration of an Emergency in $v3$

Although the shortest routes for V_1 and V_2 have the same destiny E_0, the algorithm have redirected the escape to the door E_1 to fulfill its goal in the selection of safe routes.

On the other hand, should be mentioned that each vertex stores information about segments around vertices that make up different evacuation routes. In the particular emergency illustrated in Fig. 6, the vertices stored only one evacuation route to E_1. After that V_3 is enabled, the evacuation routes to E_1 are recalculated toward the two evacuation doors. Each vertex considered between the two options the shortest one.

5 Conclusions

This paper tackles the challenging problem of computing evacuation routes in buildings with multiple exits, seeking the safe option before the shortest one. In order to fulfill this goal, the SSER approach considers variations on the graph by storing meta-data into vertices, in order to be able to select another alternative when some connection into the graph is broken, this can prevent evacuation systems to suggest routes involving blocked or dangerous areas. This approach makes the algorithm suitable to be applied on dynamic environments where the availability status of internal areas can be modified by different kind of sensors.

References

1. Cho, J., Lee, G., Won, J., Ryu, E.: Application of dijkstra's algorithm in the smart exit sign. In: The 31st International Symposium on Automation and Robotics in Construction and Mining (ISARC 2014). (2014)
2. Santos Navarrete, M.E., et al.: Estudio y simulación de algoritmos para la evacuación de personas en situaciones de emergencia sobre una estructura similar al rectorado de la espol. RTE (2015)
3. Jang, J.S., Kong, I.C., Rie, D.H.: A study for optimal evacuation simulation by artificial intelligence evacuation guidance application. Journal of the Korean Society of Safety **28**(3) (2013) 118–122
4. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische mathematik **1**(1) (1959) 269–271
5. Floyd, R.W.: Algorithm 97: shortest path. Communications of the ACM **5**(6) (1962) 345
6. Bellman, R.: On a routing problem. Technical report, DTIC Document (1956)
7. Artmeier, A., Haselmayr, J., Leucker, M., Sachenbacher, M.: The shortest path problem revisited: Optimal routing for electric vehicles. In: KI 2010: Advances in Artificial Intelligence. Springer (2010) 309–316
8. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speedup dijkstra's algorithm. J. Exp. Algorithmics **11** (February 2007)
9. Karypis, G., Kumar, V.: Metis: Family of multilevel partitioning algorithms. wwwusers. cs. umn. edu/˜ karypis/metis/main. shtml (1995)
10. Rosenthal, E.C.: Shortest path games. European Journal of Operational Research **224**(1) (2013) 132–140
11. Randell, B.: Edsger dijkstra. In: Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall. The Ninth IEEE International Workshop on. (Oct 2003) 3–3
12. Kim, D.O., Mun, H.W., Lee, K.Y., Kim, D.W., Gil, H.J., Kim, H.K., Chung, Y.S.: The development of the escape light control system. Journal of the Korean Institute of Illuminating and Electrical Installation Engineers **23**(6) (2009) 52–58
13. Xu, Y., Wang, Z., Zheng, Q., Han, Z.: The application of dijkstra's algorithm in the intelligent fire evacuation system. In: Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2012 4th International Conference on. Volume 1., IEEE (2012) 3–6
14. Johnsonbaugh, R.: Discrete Mathematics. 7 edn. Prentice Hall (12 2009)
15. Gualandi, I.: Dijkstra, Dantzig, and shortest paths. http://stegua.github.io/blog/2012/09/19/dijkstra/ (2012) [Online; accessed Jun. 18, 2015.].