PATHFINDING ALGORITHM FOR SEXUAL HARASSMENT PREVENTION, APPLIED TO MEDELLÍN-COLOMBIA

UNIVERSIDAD EAFIT®

Presentation of the team

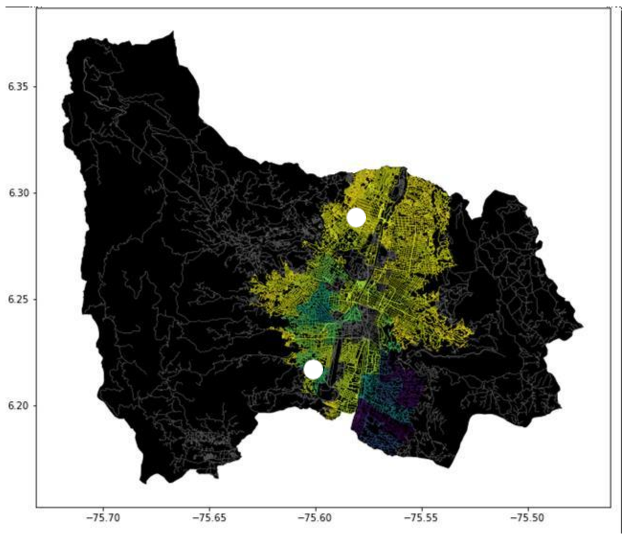
Carlos Mazo
Author

Rafael Urbina
Author

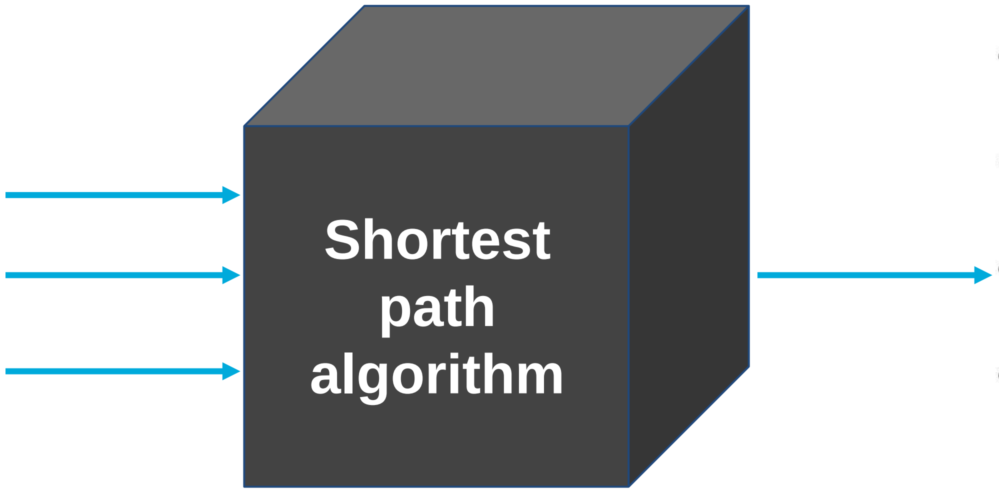
Andrea Serna
Literature review

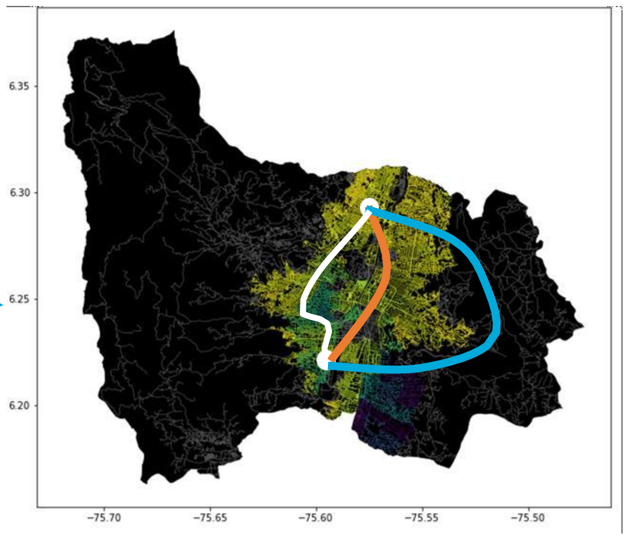
Mauricio Toro
Data preparation

Problem Statement



Streets
of Medellín,
Origin and
Destination



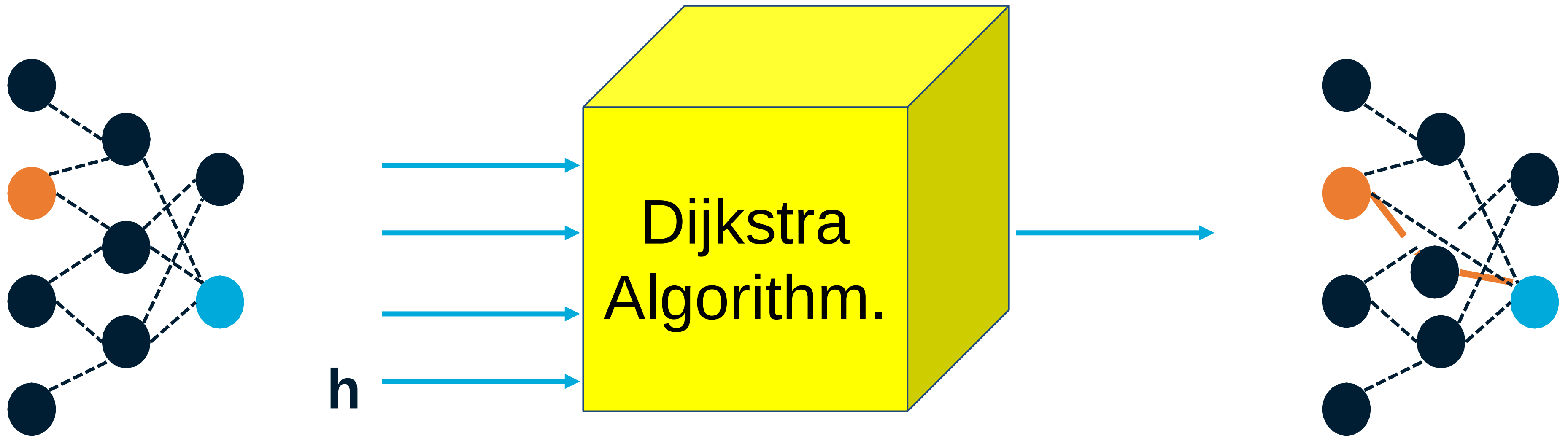


Three paths that reduce both the risk of harassment and distance

https://github.com/RafaelUrbina/st245-002

UNIVERSIDAD EAFIT®

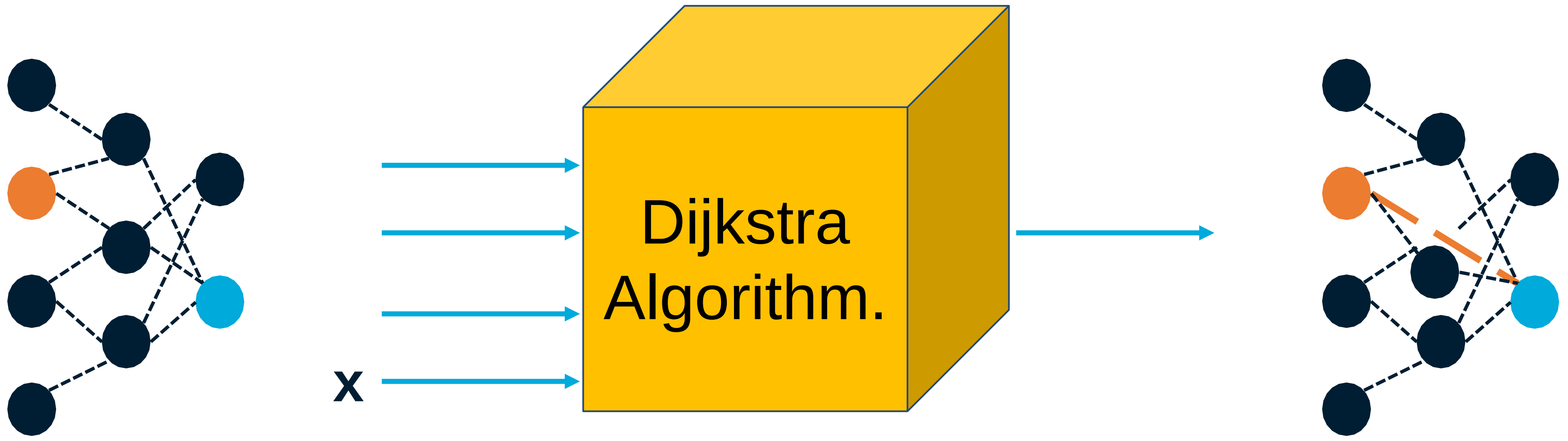
Solution Algorithm



Streets of Medellín, Origin and Destination

A path that reduces both distance and harassment

Solution Algorithm



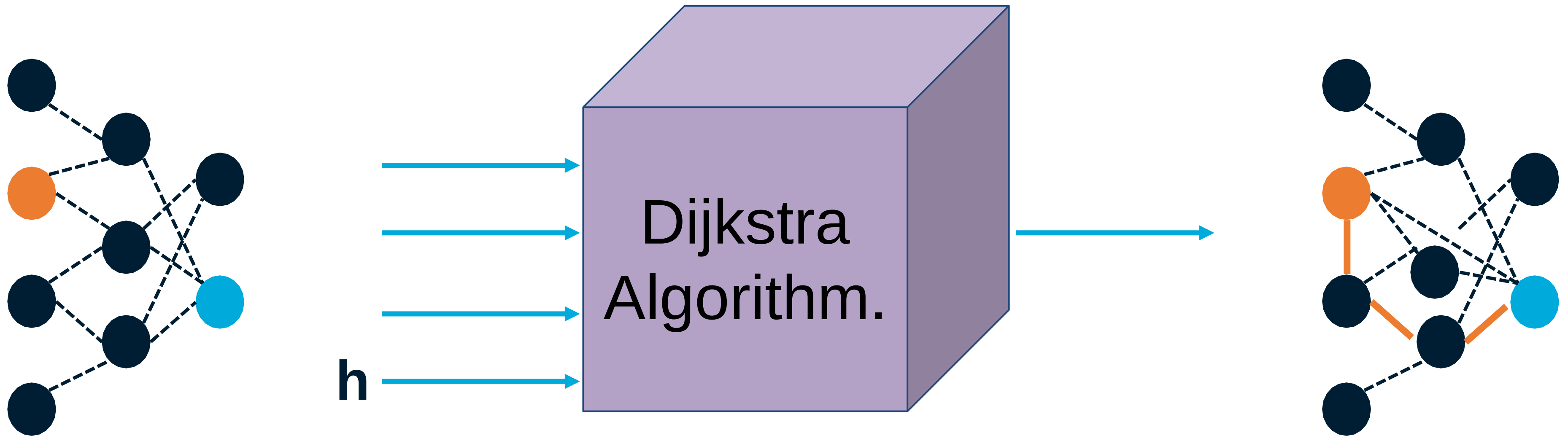
Streets of Medellín, Origin and Destination

a path that further reduces only the distance

https://github.com/RafaelUrbina/st245-002

UNIVERSIDAD EAFIT®

Solution Algorithm

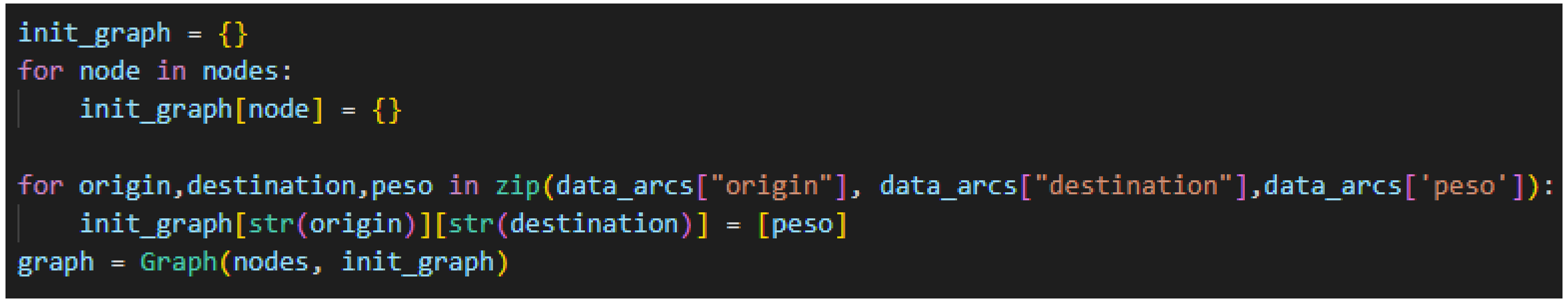


Streets of Medellín, Origin and Destination

a path that reduces risk regardless of distance

The graph implementation

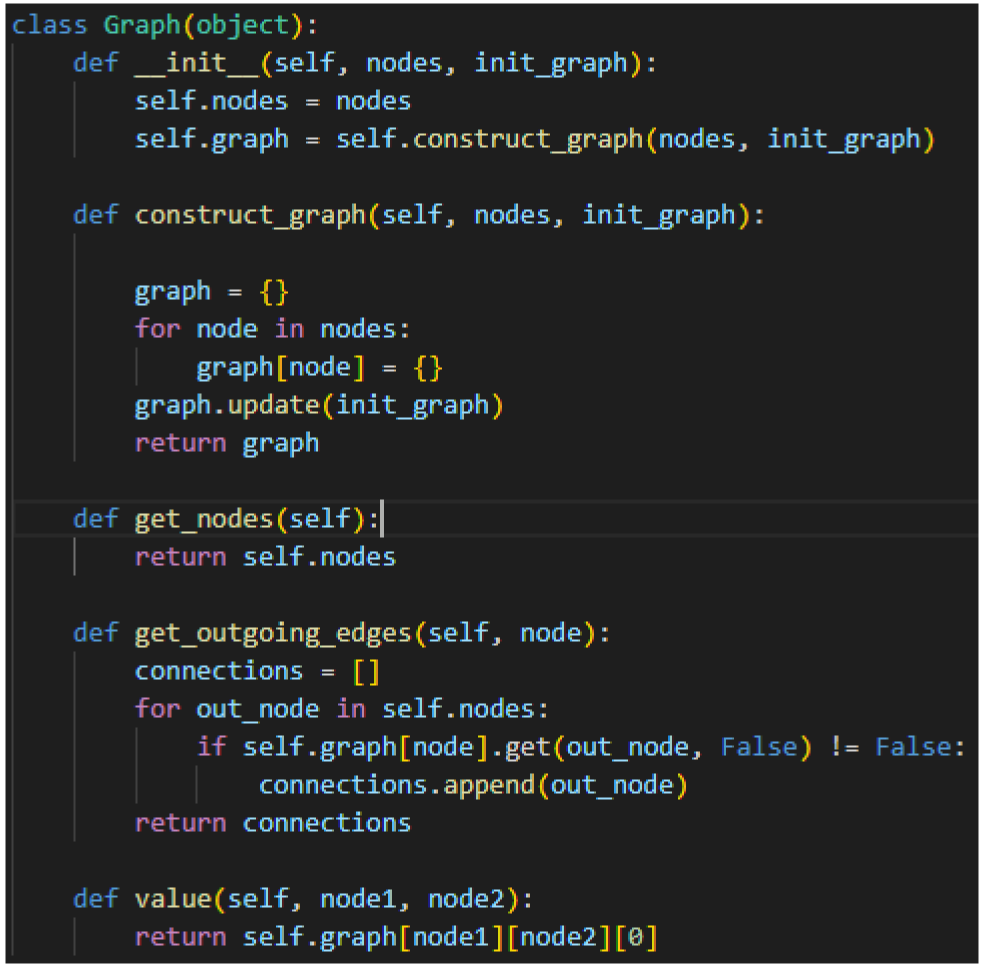
Our data structure is based on an Adjacency list implemented with dictionaries.



```
init_graph = {}
for node in nodes:
    init_graph[node] = {}

for origin,destination,peso in zip(data_arcs["origin"], data_arcs["destination"],data_arcs['peso']):
    init_graph[str(origin)][str(destination)] = [peso]
graph = Graph(nodes, init_graph)
```

The graph implementation

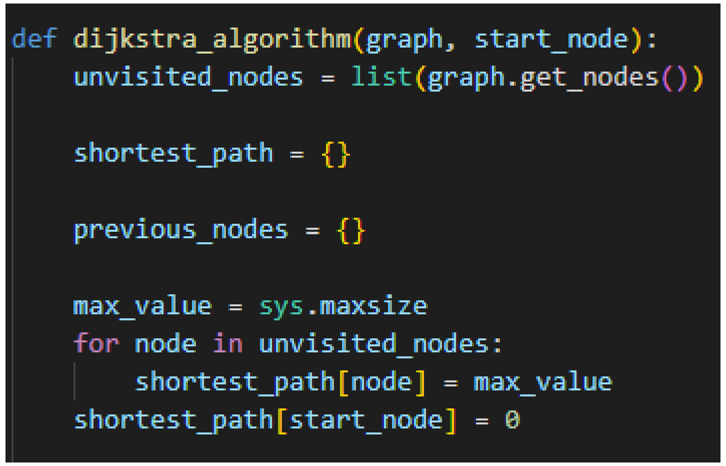
A class Graph is used since this allow us to implement the creation of the graph easily if we want to use it on another script



```python
class Graph(object):
    def __init__(self, nodes, init_graph):
        self.nodes = nodes
        self.graph = self.construct_graph(nodes, init_graph)

    def construct_graph(self, nodes, init_graph):

        graph = {}
        for node in nodes:
            graph[node] = {}
        graph.update(init_graph)
        return graph

    def get_nodes(self):
        return self.nodes

    def get_outgoing_edges(self, node):
        connections = []
        for out_node in self.nodes:
            if self.graph[node].get(out_node, False) != False:
                connections.append(out_node)
        return connections

    def value(self, node1, node2):
        return self.graph[node1][node2][0]
```

The algorithm (Dijkstra)

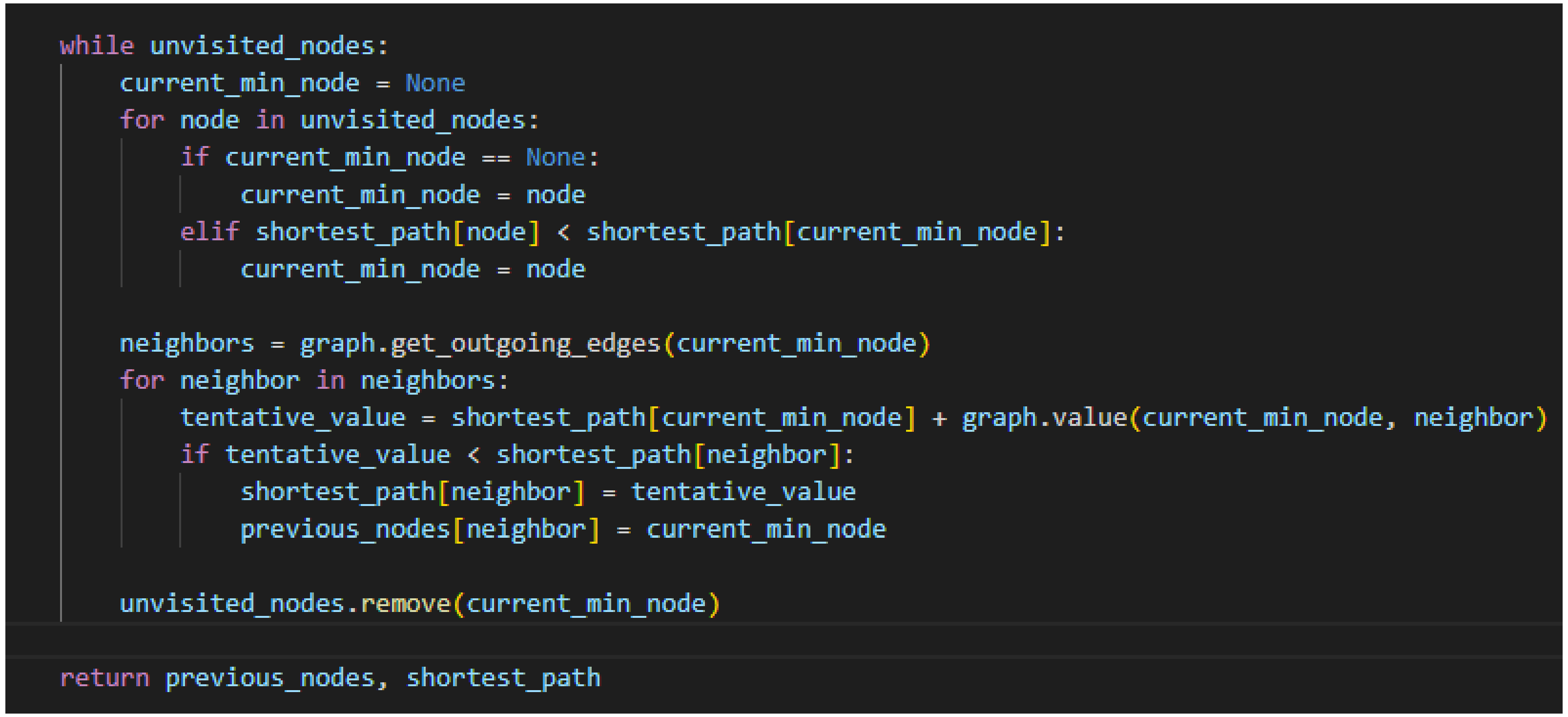
Dijkstra setting all nodes value close to infinity



```python
def dijkstra_algorithm(graph, start_node):
    unvisited_nodes = list(graph.get_nodes())

    shortest_path = {}

    previous_nodes = {}

    max_value = sys.maxsize
    for node in unvisited_nodes:
        shortest_path[node] = max_value
    shortest_path[start_node] = 0
```

The algorithm (Dijkstra)

Dijkstra finding the shortest path



```
while unvisited_nodes:
    current_min_node = None
    for node in unvisited_nodes:
        if current_min_node == None:
            current_min_node = node
        elif shortest_path[node] < shortest_path[current_min_node]:
            current_min_node = node

    neighbors = graph.get_outgoing_edges(current_min_node)
    for neighbor in neighbors:
        tentative_value = shortest_path[current_min_node] + graph.value(current_min_node, neighbor)
        if tentative_value < shortest_path[neighbor]:
            shortest_path[neighbor] = tentative_value
            previous_nodes[neighbor] = current_min_node

    unvisited_nodes.remove(current_min_node)

return previous_nodes, shortest_path
```

Explanation of the algorithm

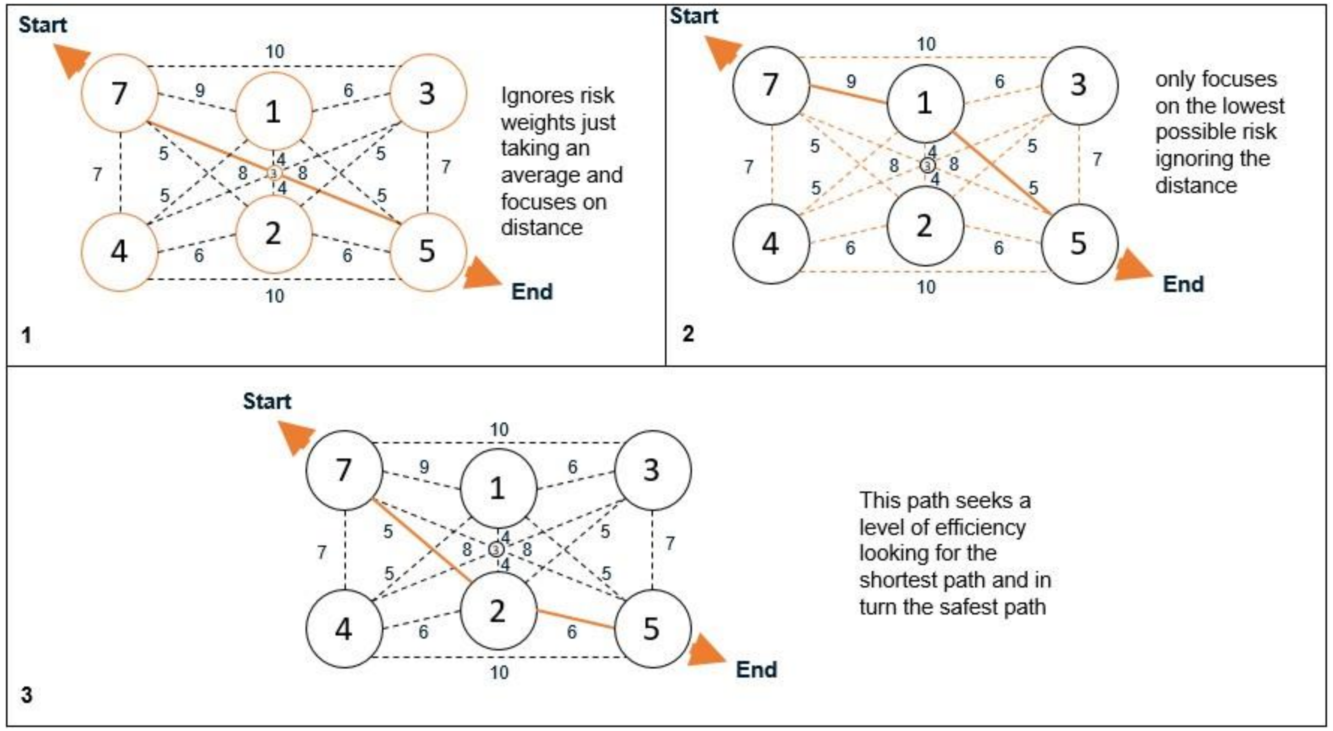

Start
10
9
6
7
1
3
5
4
8
2
End
Ignores risk weights just taking an average and focuses on distance
only focuses on the lowest possible risk ignoring the distance
This path seeks a level of efficiency looking for the shortest path and in turn the safest path

Figure 1: https://www.proclamadelcauca.com/dejame-caminar-por-la-calle-tranquila/

Dijkstra algorithm by three paths, One that takes both distance and risk[1], one that further reduces distance[2], one that gives the safest path[3].

Complexity of the algorithm

	Time complexity	Complexity of memory
Dijkstra Algorithm	$O(V^2)$ $O((V+E)\log V)$	O(V)

Time and memory complexity of the Dijkstra. where V is the number of vertices or nodes and E is the number of edges in the graph.

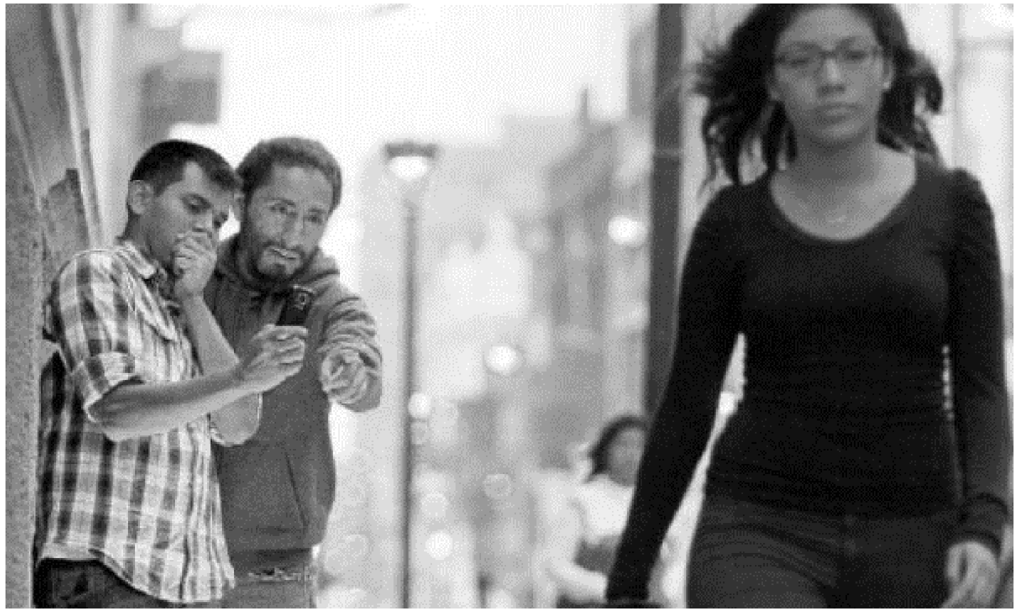


Figure 2: http://www.mujereslibresdeviolencia.usmp.edu.pe/6-cosas-que-debes-hacer-si-eres-victima-de-acoso-sexual-callejero/

First path minimizing d = Length

Origin	Destination	Distance (meters)	Risk of harassment (between 0 and 1)
EAFIT University	National University	7966.87	60.14

Distance and risk of harassment for the path that minimizes d = Length Execution time of 163.56 seconds.

Second path minimizing d = Both

Origin	Destination	Distance (meters)	Risk of harassment (between 0 and 1)
EAFIT University	National University	9671.59	35.10

Distance and risk of harassment for the path that minimizes d = Both Execution time of 164.24 seconds.

Third path minimizing d = Risk

Origin	Destination	Distance (meters)	Risk of harassment (between 0 and 1)
EAFIT University	National University	9680.52	35.10

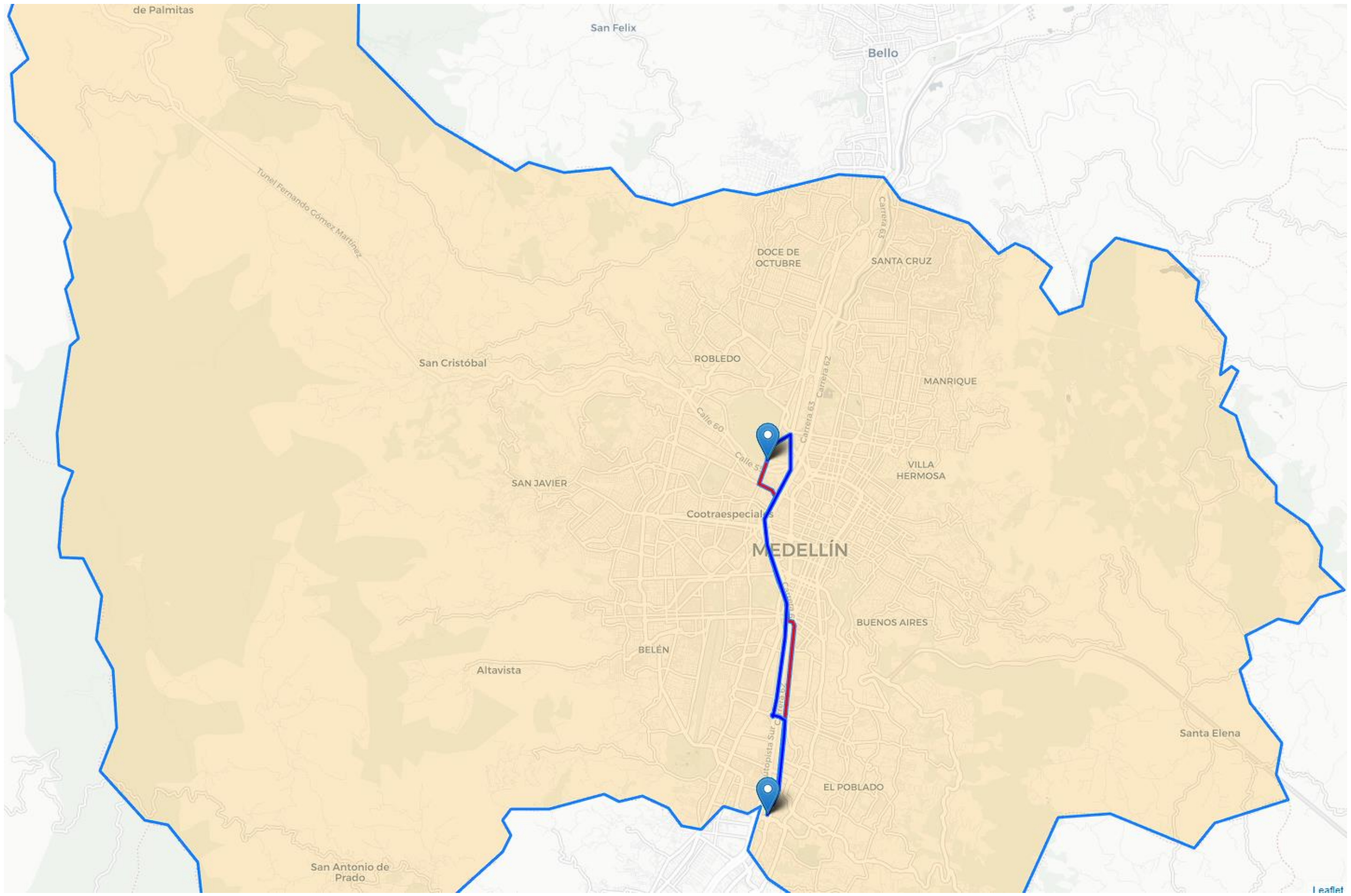
Distance and risk of harassment for the path that minimizes d = Risk Execution time of 166.48 seconds.

Other route d = all

Origin	Destination	Distance (meters)	Risk of harassment (between 0 and 1)
Las Palmas	La America	5603.60	67.66
Las Palmas	La America	6309.87	61.29
Las Palmas	La America	6365.76	61.28

Distance and risk of harassment for the path that minimizes d = all Execution time of 162 seconds.

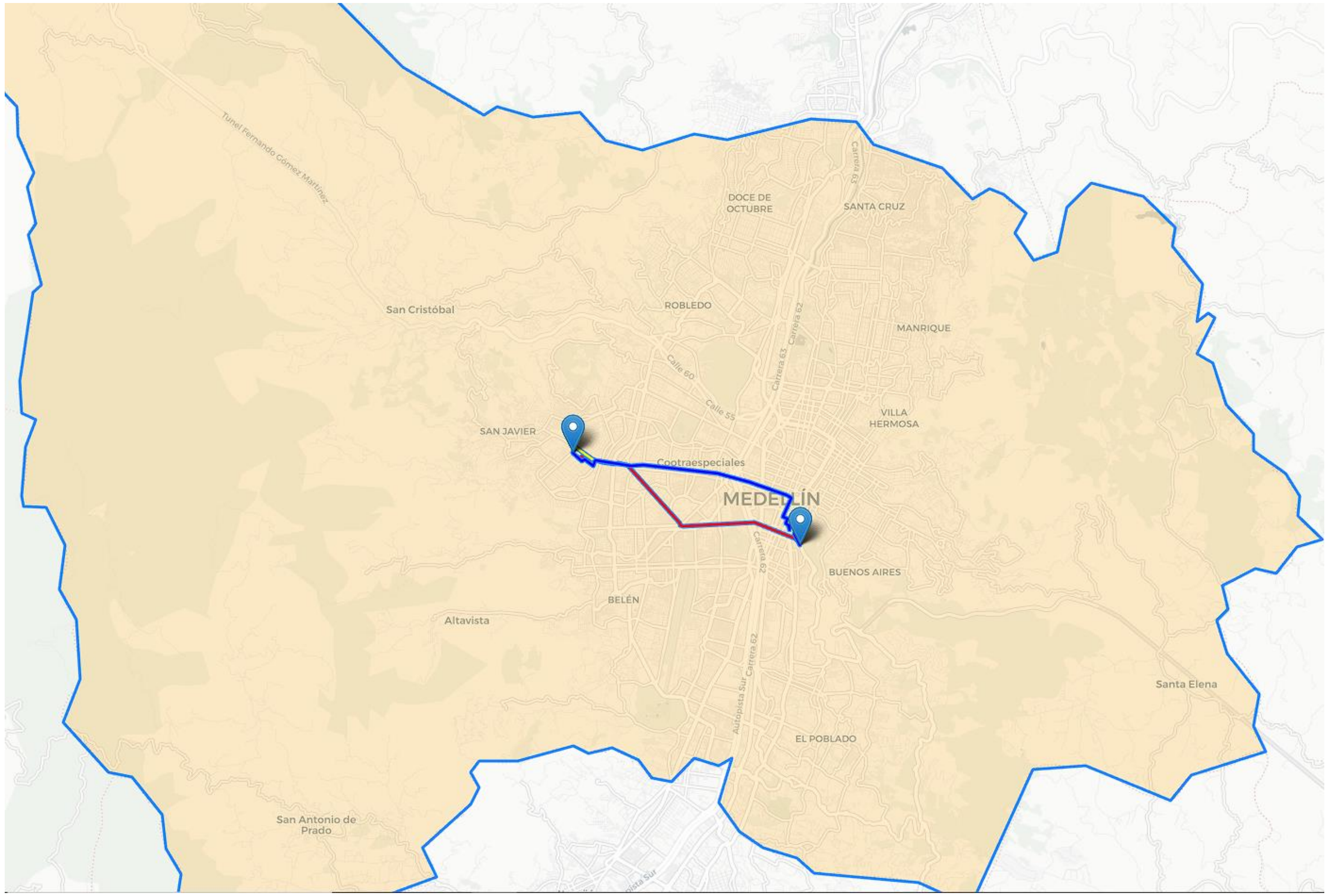
Visual comparison of the three paths (Eafit- Unal)



🟥 Shortest 🟦 Safest 🟨 Combination

UNIVERSIDAD EAFIT®

Visual comparison of the three paths (Las Palmas – La America)



🟥 Shortest 🟦 Safest 🟨 Combination

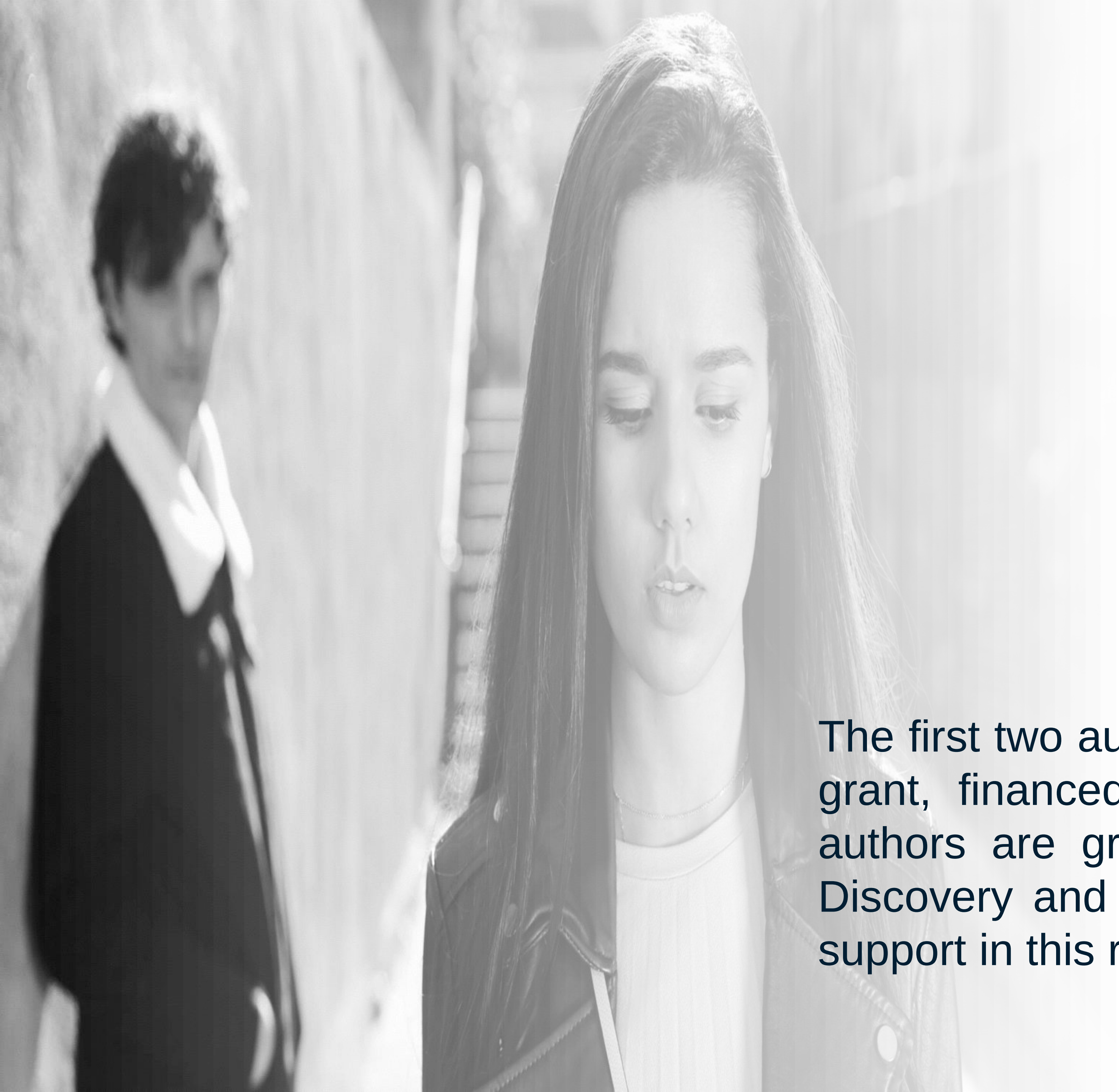
Future work directions

Databases
- Other variables
- Other algorithms

Project 1
- Web application

Software Engineering
- Mobile application

Project 2
- Include ML or VR



THANK YOU!

With the support of

The first two authors were supported by the Sapiencia grant, financed by the municipality of Medellín. All authors are grateful to the Vice Rector's Office for Discovery and Creation, Universidad EAFIT, for their support in this research.