SAFEST PATH FINDING ALGORITHMS ON STREET HARASSMENT PREVENTION

Lina Sofía Ballesteros
Universidad Eafit
Colombia
lsballestm@eafit.edu.co

Camilo Córdoba
Universidad Eafit
Colombia
ccordobab@eafit.edu.co

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

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

ABSTRACT

Street harassment is one of the main reasons why not everyone feels comfortable going out on the street, it is a form of intimidation by a stranger that occurs in public spaces. The way it has become normalized is frightening once you realize that it is expected when going out, especially by women. Causing fear of using public transportation or being alone outside, in addition to anxiety and stress affecting their lives.

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

1. INTRODUCTION

The present work aims to generate an alternative solution to the current problem of street harassment, a form of sexual harassment that generally occurs in a public space and in which women are mainly affected. According to Billi [1] (2015), "street sexual harassment corresponds to any practice with an explicit or implicit sexual connotation, which comes from a stranger [...] and has the potential to cause discomfort in the harassed person". In Medellín, it is a situation that puts the lives of the victims at risk and that threatens their dignity as people. In addition, it is an issue that also mentally affects the victims and makes them go to the streets in fear or feel vulnerable [2] to harassers. From the foregoing lies the importance of understanding this problem as a systematic issue that must be studied and addressed rigorously, seeking to reduce the cases of street sexual harassment that occur so frequently.

According to Daniela Maturana [3], former councilor of Medellín, currently there are not enough tools or ways of attention to report street harassment, that is why the following report seeks to generate a resource that allows promoting the safety and protection of affected people through the implementation of a safe path algorithm.

1.1. Problem

The main problem around the development of this work is to find an efficient way to develop an algorithm that shows the user the safest route and at the same time the most comfortable to arrive, that is, the route that offers a path with the least amount of harassment and at the same time, in terms of distance, it is not so long for the person who wants to get from one place to another. Therefore, the real challenge is to offer a balance between both needs in three different ways and, likewise, to determine the calculation of the risk of harassment in a coherent manner according to the map of the city of Medellín.

1.2 Solution

To prevent street harassment, the idea is to design a program capable of showing the best route from one point to another in terms of avoiding street harassment and considering the distance between the two points, this two term multiplied are the weights in our graph since all the weights in our graph are positive, Dijkstra was considered the best fit for our project. It is an algorithm specifically designed to find the shortest path between two vertices in a weighted graph. Not only it does what we need but also it is well known as a fairly low time complexity algorithm.

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

Below, we explain four articles related to finding ways to prevent street sexual harassment and crime in general.

2.1 “Siempre Seguras”: diagnosis and prevention of street harassment

"Siempre Seguras" is an initiative of diagnosis and prevention of street harassment [4] that seeks to generate a mapping of sexual street harassment in Mexico, through an app whose functionality is based on identifying the areas where harassment is most concentrated. This map, created in the city of Querétaro, collects statistical data on street harassment and displays them on a map of red-hot spots [5]. The data are reported by users from the Twitter platform through the account @siempre_seguras and are processed through algorithms supported by artificial intelligence that recognize natural language to identify locations, perceptions, schedules, etc. One of the findings that could be obtained

thanks to this project was the fact that women suffer more harassment in public transport [5].

2.2 Safecity: an app that seeks to create safer spaces

Safecity is a platform that aims to empower individuals, communities, police and municipal/departmental governments to create safer public and private spaces for all. Through an algorithm that collects anonymous reports of sexual violence [8], Safecity analyzes these testimonies identifying patterns and key insights, generating a map of hot spots with predictions about crime statistics near the user's location [7]. Among the most important findings, it was recognized that users, such as police officers, felt comfortable using this application and being able to report crime cases. This initiative has won a variety of awards since its launch, such as the "Winner of the World Justice Forum's Equal Rights and Non-Discrimination Award" in 2022 [6].

2.3 Harrasmap: Using collaborative data to map sexual harassment in Egypt.

Harrasmap is a digital and online platform that proposes the use of spatial information technologies to map places where incidents of street harassment have occurred, for this purpose the platform is based on "crowdmapping", a form of mapping where the incoming data comes from the crowds. Victims can anonymously report what happened and where it happened, subsequently through an algorithm these reports are categorized according to the type of harassment they deal with, to then be displayed on an online map. this platform has left a positive impact in Egypt, country where it currently operates thanks to the attention it attracted is causing a change in the attitude of society towards this problem, such has been this impact, that before its existence street harassment was not considered illegal, currently in Egypt is already considered illegal and involves consequences.[18]

2.3 “Safe and the City": a social enterprise that seeks to design smarter and safer cities.

"Safe & the City" is a mobile app that shows the worst areas in London for street harassment so that men and women can plan their routes around it. The algorithm used by the platform is designed to create personalized walking routes avoiding areas and locations with high numbers of reported incidents of street harassment are also based on crime data, the amount of lighting on a street, the number of stores open and reports of street sexual harassment made by the public. People can report anonymously if they have experienced harassment and then the map is updated for others to see. In addition to having the option to alert the authorities if they find themselves in a dangerous situation. The platform has helped to raise awareness of this problem and give users a tool that provides them with security and increased confidence when they go out on the street.[16] The platform has helped to raise awareness of this problem and to give users a tool that provides them with security and increased confidence when they go out on the streets.

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 whether the segment is one way or not, and (3) well-known binary representation of geometries were 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. This data was 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 harassment calculated. Map is available on Github[3].

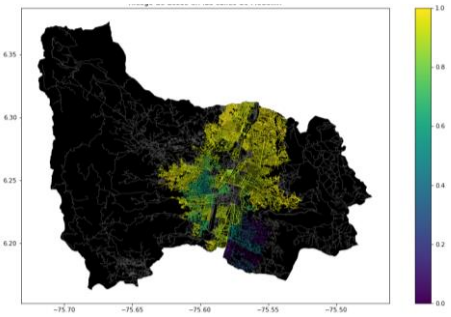


Figure 1. Risk of sexual harassment calculated as a linear combination of the fraction of households that feel insecure and the fraction of households with income below one minimum wage, obtained from Medellin’s 2017 Life Quality Survey.

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

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

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

3.2 Constrained Shortest-Path Alternatives

In what follows, we present different algorithms used for constrained shortest paths.

3.2.1 Dijkstra's algorithm

Dijkstra's algorithm was conceived in 1956 and first published in 1959 by Edsger Dijkstra, a computer scientist from the Netherlands. The main component of this algorithm is that its goal is to find the shortest distances from a source vertex to any other node of a specific network [9]. This network is known as a graph, and refers to the unit made up of objects called nodes or vertices that are linked by edges or arcs and represent binary relationships between the elements of a certain set [10]. The edges of the graph must express positive values since this algorithm does not work for negative values.

The operation of this algorithm is based on iterations, so as the size of the network increases, its development could be difficult compared to other optimization methods within mathematical computational sciences. To start working with the algorithm, it is necessary to indicate the graph to be implemented with its source or initial node joined by subnodes whose links contain "weights" or values [11] that allow calculating the shortest path, as shown in the following graphic:

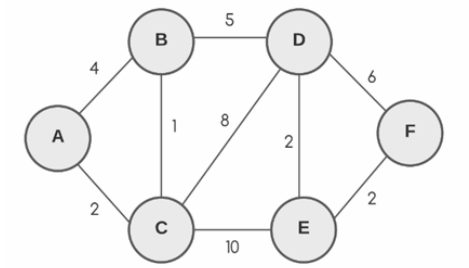


Figure 1. Graph example of Dijkstra's algorithm

In this case, the origin node will be symbolized as “A”. Starting from this initial node, the vertices adjacent to the origin, the closest ones, which in this case would be “B” and “C”, will be evaluated. Successively, the other subnodes continue to be evaluated in order in a way that in each evaluation the differences in weights of the edges that link the nodes are taken into account and following the assignment of labels that correspond to the accumulated value of the size of the arches or edges and the closest origin of the route. These tags can be temporary or permanent/definitive and vary according to the complexity of finding the shortest path [17]. For the cited case, the ideal is to calculate the shortest path between “A” and “F”.

In each step of the algorithm, as previously mentioned, the vertex closest to the origin, which has not yet been visited, is taken into account. That it has not been visited means that it has not been evaluated, and, therefore, this node has the smallest distance calculated from the initial node. Taking into account the above, all the minimum paths are recalculated, taking the node that has not been visited as the intermediate path, until reaching "F" [17].

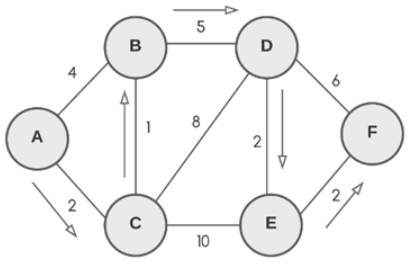


Figure 2. Representation of the search for the shortest path from “A” to “F” using Dijkstra's algorithm.

3.2.1 A* Algorithm (A Star)

The A search algorithm, also called A asterisk or A star, dates back to 1968 and was proposed by Peter E. Hart, Bertram Raphael and Nils John Nilsson. This algorithm is born according to the heuristic model proposed by Nilsson with the purpose of improving Dijkstra's algorithm so that this new version generates greater performance optimization within its operation, taking into account that it seeks to find the path of lowest cost between an initial point (as is the case of the nodes or vertices within the Dijkstra method) and a goal or objective [19]. Within the search parameters, there is a better understanding when deciding which path to follow since certain possibilities to go through are ignored and emphasis is placed on those that can be taken more effectively to go through the graph. It is a simple algorithm that is usually used in video games, which does not require much processing or memory consumption. The following graph shows an example of a first moment that occurs in the algorithm:

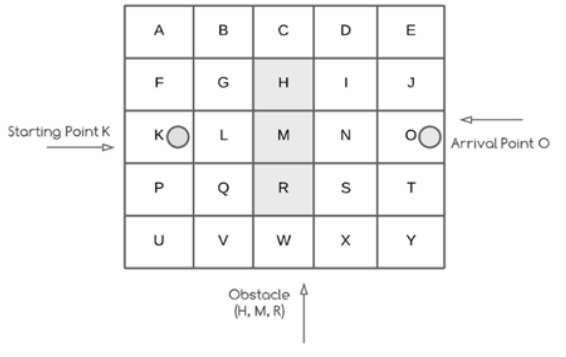


Figure 3. Example of a first moment of algorithm A*

Where K will be the initial point, simulating, for example, an individual who wants to reach O, his point of arrival, and H, M and R being an obstacle between both points. The algorithm consists of a series of steps that start with a chosen node (in this case K). The cost the initial node will have will generally be 0. After having carried out a heuristic evaluation in which the estimated cost from the beginning to the final goal is calculated, the algorithm begins to evaluate the cost of its neighboring nodes from position K and determines what position to follow [15]. Consecutively, when finding the ideal position, the current position that was previously defined by K will become the ideal position and now from this new position, the following best position is evaluated to continue evaluating the neighboring nodes until reaching the goal O.

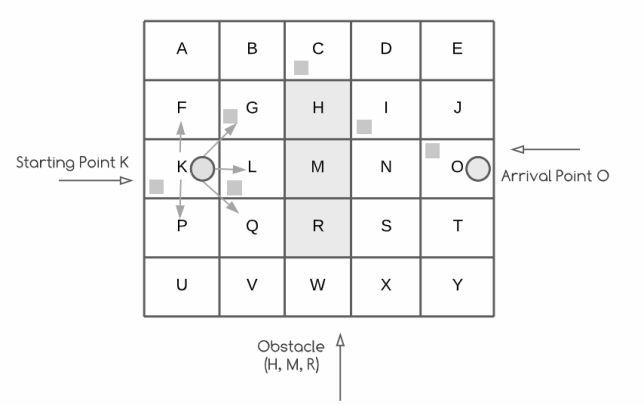


Figure 4. Simulation of A* algorithm operation

3.2.3 Floyd-Warshall Algorithm

The Floyd-Warshall, is an algorithm designed to find the shortest path between the pairs of vertices in a graph that has a numerical value associated with each edge in it.[13] The logarithm works by combining the solutions to the smaller subproblems allowing it to solve the larger starting problem. Therefore to find the quickest route between A and C it compares the quickest route between A and C ever discovered or the quickest route between A and B and B to C[14]. As a first step, we initialize the solution matrix same as the input graph matrix. The solution matrix is then updated by treating each vertex as an intermediate vertex. The plan is to select each vertex one at a time and update any shortest paths that use the selected vertex as an intermediate vertex. Vertices 0, 1, 2, k-1 are already taken into consideration when vertex number k is chosen as an intermediate vertex. There are two potential outcomes for every pair of source and destination vertices (i, j). 1) In the shortest path from I to j, k is not an intermediate vertex. We maintain the current dist[i][j] value. 2) The shortest path from I to j passes through the intermediate vertex k. If dist[i][j] > dist[i][k] + dist[k][j], we update the value of dist[i][j] as dist[i][k] + dist[k][j].[12]

Figure 5. Graph example of Floyd-Warshall algorithm

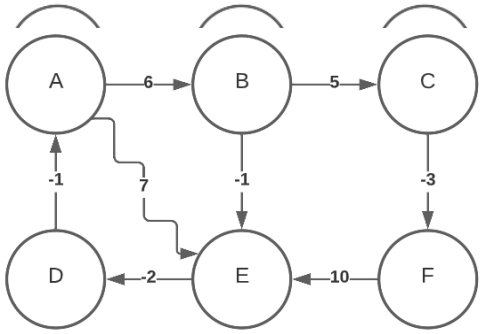


Figure 6. Representation of the search for the shortest path from “A” to “F” using Floyd-Warshall algorithm.

3.2.4 DFS

The algorithm known as "depth-first search" is used to navigate through tree or graph data structures. The algorithm begins at the root node and proceeds to examine each branch as far as it can go before backtracking. it does so by marking the node and moving to the adjacent unmarked node, repeating this process until there is no unmarked adjacent node, then it backtracks while looking for other unmarked node, and if it finds one it explores along the found branch, and ends by printing the nodes along the path. [20]

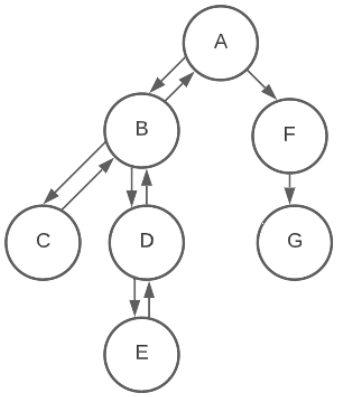


Figure 7. Representation of the search for path from “A” to “G” using Floyd-Warshall algorithm.

4. ALGORITHM DESIGN AND IMPLEMENTATION

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

4.1 Data Structures

The adjacency list using dictionaries was chosen to represent the map of the city, because of the advantages it has when it comes to determining if a vertex X is adjacent to a vertex Y in a graph by doing so in a constant time. Additionally, is one of the most efficient ways to represent the map memory wise. in this data structure each vertex of the graph is associated in a list with all adjacent vertices, in other words there's a main list containing all the vertices that are part of the graph, because we are using dictionaries this "main list" is going to a be a dictionary where the keys are all vertices of the graph and the values are other dictionaries containing the adjacent vertices as the keys and the distance and harassment risk as values. Using dictionaries is a way to make the work easier since we can find the values by using a "key" and not an index. [21]

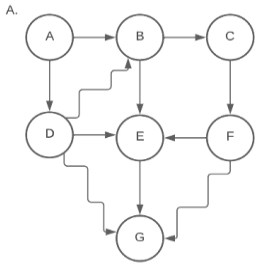


Figure 8. Example of the street map.

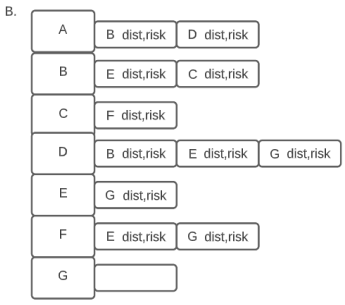


Figure 9. representation of the example of the street map as an adjacency list using dictionaries.

[4]https://github.com/linasofi13/StreetHarassmentProyect_2022-2

4.2 Algorithms

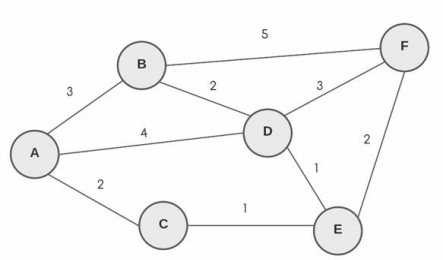
In this work, we propose an algorithm for a path that minimizes both the distance and the risk of street sexual harassment.

4.2.1 Algorithm for a path that reduces both the distance and the risk of street sexual harassment: Dijkstra Algorithm

The chosen algorithm was Dijkstra's algorithm. For the implementation of this algorithm, Priority Queues were used. In a normal queue, items are inserted and the "oldest" item is popped. A priority queue is similar; however, it is possible to associate a priority to each element and in the same way extract the element with the highest priority [22]. For this case, the highest priority is the element with the lowest priority value. With Dijkstra, the priority queue is used to store the nodes, since it contains the unvisited nodes; thus, in each iteration an unvisited node with the lowest weight is extracted from the source, removing them from the queue as they are visited. To link the distance and the risk of street sexual harassment, a variable was generated and represented as follows: weight = distance * risk.

At the beginning of the algorithm [23], a set() type collection is created, in which the nodes that have already been visited will be added, the cost of the origin is initialized to a value of 0 (since at the beginning there will be no cost, standing on the initial node), a "parent" variable is generated that will be very useful when traversing the path, since it will be the representation of the "predecessor" node and a first value of 0 is entered for the priority queue in order to initialize it. As long as the queue can be iterated, it checks if it is not empty, if so, it looks for the node with the lowest cost and then checks if it is in the visited ones. Now, if the queue runs out, then the iteration block would be exited. At first, this condition of not being in the visited ones will always be fulfilled when starting the algorithm, so the condition is true since the node is not in the visited ones. Then, the visited vertex is added and if the vertex is not equal to the goal, the adjacent or “neighbors” to that node or current node are searched with their respective weights. Later, if the adjacent one has already been visited, it is not taken into account again to save time and thus continue to evaluate the weights of the adjacent ones, emphasizing that, by default, these will have a value of "infinity" or not determined while they are being evaluated. Now, the new cost will be updated and will have

the cost of the current node plus the weight (which relates distance * risk) and finally, if the updated cost is less than the previous one, it means that this weight is more beneficial and therefore it is added to the priority queue that new cost together with the adjacent node, the cost of the adjacent node is updated with the new cost and now the predecessor to the adjacent node will be the current node. Finally, it will return "parent" or in other words the predecessors (a dictionary).
To print the path, a function is used that takes parent and destination and creates a list of tuples with the coordinates. The algorithm is exemplified in Figure 10.



	A	B	C	D	E	F
A	0 A	3 A	2 A	4 A	inf	inf
C		3 A	2 A	4 A	3 C	inf
B		3 A		4 A	3 C	8 B
E				4 A	3 C	5 E
D				4 A		5 E
F						5 E

- Shortest path from A to F: A – C – E – F
- Total cost: 5

Figure 10: Implementation of Dijkstra's Algorithm

4.2.2 Calculation of two other paths to reduce both the distance and the risk of sexual street harassment

Bearing in mind that the first algorithm designed was the one explained above, which relates both distance and harassment and multiplies them, this multiplication being the weight within the algorithm, two other Dijkstra algorithms are defined by modifying the variable that determines the weight. To do this, in the second algorithm the variable was redefined as weight = d^{2*r}, where "d" means the distance and "r" the risk of harassment, this algorithm being represented with a green route in the graphed map. For the third algorithm, represented with a purple route in the map, the variable was redefined as weight = d + (80*r) where "d" means the distance and "r" the risk of harassment. For these two algorithms, Dijkstra's algorithm was also implemented, which, as explained before, searches for the path with the least weight going through the graph represented in an adjacency list, with the support of a priority queue that reduces the complexity of the algorithm by removing the smallest value from the queue. The algorithm is exemplified in figure 4.

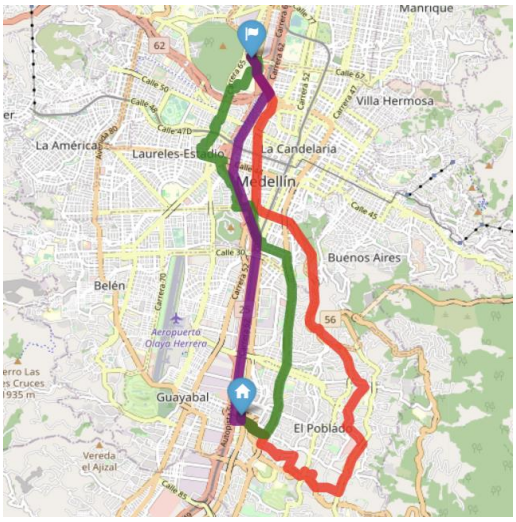


Figure 4: Map of the city of Medellín showing three pedestrian paths that reduce both the risk of sexual harassment and the distance in meters between the EAFIT University and the National University. Red path represents the variable v=d*r, green path represents the variable v=d^{2*r} and the purple path represents the variable v=d+(80r).

4.3 Algorithm complexity analysis

Using a simple Dijkstra's approach, the implementation uses an unsorted array to search through all the vertices and get the closest in the graph, which is not so efficient taking into account that the initial time complexity is O(V) by adding all vertices to the unsorted list (given a graph G = (V, E) represented as an adjacency matrix), then removing a node from the list and adding the time taken for each iteration of the loop is O(V), so time complexity becomes O(V) + O(V) * O(V) = O(V2) [24]. Knowing the above, to improve the implementation of Dijkstra's algorithm, a priority queue is used instead of an unsorted array, as well as an adjacency list instead of an adjacency matrix, in order to have a more efficient indexing. For this case, the graph is represented as an adjacency list G = (V, E) and the time complexity can be calculated as follows: It will be O(V) to set up the initial priority queue and it will be O(E) to go through all the neighbors of the vertices while updating their weight values.

The inner loop time complexity for each iteration, removing one vertex from the priority queue per loop is O(V) and the advantage of a binary heap or priority queue is that extracting the minimum value or the node with the minimum weight takes O(LogV) (getting the node and removing it from the queue). Therefore, the time complexity is O((V + E) Log V).

In terms of memory complexity, the space required by an adjacency list is O(V) where V is the number of nodes in the graph, occurring if every edge has to be visited. Now, for this representation every vertex stores its neighbors and as said before, O(V) is required for a connected graph and O(E) is needed for storing the adjacent nodes of each node in the graph. Therefore, the space complexity is O(V + E) [25]. Finally, for both time and memory complexity, the vertices correspond to the intersections in the city of Medellin, and the edges correspond to the streets that connect the intersections.

Algorithm	**Time complexity**
Dijkstra's Algorithm	O((V + E) Log V)

Table 1: Time Complexity of Dijkstra's algorithm, where V is the number of nodes and E is the number of edges.

Data Structure	**Complexity of memory**
Adjacency List	O(V + E)

Table 2: Memory complexity of the adjacency list where V is the number of nodes and E is the number of edges in the graph.

4.4 Design criteria of the algorithm

Since we did not work with negative weight values, it was decided that the Dijkstra algorithm was the most suitable algorithm for the solution of the shortest and safest path search, for reasons mainly related to its efficiency taking into account that the overall time complexity of the way we implemented Dijkstra's algorithm is O((V + E) Log V) which is very optimum, where V is the number of nodes in the graph and E is the number of adjacent nodes of each node in the graph. And not only was the efficiency of the algorithm taken into account. Most, if not all, of the implemented structures were chosen based on how favorable they were in terms of time and memory complexity.

From the beginning, when looking for the best way to represent the graph, we found that in order to iterate through the outgoing edges of a node efficiently, the adjacency list was the best fit. In our adjacency list each vertex stores its neighbors. To check on each adjacent node, in terms of time, in the worst case they will have to check all neighbors of a node which would be a complexity of O(V) and the memory complexity being O(V+E) which is great for both the time and the memory complexity. Another thing that was taken into account when designing the graph representation, was how we were going to access each node and its neighbors, seeing that the indexing of normal lists can become very confusing to work for the purpose we intended to use it, we chose to work with dictionaries in our adjacency list. Therefore, our adjacency list is a dictionary of dictionaries, where the main one contains as keys all the nodes of the graph and as value of each key of the main dictionary there is another dictionary whose keys are the adjacent nodes (neighbors) and as values the distance and the risk of each edge.

In the implementation of Dijkstra's algorithm, we chose to work with priority queues because by working with them we were able to associate a priority to each component in the queue. The magic of this is that in a priority queue the first value to be extracted is not the oldest, but the one with the highest priority associated with it. In our implementation the highest priority means the element with the smallest priority value since the priorities are the weights of the unvisited nodes. The biggest advantage of using a priority queue is that we are always visiting the node with the smallest weight value, which can make a significant difference in terms of time if a normal queue were to be used. The extraction of the node with the highest priority has a good time complexity of O(log V).

5. RESULTS

In this section, we present some quantitative results on the three pathways that reduce both the distance and the risk of sexual street harassment.

5.1 Results of the paths that reduces both distance and risk of sexual street harassment

Next, we present the results obtained from *three paths that reduce both distance and harassment,* in Table 3.

Origin	Destination	Distance	Risk
Eafit	Unal	15399.647 m	0.33579
Eafit	Unal	12228.437 m	0.6787
Eafit	Unal	5422.5 m	0.5470

Table 3. Distance in meters and risk of sexual street harassment (between 0 and 1) to walk from EAFIT University to the National University.

5.2 Algorithm execution times

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

Calculation of v	Average run times (s)
v = d * r	0.082 s
$v = d^{2*r}$	0.110 s
v = d+(80r)	0.077 s

Table 4: Dijkstra's algorithm execution times for each of the three calculator paths between EAFIT and Universidad Nacional.

6. CONCLUSIONS

According to the implemented methodology, having defined the variable that relates the distance and the risk of harassment in 3 different ways, the results demonstrate the versatility that can exist when calculating a path from one place to another. For example, for the first case of the definition of the variable (weight = distance * risk), a distance of approximately 15 km was obtained from the EAFIT University to the National University, with an approximate risk of 0.3. Comparing this result with the one obtained by the second variable (weight = d^{2*r}), it was observed that, although the second variable considers a shorter distance to travel (approximately 12 km), the risk of street harassment is slightly greater than the distance shown in the first variable, since it is roughly 0.6, so it is notable that in the second path, if the person seeks a shorter path regardless of the harassment, and has both options or paths to take, they can consider the second path. For the third path, the difference was quite considerable, since the distance obtained in meters was approximately 5.5 km when defining the variable as weight = distance + (80 * risk), and the average risk was 0.5, a value that is between the value of the first path and the second path. Taking into account the above, the third path was the most striking to analyze, because, although it considerably reduces the distance, the risk obtained is not that high, but it is not the lowest of all, so this option could be very effective for the user. However, it is necessary to clarify that when graphing the paths it is visible that there is no complete precision between all the points on the map (this could be due to inaccuracy within the dataframe) so the values are approximations but they also allow to understand the behavior of Dijkstra's algorithm and the fluctuations it can produce according to the definitions of the weight variable.

From the perspective of the team that produced this report, it is considered that the implementation and analysis of the algorithm represents a very useful tool for the city of Medellin, since, as it was stated at the beginning, street sexual harassment is a reality faced by thousands of people in their day to day. In addition, not only does this harassment occur in the streets but also in the different means of transport in the city, so that is why sometimes many people could choose to take a pedestrian route instead of taking public transport, which would mean that this investigative analysis

could be very useful for those who want to move from one place to another and would like to reduce the possibility of being harassed. The scope that this proposed solution could obtain could translate into greater security for people in the city, greater visibility of this problem, an incentive to generate more campaigns and pedagogy to deal with this problem, among others.

The execution times are reasonable in the sense that they are fast and efficient, since both the complexity in time and in memory are understandable and it could be a good option in the creation of, for example, a digital platform that is much more familiar to the user, realizing that currently the seen need is that systems and programs run quickly and, in addition, adapt to the needs expressed by the user. That is why it is considered that the three paths obtained can be recommended in the implementation of a digital platform, since it could offer the ability to show, for example, a short path, but with a higher risk, or if the user wishes a path that is longest but has the least risk among the three paths, etc. Finally, to have an even more conclusive vision of what this project could mean for the city, a kind of survey of random citizens of Medellín could be held, who can observe how the project works and if they think they would use it or if it would be a valuable tool for the city.

6.1 Future works

For future work we would like to improve many aspects of our program specially to make it more user friendly, starting with creating a user interface. At the moment, when our program is running all interactions are done through the terminal. Although our program prints a menu with some options to facilitate interaction, showing a real interface would be much better. Another thing we would like to improve in our program is the type of input it receives. Right now, it receives two pairs of coordinates, one of them is the origin and the other is the destination. We are sure that not everybody knows the coordinates of the places they are interested in going to, so we would like the program to receive the name of the origin and the destination to make it easier for everybody to use. In the menu we have created there are some places, but our goal is to extend the possibility of finding the way just by entering the name of the origin and destination, no matter what the origin and destination are. We would also like to add more variables to our algorithm to show us more information about the route it has found, for example the estimated time it would take you to get from the origin to the destination if you follow the route it shows you.

We would also like to be able to implement our program as a web application that people can easily access, if they do not want to download any mobile application. And also, to be able to implement it as a mobile application. Both implementations with a user-friendly interface. Additionally, for future work we would like to implement machine learning. In this fast-changing world, we are in, it is important to keep information up to date, and for this purpose we need machine learning to analyze the data and help us give users more accurate information.

ACKNOWLEDGEMENTS

The realization of this work was possible thanks to the continuous support of the teacher assistants of the Data Structures I course of the EAFIT University and the students of this class, since they were of great help when implementing the different programming techniques, guiding the project and offering the necessary feedback on each delivery.

The first two authors also wish to thank their parents for their support during the project.

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. Arancibia, J., Billi, M., Bustamante, C., Guerrero, M., Meniconi, L., Molina, M. and Saavedra, P., 2015. Acoso Sexual Callejero: Contexto y dimensiones. OCAC Chile. Retrieved August 6, 2022, from: https://www.ocac.cl/wp-content/uploads/2016/09/Acoso-Sexual-Callejero-Contexto-y-dimensiones-2015.pdf

2. Street Harassment. myUSF. University of San Francisco. (n.d.). Retrieved August 6, 2022, from: https://myusf.usfca.edu/title-ix/street-harassment

3. El 90,1 por ciento de las mujeres no denuncia el acoso callejero. 2019. El Tiempo. Retrieved August 6, 2022, from: https://www.eltiempo.com/colombia/medellin/el-90-1-por-ciento-de-las-mujeres-no-denuncia-el-acoso-callejero-en-medellin-355056

4. Siempre Seguras. 2019. Retrieved August 6, 2022, from: https://contactosiempreseg.wixsite.com/siempre-seguras-1

5. Forbes Staff, 2021. Siempre Seguras: la app para mapear el acoso sexual callejero en México. Forbes México. Retrieved August 6, 2022, from: https://www.forbes.com.mx/tecnologia-siempre-seguras-app-mapear-acoso-sexual-callejero-mexico/

6. Safecity. About Us. (n.d.) Retrieved August 6, 2022, from: https://www.safecity.in

7. Manzatto, A., 2022. Using Heatmap and Machine Learning to Predict Sexual Abuse Hotspots. Omdena. Retrieved August 6, 2022, from: https://omdena.com/blog/heatmap-machine-learning/

8. Preventing Sexual Harassment with Artificial Intelligence. Omdena. (n.d.). Retrieved August 6, 2022, from: https://omdena.com/projects/sexual-harassment/

9. Adeel, M., 2013. Understanding Dijkstra's Algorithm. Retrieved August 7, 2022, from: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2340905

10. Graphs. Isaac Computer Science. (n.d). Retrieved August 7, 2022, from: https://isaaccomputerscience.org/concepts/dsa_datastruct_graph?examBoard=all&stage=all

11. Algoritmo de Dijkstra. EcuRed. (n.d). Retrieved August 7, 2022, from: https://www.ecured.cu/Algoritmo_de_Dijkstra

12. GeeksforGeeks 2012. Floyd Warshall Algorithm | DP-16 - GeeksforGeeks. Retrieved August 14, 2022 from https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

13. Programiz. Floyd-Warshall Algorithm. Retrieved August 14, 2022 from https://www.programiz.com/dsa/floyd-warshall-algorithm

14. Brilliant | Brilliant Math & Science Wiki. Floyd-Warshall Algorithm | Brilliant Math & Science Wiki. Retrieved August 14, 2022 from https://brilliant.org/wiki/floyd-warshall-algorithm/

15. Tech with Tim, 2020. A* Pathfinding Visualization Tutorial - Python A* Path Finding Tutorial. Retrieved August 7, 2022, from: https://www.youtube.com/watch?v=JtiK0DOeI4A

16. Alexandra Richards. 2018. This new mobile app maps out catcalling and sexual harassment across London. Evening Standard. Retrieved August 14, 2022 from https://www.standard.co.uk/news/uk/new-app-maps-out-catcalling-and-sexual-harassment-across-london-a3780326.html

17. Programiz. Dijkstra's Algorithm. (n.d). Retrieved August 7, 2022, from: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

18. Grove, 2015. The cartographic ambiguities of HarassMap: Crowdmapping security and sexual violence in Egypt retrieved August 7,2022, from: https://www.researchgate.net/publication/281478814_The_cartographic_ambiguities_of_HarassMap_Crowdmapping_security_and_sexual_violence_in_Egypt

19. Graph Everywhere. Algoritmo A*. (n.d). Retrieved August 7, 2022, from: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

20. 1.2012. Depth First Search or DFS for a Graph - GeeksforGeeks. GeeksforGeeks. Retrieved August 14, 2022 from https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

21. Una lista de adyacencia. Pythoned. Retrieved October 12, 2022, from: https://runestone.academy/ns/books/published/pythoned/Graphs/UnaListaDeAdyacencia.html

22. Actually implementing Dijkstra's Algorithm. Nil Manano Blog, 2020. Retrieved October 13, 2022, from: http://nmamano.com/blog.html

23. Harold, L., 2018. Dijkstra's Algorithm in Python using PriorityQueue. Retrieved October 14, 2020, from: https://gist.github.com/qpwo/cda55deee291de31b50d408c1a7c8515

24. Baeldung, 2021. Understanding Time Complexity Calculation for Dijkstra Algorithm. Retrieved November 11, 2022, from: https://www.baeldung.com/cs/dijkstra-time-complexity

25. GeeksforGeeks, 2021. Comparison between Adjacency List and Adjacency Matrix representation of Graph. Retrieved November 11, 2022, from: https://www.geeksforgeeks.org/comparison-between-adjacency-list-and-adjacency-matrix-representation-of-graph/