SHORTEST PATH FINDING ALGORITHM TO PREVENT STREET HARASSMENT

Santiago Neusa Ruiz
Universidad Eafit
Colombia
sneusar@eafit.edu.co

Manuela Castaño Franco
Universidad Eafit
Colombia
mcastanof1@eafit.edu.co

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

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

ABSTRACT

The streets are a vital space for the development of society, but the interactions that take place there are not always pleasant. Sexual harassment is a reality, and it is our duty to fight it. This paper presents the creation of an algorithm aimed at solving the need to find routes that are short in distance, involving the least amount of street harassment possible. The project is contextualized in the present, while providing some background (similar algorithms) in order to show previously developed alternatives besides our support for the fight against sexual street harassment. The proposed algorithm is Dijkstra, which, based on a variable v that combines the distance and harassment risk of every street, generates three different paths from one point in the city to another. There is an evident contrast between the paths, which shows that the redefinition of the variable v will greatly impact the outcome of the generated path. The total running time of the three paths is under 1 second, which is quite fast thanks to the implementation of a priority queue. This is promising for Medellín's future and could potentially help lower the number of victims from street harassment.

Key words

Shortest route, street sexual harassment, identification of safe routes, crime prevention

1. INTRODUCTION

Sexual harassment is a problem that has been present in our society for decades. We tend to think of solutions such as increasing education levels or increasing penalties or punishments for those who commit such an act, but science has a different approach. Different factors have led to develop a solution from technology, since many daily processes are involved by this, it can be a great tool to fight this problem. We propose an algorithm that takes into account the distance and the level of harassment in the streets of Medellín, in order to find the best route that a person can take when going from a point A to a point B. The proposed method does not leave aside the ideas presented at the beginning of this section, but complements them.

1.1. The problem

The problem we are facing is the creation of an algorithm that identifies the three shortest and safest paths, in order to reduce the crime and sexual harassment risk pedestrians face on a daily basis. Thereby, the likelihood of these encounters will decrease; this will let pedestrians, especially vulnerable groups like women and children, feel safe while walking towards their destination.

1.2 Solution

The solution suggested is three pedestrian paths that reduce both distance and the risk of harassment. From the algorithms presented in this work, the Depth First Search and Breadth First Search algorithms were discarded given that they were not designed to find the shortest path or safest path. Additionally, they cannot be applied to weighted graphs, therefore, they are not useful for this particular problem. Instead, the Dijkstra algorithm was selected and modified. Since Dijkstra's algorithm specifically finds all the shortest paths from a node to every single point in a graph, it is crucial to modify it so that it stops when the target destination is reached. Simultaneously, a weight variable composed of a combination of the distance and the harassment risk was considered in order to generate a route that creates a balance of these two factors.

For all three paths, the same algorithm is used. Only the parameter of weight that is passed to the function is changed. This weight expresses the length and risk in three different ways, fusing them through different operations as a means to get diverse results for a short and safe path.

1.3 Structure of the article

Next, in Section 2, we present work related to the problem. Then, in Section 3, we present the datasets and methods used in this research. In Section 4, we present the algorithm design. Then, in Section 5, we present the results. Finally, in Section 6, we discuss the results and propose some directions for future work.

2. RELATED WORK

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

2.1 Urban Navigation Beyond Shortest Route: The Case Of Safe Paths

This study is labeled as an intelligent urban navigation work and it was focused on the cities of Chicago and Philadelphia, generating a bicriteria solution to the safest and shortest path problem. The two factors taken into consideration were the length and risk of the route. For the construction of the algorithm, data from OpenStreetMap (OSM) was exported and publicly available crime information was recollected.

The algorithm is recursive, and was based mostly on the Dijkstra solution. This algorithm proved a great performance, as shown in the results through efficiency and efficacy. Finally, the program displays not only one but rather a small subset of the best path options for the pedestrian to choose from. [1]

2.2 Proposing A Multi-Criteria Path Optimization Method In Order To Provide A Ubiquitous Pedestrian Wayfinding Service

This study presents a more complete method of calculating paths tailored to pedestrians. Put to test in Tehran, Iran, this method focused on context and user awareness to give the best path option. These two factors are then broken down into four criteria: length, safety, difficulty and attraction. The last three also have, inside each one of them, diverse categories that take into account the user's age, gender, infrastructure preference, as well the street's width and slope. The above, with the intention to deliver the best result personalized for each user.

This method made use of the Dijkstra algorithm. The four factors go through a mathematical model, where they are added and generate one final path option. [2]

2.3 SafeJourney: A Pedestrian Map Using Safety Annotation For Route Determination

This paper showcases the creation and implementation of a website to calculate the best route for students at Universiti Teknologi PETRONAS, a private university in Malaysia. The scope of the project was small, only focused on the commute students have to do inside the campus, e.g. from one block to another and so on.

The solution was mainly based on the Dijkstra algorithm. The website was built using XHTML, PHP and MySQL to display the best path highlighted on the university's campus map. It also shows step by step instructions through text and images. [3]

2.4 Shortest Path Algorithms for Pedestrian Navigation Systems

This work presents a penalty-based algorithm to calculate the shortest and most accessible path for people with reduced mobility, emphasizing mostly on wheelchair users.

The algorithm is based on the k shortest paths model (KSP), while also contemplating factors like the road's pavement state, its width, and whether it is a ramp or not. The 10 shortest paths are calculated and then the most accessible one is chosen to be delivered to the user. The method was tested in the city of Thessaloniki, in Greece, proving great performance in most of the conducted experiments. [4]

3. MATERIALS AND METHODS

In this section, we explain how the data were collected and processed, and then different alternative path algorithms that reduce both the distance and the risk of sexual street harassment.

3.1 Data collection and processing

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

For this project, a linear combination (LC) was calculated that captures the maximum variance between (i) the fraction of households that feel insecure and (ii) the fraction of households with incomes below one minimum wage. These data were obtained from the 2017 Medellín quality of life survey. The CL was normalized, using the maximum and minimum, to obtain values between 0 and 1. The CL was obtained using principal components analysis. The risk of harassment is defined as one pminus the normalized CL. Figure 1 presents the calculated risk of bullying. The map is available on GitHub[3] .

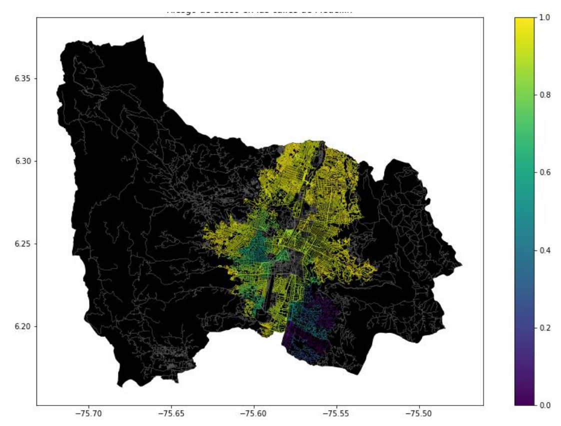


Figure 1. Risk of sexual harassment calculated as a linear combination of the fraction of households that feel unsafe and the fraction of households with income below one minimum wage, obtained from the 2017 Medellín Quality of Life Survey.

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

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

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

3.2 Algorithmic alternatives that reduce the risk of sexual street harassment and distance

In the following, we present different algorithms used for a path that reduces both street sexual harassment and distance.

3.2.1 Depth First Search

The Depth First Search (DFS) is an algorithm for deeply searching or exploring graphs or trees. [5] The execution of the algorithm begins at the root node, examining each branch before backtracking. It uses a stack data structure to get the subsequent vertex, and to start a search, whenever a dead-end appears in any iteration. [6]

Because depth-first search constantly grows until the deepest node in the current branch of the search tree, we implement it using the LIFO queue, often known as a stack. According to Norvig and Russell [7]:

> "A LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent — which, in turn, was the deepest unexpanded node when it was selected."

One notorious problem we may encounter with DFS is the possibility of having the target node next to the current node, but since DFS goes to the end of each branch, we cannot reach it until that branch is fully explored.

The time complexity of DFS is O(V + E) with an adjacency list and O(V^2) with an adjacency matrix, where V is the number of vertices and E the number of edges [14].

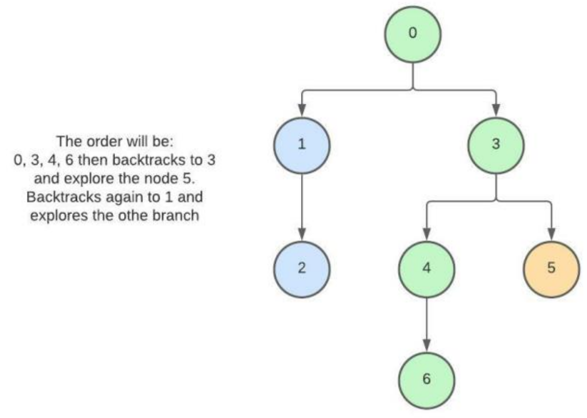

The order will be:
0, 3, 4, 6 then backtracks to 3 and explore the node 5. Backtracks again to 1 and explores the othe branch
0
1
3
2
4
5
6

Figure 2. DFS Algorithm.

3.2.2 Breadth First Search

Breadth-first search (BFS) is an algorithm that is used for searching or exploring graphs or trees [8]. This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. BFS accesses these nodes one by one. Once the algorithm visits and marks the starting node, it moves towards the nearest unvisited nodes and analyses them [9]. These iterations continue until all the nodes of the graph have been successfully visited and marked, or until the goal is reached.

BFS implements the FIFO queue (first in, first out) because when queried, returns the oldest element, based on the order they were inserted [8].

The BFS algorithm helps to solve the problem left by DFS.

The time complexity of BFS is O(V + E) with an adjacency list and O(V^2) with an adjacency matrix, where V is the number of vertices and E the number of edges [14].

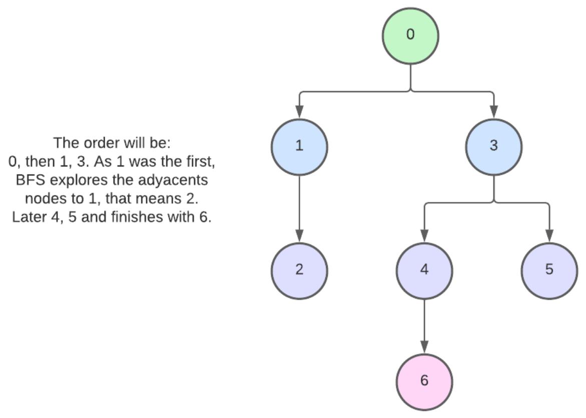

The order will be:
0, then 1, 3. As 1 was the first, BFS explores the adyacents nodes to 1, that means 2. Later 4, 5 and finishes with 6.
0
1
3
2
4
5
6

Figure 3. BFS Algorithm.

3.2.3 Dijkstra's Shortest Path Algorithm

Dijkstra's algorithm is one way to find the shortest route across a weighted graph from a starting node to a target node. This procedure builds a tree of all the shortest routes from the source, or first vertex, to every other point in the graph. By calculating the travel time to each linked node, Dijkstra's algorithm determines the shortest path. It then records the edge's distance in a list, placing the shortest at the top, and begins exploring other nodes in the order of that list. Just until the destination node is reached, this process stops [10].

This method differs from the others in that it provides the shortest distance, whereas DFS and BFS just provide an answer to whether a node is connected to another.

One significant problem with this algorithm is the fact that even if the shortest path is the better option, sometimes that's not the way you can get to the target node. This algorithm does not have any orientation variable [11].

The time complexity of this algorithm with an adjacency matrix is $O(V^2)$. But, when paired with an adjacency list and a priority queue, the complexity becomes $O((V + E)\log V)$, where V is the number of vertices and E the number of edges [15].

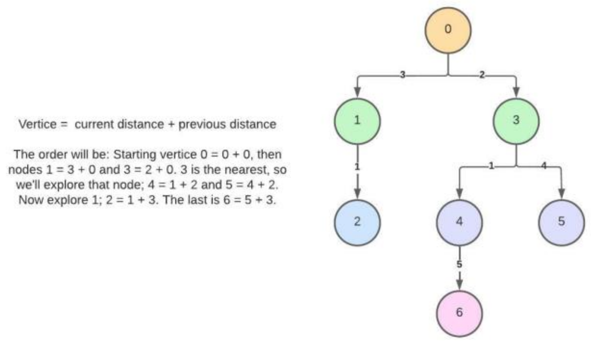

Vertice = current distance + previous distance
The order will be: Starting vertice 0 = 0 + 0, then nodes 1 = 3 + 0 and 3 = 2 + 0. 3 is the nearest, so we'll explore that node; 4 = 1 + 2 and 5 = 4 + 2. Now explore 1; 2 = 1 + 3. The last is 6 = 5 + 3.
0
3
2
1
3
1
1
4
2
4
5
5
6

Figure 4. Dijkstra's algorithm.

3.2.4 A* (A Star algorithm)

A* (pronounced A Star) is a popular pathfinding and graph traversal technique. In order to make more optimum decisions, the A* algorithm adds heuristic to a standard graph-searching algorithm, essentially planning ahead at each step so a more optimal decision is made. Like Dijkstra, A* constructs a lowest-cost path tree from the start node to the destination node, but it differs from Dijkstra in the way it utilizes a function called f(n) for each node to estimate the overall cost of a path by adding the edge cost and a heuristic variable. In this instance, that outcome will be at the front of our queue [12].

The heuristic variable helps to avoid the problem Dijkstra has. With 2 variables in game the judgment is better.

The time complexity depends on the heuristic variable, so it may vary. Though we can define it as O(E), where E is the number of edges, in the worst case where it would have to visit all the nodes in the graph.

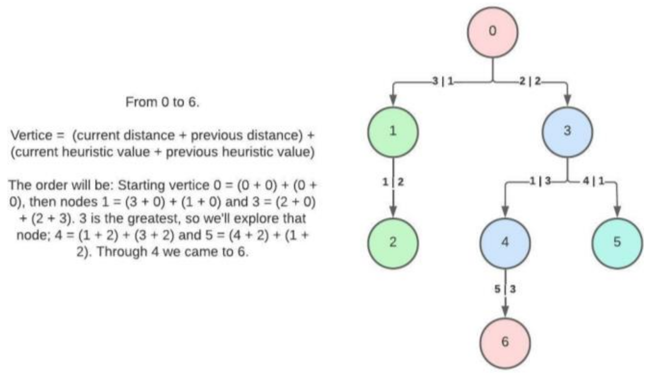

From 0 to 6.
Vertice = (current distance + previous distance) + (current heuristic value + previous heuristic value)
The order will be: Starting vertice 0 = (0 + 0) + (0 + 0), then nodes 1 = (3 + 0) + (1 + 0) and 3 = (2 + 0) + (2 + 3). 3 is the greatest, so we'll explore that node; 4 = (1 + 2) + (3 + 2) and 5 = (4 + 2) + (1 + 2). Through 4 we came to 6.
0
3 | 1
2 | 2
1
3
1 | 2
1 | 3
4 | 1
2
4
5
5 | 3
6

Figure 5. A 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 data of the streets of Medellín was given in a CSV file. Firstly, all of the information was translated to a Pandas DataFrame. Then, an adjacency list was implemented using Python dictionaries.

This adjacency list was preferred over an adjacency matrix due to the type of graph of this particular case. For instance, the map of Medellín could be categorized as a sparse graph, since each street is only directly connected to a few other streets, thus, an adjacency list is more efficient in terms of storage [13].

The adjacency list was carried out with Python dictionaries to represent the weight of each edge in the graph. To illustrate, for every key, or origin in the DataFrame, there are multiple destinations. Likewise, for every destination there is a length, a harassment risk, and an additional variable that combines both of them. The above corresponds to the path cost for each edge in the graph.

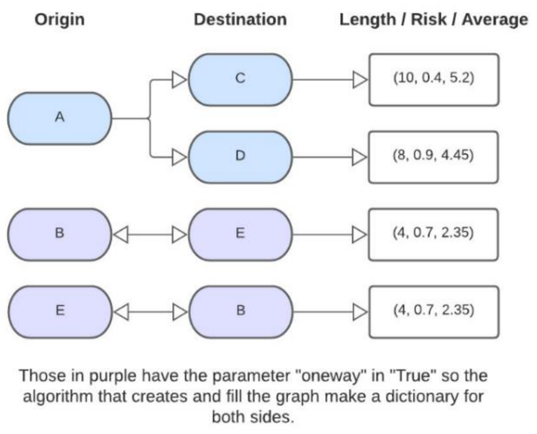

Origin
Destination
Length / Risk / Average
A
C
(10, 0.4, 5.2)
D
(8, 0.9, 4.45)
B
E
(4, 0.7, 2.35)
E
B
(4, 0.7, 2.35)
Those in purple have the parameter "oneway" in "True" so the algorithm that creates and fill the graph make a dictionary for both sides.

Figure 6. Graph as an adjacency list with dictionaries.

[4] https://github.com/sneusar/ST0245-002

4.2 Algorithms

In this paper, 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 pedestrian path that reduces both distance and risk of sexual street harassment

The implemented algorithm was Dijkstra, as it offers simplicity and speed. The parameters sent to the function are the graph, the origin and the target destination. The latter consists of a tri-tuple where the first slot is occupied by the length, the second by the harassment risk, and the third one by a mixture of both.

First and foremost, the algorithm creates four dictionaries. The first three are the distances between each vertex and its predecessor, the harassment risk, and a variable v that combines these two. These dictionaries are initialized to infinity since those values are not known yet. The other dictionary initializes the predecessors of all vertices to "None", except for the origin. The distance between the origin and its predecessor is zero, as well as its harassment risk and variable v, so we need to specifically initialize those to zero.

Subsequently, a priority queue is created. This is a queue that differs from a conventional queue. It inherently stores elements by an order, this is, from smallest to largest. The first elements introduced to the priority queue are the origin and its length, risk, and variable v.

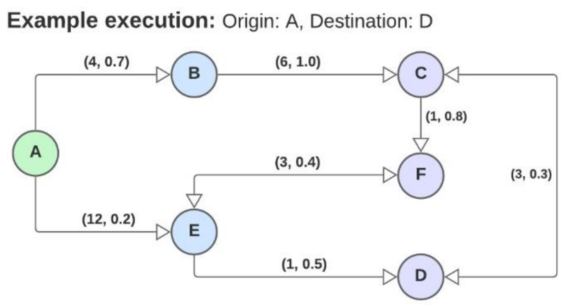
That is one of the biggest advantages and the main reason as to why this data structure was paired with Dijkstra algorithm; due to the fact that the nodes with the shortest length are always the next in line to be explored, we can ensure more efficiency, and avoid longer traversal times.

As long as the priority queue is not empty, the algorithm will keep running. Meanwhile, we use the pop() method to get the first element in the queue, and its values will be stored in a length variable, a risk variable, a combination of both, and a current vertex variable. The latter will be constantly changing throughout the algorithm according to the elements being introduced and extracted from the priority queue. It is important to note that this current vertex has adjacent nodes, and we are going to compare them and their values in an internal loop.

Now, the loop employs three variables which get updated with every iteration. Those variables add the distance, risk, and variable v between the origin and the current node (the predecessor of the adjacent node) and between the current node and its adjacent. If the variable v is smaller than the one that was calculated previously, this value will be introduced to the priority queue, otherwise, it will not. As seen in the beginning, the three dictionaries' values are defined as infinity, so we will replace them with actual values that arise during the process.

At last, an empty stack is created and this is returned along with the dictionary of the predecessors. This information will be useful later on to store the predecessors of the target destination and eventually graph all of them with the help of an external function. This function is critical since Dijkstra finds the predecessors of many routes, not only the one to the target, so we start searching recursively for the predecessors of the destination until we reach the origin.

The algorithm is exemplified in Figure 7.



Dictionaries

Vertex	Path	Cost
A	None	0
B	A	2.35
C	C	5.85
D	E	6.85
E	A	6.1
F	C	6.75

Priority queue

Information	1°	2°
From:	C	E
To:	D	F
Cost:	6.85	7.8

Figure 7. Dijkstra's algorithm using a priority queue.

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

The other two paths were calculated using the same Dijkstra algorithm as presented in section 4.2.1. The only modification is the cost passed to the function, since the variable v is redefined in two other ways with the purpose of obtaining paths with different harassment risks and distances.

The Figure 7 shows the algorithm used for all three paths. The column "Cost" is what varies, since two different operations combining the distance and risk are made.

The algorithm is exemplified in Figure 8.

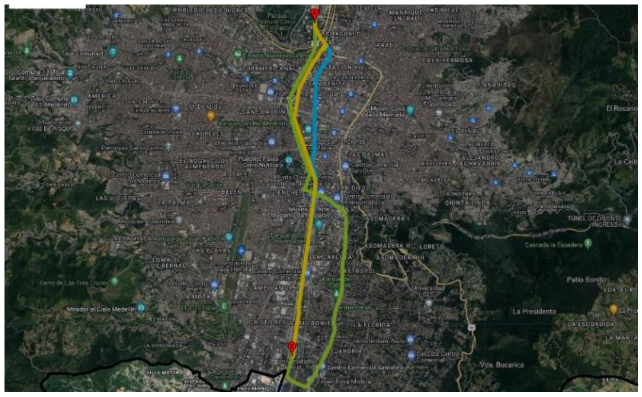


Figure 8: Map of the city of Medellín showing three pedestrian paths that reduce both the risk of sexual harassment and the distance in meters between the EAFIT University and the National University.

4.3 Algorithm complexity analysis

In this case, Dijkstra's algorithm was implemented using an adjacency list and a priority queue. As a result, we obtain a significant reduction of the overall complexity of the program.

In particular, the time complexity becomes O((V + E) log V), being V the vertices of the graph, or intersections in the city, and E the edges, or streets in the map. For the worst case, during the outer loop of the algorithm, it would be necessary to traverse and add every single node to the priority queue. Consequently, inside the inner loop, we would have to visit all the adjacent nodes to the current node, so the time complexity for this part turns out to be O(V+E). Additionally, the priority queue operations have an established time complexity of O(log N) [16] or in this case O(log V) for every node in the graph, which gives our algorithm a final time complexity of O((V + E) log V).

Lastly, if all the vertices or nodes had a direct link to all the other nodes in the graph, the space complexity would be O(V^2). But analyzing Medellín's map, this is impossible since every street is not connected to all the streets in the city. Finally, our space complexity drops down to O(V).

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

Table 1: Time complexity of the Dijkstra Algorithm where V is the number of vertices and E is the number of edges or streets in the city.

Data Structure	Complexity of memory
Adjacency List	O(V)

Table 2: Memory complexity of the adjacency list where V is the number of vertices.

4.4 Algorithm design criteria

The algorithm was designed with approachability in mind. For this reason, we made it possible so that the user can type in the name of a place, instead of the coordinates of their origin and destination. The program outputs a GUI for a more pleasant experience, and it receives the string of text and internally converts it to the coordinates, to then pass it onto the other functions.

Additionally, not only do we need to focus on what to choose for the algorithm, but also how to implement it efficiently. That is why we worked with dictionaries to create the adjacency list, since Python dictionaries allow faster traversal and lookup times than a regular list. As previously mentioned, the adjacency list was preferred over an adjacency matrix due to the differences in terms of space complexity.

Following this line of thought, a priority queue, or also called min heap, was implemented so that the time complexity of Dijkstra's algorithm would significantly drop down. This happens for two reasons; first, the elements in the front of the queue are always the ones with the lowest distances (thus the name min heap); and second, priority queue operations always perform in a O(log N) time [16] which will keep the running time rather small. Moreover, telling the algorithm to stop when it finds the target node also minimizes costs.

Also, in order to generate the path and store all the nodes, a Python deque() is used instead of a regular list since appending an element is done in O(1) time [17].

Finally, we employed the GmPlot library to graph the resulting path in view of the fact that it is easy to use and allows the user to get comfortable with the route and visualize it better.

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	7744.230	0.691
Eafit	Unal	7906.476	0.368
Eafit	Unal	7977.367	0.788

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) /2	0.077
v = d^r * r	0.176
v = (d * 2π) + (r * 1000)	0.788

Table 4: Dijkstra's algorithm execution times for each of the three calculator paths between EAFIT and Universidad Nacional, where d is the distance of each street and r the harassment risk.

6. CONCLUSIONS

This project showed great results, and is very promising for Medellín's future since it could help people be safer in the streets if implemented correctly.

After several tests, it can be concluded that the three paths are very different from one another thanks to the redefinition of the variable v, which combines the length and harassment risk of every street. It was noted that better results are visualized when the origin and destination are far apart, because the variable v plays a bigger role, whereas if the origin location and target are closer to each other, not many different routes are available between them and the final generated paths may overlap.

The algorithm proved to be highly efficient, with low running times and a correct functioning of all of the parts that integrate it. The time to calculate the three paths falls under the 1 second mark, which proves to be quite fast and exactly what a real user would prefer when trying to get from one point in the city to another.

There is a big contrast between the total distance and the average harassment risk of all paths, which shows that by modifying the variable v, many possibilities are created. Particularly, the first definition of the variable which is an average between the length and risk, exhibits better results when compared to the other two paths. The total distance is usually the shortest, the overall risk is neither the highest nor the lowest, and the running time is the fastest, which would be a great candidate if applied to a mobile or web application due to the fact that it is the more appropriate for the user.

To conclude, all of the above indicates that the algorithm can be carried out so that people can feel safer when walking through Medellín while avoiding dangers and without sacrificing a short route. This could potentially help lower the number of victims from sexual assault, theft, and other crimes. We hope that this work encourages the creation of new projects that continue exploring more possibilities of protecting the well-being of the citizens through improving path finding algorithms.

6.1 Future work

To let the user have a better and more convenient experience, a GUI was implemented. However, this was a very simple and basic approach that can definitely be improved if the project is brought to life. So, in the first instance, we would like to refine the graphical and aesthetic aspect of the program.

Secondly, we wish to be able to collect information and keep a record of all the origins and destinations the user has entered.

Additionally, with statistics, we hope to expand the scope of the algorithm through demographics. After knowing the users' ages, gender, frequented places, and such, a more suitable path can be tailored for every individual.

Lastly, through machine learning, it is important to be able to recommend and predict better paths based on the data of each user. And, at a general level, be able to identify paths that have an excessive harassment risk and suppress it from the deck of possible routes.

ACKNOWLEDGEMENTS
We are grateful to the teacher assistants of Data Structures and Algorithms I who were always present and willing to answer all the questions that arose during the process.

The authors thank Professor Juan Carlos Duque, Universidad EAFIT, for providing the data from the 2017 Medellín Quality of Life Survey, processed in a *Shapefile*.

REFERENCES

1. Galbrun E., Pelechrinis K., and Terzi E. Urban navigation beyond shortest route: The case of safe paths. *ScienceDirect, 57.* 160-171. Retrieved August 15, 2022 from https://www.sciencedirect.com/science/article/pii/S0306437915001854

2. Sahelgozin, M., Sadeghi-Niaraki, A., and Dareshiri, S., Proposing a multi-criteria path optimization method in order to provide a ubiquitous pedestrian wayfinding service. in *International Conference on Sensors & Models in Remote Sensing & Photogrammetry*, (Kish Island, Iran, 2015), International Society for Photogrammetry and Remote Sensing. 639–644. Retrieved August 15, 2022 from https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-1-W5/639/2015/

3. K. H. Yew, T. T. Ha and S. D. S. J. Paua, SafeJourney: A pedestrian map using safety annotation for route determination. in *2010 International Symposium on Information Technology*, (2010), IEE, 1376-1381. Retrieved August 15, 2022 from https://www.researchgate.net/publication/224171685_S

afeJourney_A_pedestrian_map_using_safety_annotation_for_route_determination

4. Koritsoglou, K., Tsoumanis, G., Patras, V., and Fudos, I, Shortest Path Algorithms for Pedestrian Navigation Systems 2022. *Information 13*, 6, 269. Retrieved August 15, 2022 from https://www.mdpi.com/2078-2489/13/6/269.

5. En.wikipedia.org. 2022. Depth-first search - Wikipedia. Retrieved August 13, 2022 from https://en.wikipedia.org/wiki/Depth-first_search

6. Walker, A., 2022. BFS Vs. DFS: Know the Difference with Example. Guru99. Retrieved August 13, 2022 from https://www.guru99.com/difference-between-bfs-and-dfs.html

7. Norving, P. and Russell, S. Artificial Intelligence, A Modern Approach, Thirth Edition. Pearson Education, Inc., New Jersey, 2010.

8. Khan Academy. 2022. The breadth-first search algorithm (BFS). Khan Academy. Retrieved August 13, 2022 from https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/the-breadth-first-search-algorithm

9. Sehgal, A., 2022. Breadth First Search | BFS Algorithm - Scaler Topics. Retrieved August 13, 2022 from https://www.scaler.com/topics/data-structures/breadth-first-search/

10. Stack Abuse. 2022. Graphs in Python - Theory and Implementation. Retrieved August 14, 2022 from https://stackabuse.com/courses/graphs-in-python-theory-and-implementation/lessons/dijkstras-algorithm

11. Bullinaria, J. Lecture Notes for Data Structures and Algorithms. School of Computer Science University of Birmingham, Birmingham, 2019.

12. Simplilearn. 2022. A* Algorithm Concepts and Implementation. Retrieved August 14, 2022 from https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

13. Bradfield School of Computer Science. (N.D.) Representing a Graph. Retrieved October 15, 2022 from https://bradfieldcs.com/algos/graphs/representing-a-graph/

14. Geeks for Geeks, 2022. Difference between BFS and DFS. Retrieved November 4, 2022 from https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/#:~:text=The%20Time%20complexity%20of%20BFS,and%20E%20stands%20for%20edges

15. Geeks for Geeks, 2022. Dijkstra’s Algorithm for Adjacency List Representation | Greedy Algo-8. Retrieved November 4, 2022 from https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/

16. Yujian, T., August 17, 2022. Python HeapQ Use Cases and Time Complexity. Medium. Retrieved November 5, 2022 from https://medium.com/plain-simple-software/python-heapq-use-cases-and-time-complexity-ee7cbb60420f

17. Python.org. Collections — Container datatypes. Retrieved November 5, 2022 from https://docs.python.org/3/library/collections.html#collections.deque