DEVELOPMENT OF AN ALGORITHM FOR THE OPTIMIZATION OF PEDESTRIAN ROUTES BASED ON STREET HARASSMENT OF THE LGBTQ COMMUNITY

Valentina Movil Universidad
Universidad Eafit
Colombia
vmovils@eafit.edu.co

Felipe Gil Macia
Universidad Eafit
Colombia
fgilm@eafit.edu.co

Andrea Serna
Universidad Eafit
Colombia
asernac1@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

ABSTRACT

The acronym LGBTQ has been used to refer to people that identify as lesbian, gay, bisexual, or transgender or those who have doubts about their sexuality and/or gender identity (queers) [1]. On the other hand, during 2017, a large part of the violent deaths of LGBTQ individuals in Colombia occurred at Medellín. Then, many of the victims were engaged in low-paid jobs and high exposure to violence, that is, recycling, informal trade, and prostitution [2]. Therefore, the LGBTQ population is mostly exposed to violence on the street. Next, in the Aburrá Valley, 30% of trips are made on foot or by bike [3]. For this, it is necessary to provide security to the citizens, making special emphasis on population groups that, when moving, are more likely to be attacked, for example, LGBTQ individuals and/or groups. In order to provide security to the LGBTQ community, in this project is implemented the Dijkstra´s Algorithm for the find of the shortest path without exceeding a specified risk of harassment; so, the people could walk by roads that in terms of statistics are secure. The average time for the program to work is about 2 minutes. In conclusion the technology and the algorithms are very useful to create solutions for the daily lives of the people; and the implementation of some solution by algorithms is well done if the efficiency in time and computational memory are analyzed and have sense for the requirements of the solution.

Keywords

Constrained shortest path, street sexual harassment, secure-path identification, crime prevention.

1. INTRODUCTION

The denomination LGBTQ is an inclusive name that refers to lesbians, gays, bisexuals, transgender, or queers [1]. On the other hand, LGBTQ people have been overwhelmingly subjected to domestic and sexual violence [4, 5]. For example, during 2017 a large part of the violent deaths of LGBTQ people occurred in Medellín. About their trades, many of the victims were engaged in low-paid jobs and high exposure to violence, that is, recycling, informal trade, and prostitution [2]. Additionally, the origin-destination survey carried out in the Aburrá Valley (AMVA-in Spanish acronym) that same year showed that around 30% of the trips in this place are made on foot or by bicycle. Then, with the interest of being a sustainable territory, government entities have promoted the use of these means of transport among the inhabitants of the AMVA. However, vulnerable groups such as women and the LGBTQ community use them the least [3]. For example, it is estimated that LGBTQ people and/or groups are 3 or 4 times more likely to be attacked [5]. For this reason, it is necessary to propose and carry out projects, in this case, in science, technology, and innovation that improve their safety guarantees on the street. Specifically, this paper seeks to point to what has been said through the development of an optimization algorithm for pedestrian routes based on street harassment of people from the LGBTQ community.

1.1. Problem

In the AMVA, the local government promotes walking and cycling as a tool to reduce air pollution. Likewise, it places the pedestrian at the top of priorities in terms of mobility [3]. Currently, however, the main reason for vulnerable communities (women, LGBTQ people, etc.) not to commute sustainably (i.e., walking or cycling) is exposure to street harassment [6]. That is why, in this work, the development of an algorithm for optimizing pedestrian routes through a data structure is presented, which focuses on calculating the shortest path without exceeding a weighted average risk of street harassment for people from the LGBTQ community. In fact, this solution provides a reference framework in ST+I in the reduction of insecurity in terms of sustainable mobility with a gender approach.

1.2 Solution

In the second deliverable there were two problems to solve; the first one was to find the shortest path from a point A to a point B and the second one was to find the path from a point A to a point B that have the less weighted-average risk of harassment. The dataframe given was processed with the help of Pandas, and then with the dataset was created a dictionary that represent a graph; and with this data structure the next step was to implement the algorithm that gives the solution to those problems. The Dijkstra algorithm has been implemented to solve the two problems; the first problem was solved for the variable distance and the second one was solved for the value r (the risk of harassment). The Dijkstra algorithm was selected after studying other algorithms like A*, BFS or Floyd-Warshall's algorithm; and the reason is because after investigating, it seems that A* and Dijkstra were potential algorithms for solve this problem; although A* is very powerful because uses more information from the dataset (the heuristic estimation) that helps to get the solution in less time than in Dijkstra but because of the amount of data is very expensive in terms of memory; because of that the algorithm selected was Dijkstra.

1.3 Article structure

In what follows, in Section 2, we present related work to the problem. Later, in Section 3, we present the data sets and methods used in this research. In Section 4, we present the algorithm design. After, in Section 5, we present the results. Finally, in Section 6, we discuss the results, and we propose some future work directions.

2. RELATED WORK

In what follows, we explain four related works to path finding to prevent street sexual harassment and crime in general.

2.1. RELATED PROJECTS

2.1.1. Walking Secure: Safe Routing Planning Algorithm and Pedestrian's Crossing Intention Detector Based on Fuzzy Logic App

This job pretends to improve road safety through artificial intelligence (AI). AI is now crucial to achieving more secure smart cities. For this reason, another objective of this reference is to develop a mobile app based on the integration of the smartphone sensors, and a fuzzy logic strategy to determine the pedestrian's crossing intention around crosswalks is presented. Also, the app developed also allows the calculation, tracing, and guidance of safe routes thanks to an optimization algorithm that includes pedestrian areas on the paths generated over the whole city through a cloud database (i.e., zebra crossings, pedestrian streets, and walkways). Finally, a total of 30 routes were calculated by the proposed algorithm and compared with Google Maps considering the values of time, distance, and safety along the routes. As a result, the routes generated by the proposed algorithm were safer than the routes obtained with Google Maps, achieving an increase in the use of safe pedestrian areas. [8].

2.1.2. A computational model of pedestrian road safety: the long way round is the safe way home

This work proposes a novel linear model of pedestrian safety in urban areas with respect to road traffic crashes that considers a single independent variable of pedestrian path safety. This variable is estimated for a given urban area by sampling pedestrian paths from the population of such paths in that area and in turn estimating the mean safety of these paths. We argue that this independent variable directly models the factors contributing to pedestrian safety. This contrasts previous approaches, which, by considering multiple independent variables describing the environment, traffic, and pedestrians themselves, indirectly model these factors. In sum, using data about 15 UK cities, it demonstrated that the proposed model accurately estimates numbers of pedestrian casualties [9].

2.1.3. A route navigation algorithm for pedestrian simulation based on grid potential field

This job explains that pedestrian simulation modeling has become an important means to study the dynamic characters of dense populations. In the continuous pedestrian simulation model for complex simulation scenarios with obstacles, the pedestrian path planning algorithm is an indispensable component, which is used for the calculation of pedestrian macro path and microscopic movement desired direction. However, there is less efficiency and poor robustness in the existing pedestrian path planning algorithm. To address this issue, we propose a new pedestrian path planning algorithm to solve these problems in this article. In our algorithm, we have two steps to determine pedestrian movement path, that is, the discrete potential fields are first generated by the flood fill algorithm and then the pedestrian desired speeds are determined along the negative gradient direction in the discrete potential field. Combined with the social force model, the proposed algorithm is applied in a corridor, a simple scene, and a complex scene, respectively, to verify its effectiveness and efficiency. In conclusion, the results demonstrate that the proposed pedestrian path planning algorithm in this article can greatly improve the computational efficiency of the continuous pedestrian simulation model, strengthen the robustness of application in complex scenes [10].

2.1.4. Routes' Safety Evaluation Application Development

For this job, pedestrians and cyclists are the most vulnerable road users. One of the ways to increase their safety is the safest route selection. Nowadays, there are a lot of navigation systems and route planners. However, these routes are not assessed from the viewpoint of their safety. The article describes one of the possible system alternatives to assess the safety of routes for different types of road users. The developed system is based on the multifactorial analysis of information. Correction factors allow considering special

features of each type of road user. Finally, the developed algorithm is implemented as an application [11].

3. MATERIALS AND METHODS

In this section, we explain how data was collected and processed and, after, different constrained shortest-path algorithm alternatives to tackle street sexual-harassment.

3.1 Data Collection and Processing

The map of Medellín was obtained from Open Street Maps (OSM)[1] and downloaded using Python OSMnx API[2]. The (i) length of each segment, in meters; (2) indication wheter the segment is one way or not, and (3) well-known binary representation of geometries was obtained from metadata provided by OSM.

For this project, we calculated the linear combination that captures the maximum variance between (i) the fraction of households that feel insecure and (ii) the fraction of households with income below one minimum wage. These data were obtained from the quality-of-life survey, Medellín, 2017. The linear combination was normalized, using the maximum and minimum, to obtain values between 0 to 1. The linear combination was obtained using principal components analysis. The risk of harassment is defined as one minus the normalized linear combination. Figure 1 presents the risk of harrament calculated. Map is available at Github[3].

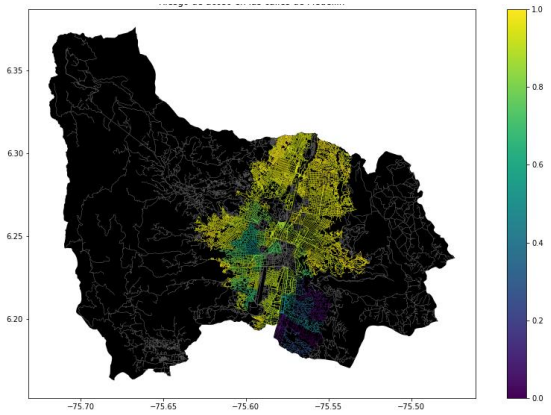


Figure 1. Risk of sexual harassment calculated as a lineal combination of the fraction of households that feel insecure and the fraction of households with income below one minimum wage, obtained from Life Quality Survey of Medellín, in 2017.

2.2. RELATED ALGORITHMS

2.2.1. Dijkstra algorithm

The objective of this algorithm is to travel from vertex A to vertex B by the fastest possible path. But ¿How does it work? This algorithm only has two processes; the first one is to update the estimates and the second one is to choose the next vertex.

At first, it has the weighted graph (i.e., vertex connected by edges that have different weights, those could be the distance or the time that takes going from a vertex to another one). So, the first step is to estimate the time or distance (which is dependent on the weights at the edges) that takes going from the initial vertex to any vertex. It's remarkable that in this step, the only time it knows is the time that takes going from the initial vertex to the same vertex, this is 0. Next, for the other vertexes, it does not know how long it takes to get there. Therefore, it estimates the infinite time in all the unexplored vertexes.

Following, it begins to compare which one of those vertexes has the minimum distance or time to get there. For example, where is an update estimation, must compare all the new vertexes that are known, and must choose the one that has the minimum time or distance to get there (choose the next vector). After, it repeats that process until arrived at the goal vertex by the shortest possible path. It should be notated, that must add a process that saves the edges that were taken to have the shortest path in our memory (See Figure 1.).

Finally, if it is in a vertex and at the time it is updating estimates and if one vertex has an estimate minor then the new estimate that it could have goes by a different edge or path. Then, it must not change the valor of the estimate in that vector. This is, it should only update estimations if the new value that we found is minor then the actual estimation. Too, this algorithm does not work when the weights of the edges are negative.

The complexity in time of this algorithm is $O(V^2)$.

[1] https://www.openstreetmap.org/

[2] https://osmnx.readthedocs.io/

[3]https://github.com/mauriciotoro/ST0245Eafit/tree/master/proyecto/Datasets/

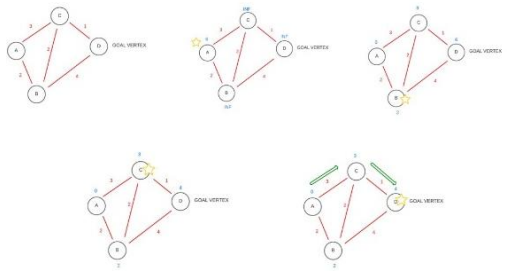


Figure 1: An example of finding the shortest path by Dijkstra algorithm [7].

2.2.2. A-star (A*)

Like before, the objective of this algorithm is to travel from vertex A to vertex B by the fastest possible path. For this reason, the A* algorithm uses the same method as the Dijkstra algorithm with the difference that this algorithm it uses an additional parameter to compare the shortest path, and that parameter is called the heuristic.

Consequently, while Dijkstra only chose the shortest edge between two vertexes, even if going that way means being further from the goal vertex; the heuristic measures the distance between the vertex that we are estimating and the final vertex. Therefore, the algorithm described so far gives us only the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each node on the path keeps track of its predecessor. After this algorithm is run, the ending node will point to its predecessor, and so on, until some node's predecessor is the start node.

In addition, is possible to say that the A* algorithm is more efficient. For example, the following figure shows that if we are standing in A and that vertex relates to B and C, the weight of the edge A to B is 4 and the weight of the edge A to C is 5, and we want to go to D that has a distance of 7 with B and a distance of 5 with C. Then, if we follow the Dijkstra algorithm, we are going by the vertex B but, the best option was vertex C, the A* algorithm knew it because of the heuristic that shows not only how long are the connected vertex but also how long are the connected vertex with the goal vertex.

For all above, as the same problem shown in Figure 1 whose was solved by Dijkstra now it is solved by A*, here the lector can see that the process is faster than in Dijkstra.

The time complexity of this algorithm is O(|V|+|E|), where V is the number of vertices and E the number of edges.

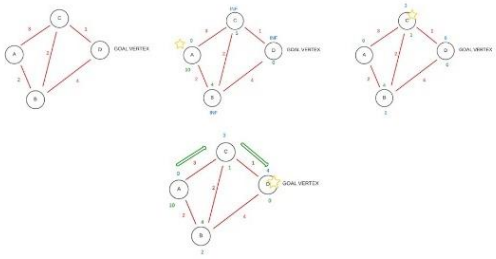


Illustration 2: The example solved in illustration 1, but now solved by A* algorithm [7].

2.2.3 Floyd-Warshall's algorithm

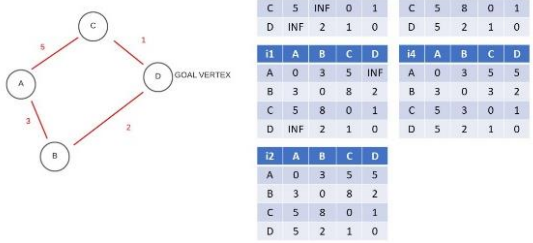
In the same way, the objective of this algorithm is to travel from A to a B by the fastest possible path.

In consequence, this algorithm has two processes; the first one is to make a table where we calculate all the distances between all the pairs of vertexes, and the second one is to select the minimum value between two paths that takes to the same vertex, passing by different edges.

Then, to start, the first table (i0) calculates the distances between pairs of vertexes witch pass for a particular pair of vertexes. Hence, if those vertexes do not have any common path, must be assumed infinite distance. After it starts to iterate, being the first iteration the distance between a pair of vertexes passing by the vertex 1, so it compares the new values of distances with the table of the last iteration and if it found shorter values than those it had before; it proceeds to add those values to the new table.

Finally, it iterates until we get to the table that makes the paths cross the vertex n (being vertex n the last vertex of the table); and in that table, we should have the shortest path for all the pairs of vertexes (See Figure 3.).

The time complexity of this algorithm is $O(V^3)$, where V is the number of vertices.



i0	A	B	C	D
A	0	3	5	INF
B	3	0	INF	2
C	5	INF	0	1
D	INF	2	1	0

i1	A	B	C	D
A	0	3	5	INF
B	3	0	8	2
C	5	8	0	1
D	INF	2	1	0

i2	A	B	C	D
A	0	3	5	5
B	3	0	8	2
C	5	8	0	1
D	5	2	1	0

i3	A	B	C	D
A	0	3	5	5
B	3	0	8	2
C	5	8	0	1
D	5	2	1	0

i4	A	B	C	D
A	0	3	5	5
B	3	0	3	2
C	5	3	0	1
D	5	2	1	0

Figure 3: An example of finding the shortest path between vertex by Floyd-Warshall's algorithm [7].

2.2.4 BFS algorithm

The objective of this algorithm is to find paths from one vertex to another one. Hence, this algorithm is a recursive one witch asks the vertex where it is at the beginning and if there is an edge that connects this vertex with the goal vertex and if the answer is false, it calls the vertexes that relate to the initial vertex and makes the same question to each connected vertex until the answer to the question is true or until it asks to all the vertexes of the graph. It is remarkable to say that this algorithm works wide and not deep because he asks to the first vertex and then to all his related vertex before he asks for deeper vertexes. This algorithm works for unweighted graphs. In sum, the following figure shows a practical view of what was said before.

The time complexity of this algorithm is O(|V|+|E|), where V is the number of vertices and E the number of edges.

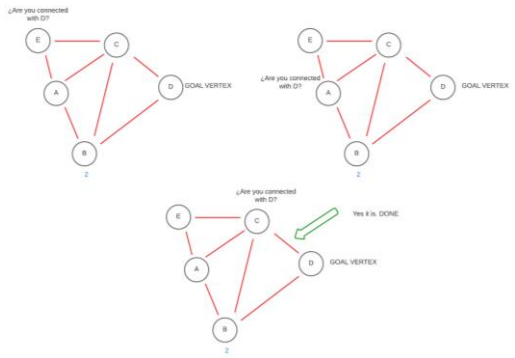


Figure 4: An example of finding a specific node by BFS algorithm [7].

4. ALGORITHM DESIGN AND IMPLEMENTATION

In what follows, we explain the data structures and the algorithms used in this work. The implementations of the data structures and algorithms are available at Github[4].

4.1 Data Structures

The data structure that is used to solve this problem is an adjacency list using a dictionary, with this list is possible to create the graph that is necessary for the searching of the shortest path or the path with less risk of harassment.

The dictionary makes the adjacency list creating a key for all the nodes of the graph; and the value assigned to each key (each node) is a new dictionary that has the nodes (key) that are connected to the key node and the value assigned is the weight of the edge (it could be the length of the street or the risk of harassment).

In the following figure it is illustrated the graph that is represented by an adjacency list using dictionary:

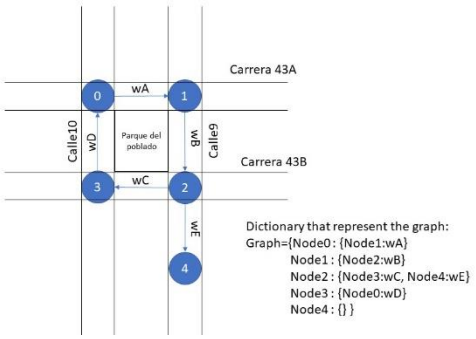


Figure 2: An example of a street map of Medellin is presented and the adjacency list created using dictionary that represent the graph that is in the map.

4.2 Algorithms

In this work, we propose algorithms for the constrained shortest-path problem. The first algorithm calculates the shortest path without exceeding a weighted-average risk of harassment *r*. The second algorithm calculates the path with the lowest weighted-average risk of harassment without exceeding a distance *d*.

4.2.1 First algorithm

The design of the Dijkstra algorithm has the following steps:

1. Set the values of the origin node and the uncertain nodes: Because of the uncertainty in the cost of going to any node from the starting node; all the cost of the nodes will be considered infinite; the only node that the cost is known is the starting node and the cost to visit this node is 0.
2. Calculate the cost of the neighbor nodes: Calculate the cost of going by each neighbor node; and changing the value of infinite in the cost of the neighbor nodes to the value that is the sum of the cost of the origin node with the weight of the edge between the nodes.
3. Find the shortest path between nodes: Because of having the cost of the neighbor nodes, it is possible to discover the cost of the new neighbor nodes, comparing all the possible path to get to a node and always choosing the path of the minimum cost.

[4]https://github.com/FelipeGilM/ST0245002/tree/master/codigo

4. Show the path of the shortest cost: After knowing the shortest distances between nodes it is implemented a function that give the path of the shortest distance going from the origin node of the third point to any node that is ask in this function.

The steps 1,2 and 3 are in the 'calculate_distances_from' function; and the step 4 is in the 'get_path_to' function.

Algorithm is exemplified in Figure 3:

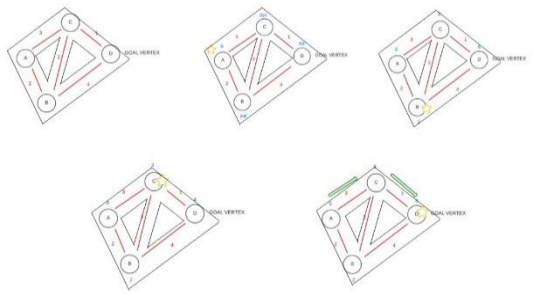


Figure 3: Solving the problem of the shortest path with the implemented algorithm (Dijkstra), the last step of the figure is the result of the 'get_path_to' function.

4.2.2 Second algorithm

This second algorithm is a variation of the first one; and the variation is to achieve not only the shortest path or the less risk of harassment path but the shortest path without exceeding an established risk of harassment or the less risk of harassment path without exceeding an established distance.

4.4 Complexity analysis of the algorithms

As stated above this is the worst-case complexity for Dijkstra's algorithm with O(V^2) when implementing using an unsorted array and no priority queue. This is because for each vertex (V), we need to relax the connected edges to find the minimum cost edge that connects a vertex to V. We need to do V number of calculations and each operation takes O(V) times, therefore leaving us with the complexity of O(V^2).

Algorithm	Time Complexity
Dijkstra algorithm, optimizing only one variable	$O(V^2)$

Table 1: Time Complexity of the used algorithms, where V is the number of vertexes of the graph and E is the number of edges of the graph.

Data Structure	Memory Complexity
Adjacency List	$O(V^2)$

Table 2: Memory Complexity of adjacency list, where V is the number of vertexes of the graph and E is the number of edges of the graph.

4.5 Design criteria of the algorithm

First, it's important to say that the Dijkstra algorithm is not the best option for solving the shortest path problem; there are other algorithms like A*, that is heuristic, algorithms like that one is more efficient than the implemented algorithm (Dijkstra).

Although is recognized that A* is more efficient than Dijkstra for this type of problem; in this case, with the dataset and the information given in order to solve the problem; its inefficient to use A*; the complexity of A* in this case was far larger than the Dijkstra one; and other reason of the use of Dijkstra was due to the complete documentation of this algorithm in the network that makes possible the implementation and the solution of the problem.

5. RESULTS

In this section, we present some quantitative results on the shortest path and the path with lowest risk.

5.1.1 Shortest-Path Results

In what follows, we present the results obtained for the shortest path without exceeding a weighted-average risk of harassment *r* in Table 3.

Origin	**Destination**	**Shortest Distance**	**Without Exceeding *r***
Universidad EAFIT	Universidad de Medellín	490.864	0.84
Universidad de Antioquia	Universidad Nacional	815.437	0.83
Universidad Nacional	Universidad Luis Amigó	1231.125	0.85

Table 3. Shortest distances without exceeding a weighted-average risk of harassment *r*.

5.1.2 Lowest Harassment-Risk Results

In what follows, we present the results obtained for the path with lowest weighted-average harassment risk without exceeding a distance *d* in Table 4.

Origin	Destination	Lowest Harassment	Without Exceeding *d*
Universidad EAFIT	Universidad de Medellín	0.73	5,000
Universidad de Antioquia	Universidad Nacional	0.87	7,000
Universidad Nacional	Universidad Luis Amigó	0.86	6,500

Table 3. Lowest weighted-average harassment risk without exceeding a distance *d* (in meters).

5.2 Algorithm Execution-Time

In Table 4, we explain the relation of the average execution times for the queries presented in Table 3.

Compute execution time for the queries presented in Table 3. Report average execution times.

	Average execution times (s)
Universidad EAFIT to Universidad de Medellín	60 s
Universidad de Antioquia to Universidad Nacional	120 s
Universidad Nacional to Universidad Luis Amigó	210 s

Table 4: Execution times of the Djikstra for the queries presented in Table 3.

5. CONCLUSIONS

To conclude is important to said that the technology and the algorithms are very useful for the people´s daily lives; this work gives the opportunity to the people to be more secure and to have the better route not only in terms of length but also in terms of security.

Also is very important to consider the complexity in memory and also in time and after find the way to optimized those factors; see if the application of the program have sense with those values; for example if the user want to know what path to choose; the time of calculation should not overpass certain time because the user don't want to stay in a place waiting to know with path take and the program should be functional for a big variety of phones, so it needs to not take a lot of memory.

Because of the achieve results, is correct to said that this work can be useful to the LGBTQ community to walk more secure in Medellin, but it is important to validate the results because in the table 2 in the finding of the shortest path between Universidad Eafit and Universidad de Medellin it gives a distance lower than 1km; and that do not have sense because those universities are further than 1km.

6.1 Future work

This project has a lot of opportunities to keep working on it in Mathematical engineering; the 3 topics that are considered important to keep exploring are: to find the way to use A* algorithm that is faster than Dijkstra, and achieve to use it with less use of memory, the find of the path that have the best relation between 2 variables; and to implement this project in a mobile app.

ACKNOWLEDGEMENTS

The authors are grateful to Prof. Juan Carlos Duque, from Universidad EAFIT, for providing data from Medellín Life Quality Survey, from 2017, processed into a Shapefile.

REFERENCES

[1] Lambda Legal and Child Welfare League of America, "Conceptos básicos sobre ser LGBT", Lambda Legal, 2013.

[2] S. Kyu Choi, S. Divsalar, J. Flórez-Donado, K. Kittle, A. Ilan H. Meyer and P. Torres-Salazar, "STRESS, HEALTH, AND WELL-BEING OF LGBT PEOPLE IN COLOMBIA Results from a National Survey", 2019.

[3] Área Metropolitana del Valle de Aburrá, "AVANZA EL PROGRAMA DE CICLOCAMINABILIDAD El Aburrá le camina a la bici", El Metropolitano del Valle de Aburrá, Medellín, 2019.

[4]"Violence Against LGBTQ People | CARE Office", Care.ucmerced.edu, 2022. [Online]. Available: https://care.ucmerced.edu/advocacy/violence-against-lgbtq-people. [Accessed: 23- Feb- 2022].

[5]"Sexual Assault and the LGBTQ Community", Human Rights Campaign, 2022. [Online]. Available: https://www.hrc.org/resources/sexual-assault-and-the-lgbt-community. [Accessed: 23- Feb- 2022].

[6]S. Aguirre, "¿MEDELLÍN ES UNA CIUDAD HOSTIL PARA LAS CICLISTAS?", Colectivo SiCLas, Medellín, 2021.

[7]D. Delling, P. Sanders, D. Schultes and D. Wagner, "Engineering Route Planning Algorithms", Algorithmics, vol. 5515, pp. 117–139, 2009. Available: https://i11www.iti.kit.edu/extra/publications/dssw-erpa-09.pdf. [Accessed 23 February 2022].

[8]J. Lozano Domínguez and T. Mateo Sanguino, "Walking Secure: Safe Routing Planning Algorithm and Pedestrian's Crossing Intention Detector Based on Fuzzy Logic App", Sensors, vol. 21, no. 2, p. 529, 2021. Available: 10.3390/s21020529.

[9]C. Hannah, I. Spasić and P. Corcoran, "A computational model of pedestrian road safety: The long way round is the safe way home", Accident Analysis & Prevention, vol. 121, pp. 347-357, 2018. Available: 10.1016/j.aap.2018.06.004.

[10]M. Li, Y. Wei and Y. Xu, "A route navigation algorithm for pedestrian simulation based on grid potential field", Advances in Mechanical Engineering, vol. 11, no. 12, p. 168781401989783, 2019. Available: 10.1177/1687814019897831.

[11]K. Shubenkova, V. Mavrin, G. Sadygova and P. Buyvol, "Routes' Safety Evaluation Application Development", MATEC Web of Conferences, vol. 334, p. 02010, 2021. Available: 10.1051/matecconf/202133402010.