# **TÍTULO (UNA BREVE DESCRIPCIÓN DEL PROYECTO, DE ENTRE 8 Y 12 PALABRAS)**

|  |  |  |  |
| --- | --- | --- | --- |
| Alverth Antonio de la Barrera Moreno  Universidad Eafit  País  aalab@eafit.edu.co | Pedro Cárdenas Restrepo  Universidad Eafit  País  pcardenasr@eafit.edu.co | Sara María Castrillón Ríos Universidad Eafit Colombia smcastril1@eafit.edu.co | Mauricio Toro  Universidad Eafit  Colombia  mtorobe@eafit.edu.co |

# **RESUMEN**

Un problema que afecta a Medellín es el acoso callejero debido al género hacia las mujeres, es importante darle solución a esta situación para reducir el número de víctimas de este tipo de violencia y generar unas mejores condiciones para que la mujer pueda desenvolverse de una manera más tranquila y segura y la sociedad; Pues Al pensar en esta problemática surgen varias incógnitas que nos llevarán a darle solución al problema:

* ¿Como serán recogidos e interpretados los datos, que alimentarán nuestro estudio y posible solución?
* ¿Cómo aplicar la tecnología para la creación de rutas cortas, pero a su vez seguras para ser utilizadas por las mujeres de la ciudad?

## **Palabras clave**

|  |
| --- |
| Camino más corto restringido, acoso sexual callejero,  identificación de rutas seguras, prevención del crimen. |

# **INTRODUCCIÓN**

En el siguiente trabajo se pensó en la necesidad de buscar una alternativa para reducir el acoso callejero en la ciudad de Medellín. Como estudiantes de ingeniería de sistemas, plantearemos una solución tecnológica donde se aplique un algoritmo capaz de arrojar el camino más corto, sin superar un riesgo promedio ponderado de acoso r y otro algoritmo que indique el camino con menor riesgo promedio ponderado de acoso sin superar una distancia d; según el mapa de la ciudad y los datos estadísticos de loa misma.

Es importante la implementación de ambos algoritmos porque nos permitirá realizar una comparación donde podamos identificar cuál de los dos es más eficiente dependiendo de la necesidad del usuario.

Para darle solución a este problema hemos decido hacer uso de una variación al algoritmo de Dijkstra ya que su complejidad O(v\*\*2) crece de manera moderada para cualquier número “v” de vértices, ed por otro lado, es un algoritmo ampliamente utilizado para dar solución a este tipo de problemas. Además de ser fácil de entender y manipular para manejarlo a la hora de establecer las restricciones; para esto hemos planteado etiquetar cada arista con un doble peso, una haciendo referencia a la distancia “d” recorrida en metros y otra al riesgo medido estadísticamente de acoso callejero. De manera nuestro algoritmo hará un balance constante entre ambas ponderaciones y nos devolverá la ruta más corta en metros sin sobrepasar un riesgo promedio ponderado de acoso r.

**1.3 Estructura del artículo**

A continuación, en la Sección 2, presentamos trabajos relacionados con el problema. Posteriormente, en la Sección 3, presentamos los conjuntos de datos y los métodos utilizados en esta investigación. En la Sección 4, presentamos el diseño del algoritmo. Después, en la Sección 5, presentamos los resultados. Finalmente, en la Sección 6, discutimos los resultados y proponemos algunas direcciones de trabajo futuro.

**2. TRABAJOS RELACIONADOS**

A continuación, explicamos cuatro trabajos relacionados con la búsqueda de caminos para prevenir el acoso sexual callejero y la delincuencia en general.

* **“Riesgo Percibido de Acoso Callejero y Opciones Educativas de las Mujeres”** Este documento examina el impacto del riesgo percibido por acoso callejero en los derechos humanos de las mujeres. Además, reúne un conjunto de datos único que combina información de 4.000 estudiantes en la Universidad de Delhi, a partir de esto, se implementa un algoritmo desarrollado en Google que usa mapas y datos de seguridad de aplicaciones móviles de fuentes múltiples. Finalmente, utilizando métodos econométricos estructurales, estiman que las mujeres están dispuestas a sacrificar calidad en educación por obtener una ruta con un promedio que se percibe más segura. Borker, G., 2018, Safety First: Perceived Risk of Street Harassment and Educational Choices of Women, JOB MARKET PAPER.
* **“La construcción social de los miedos de las mujeres asociados al uso del espacio público del centro de Medellín”** Este trabajo de grado realizado por una estudiante de la universidad EAFIT, es un estudio que mediante la implementación de modelos estadísticos reconoce las principales zonas del centro de Medellín que representan riesgos desde la perspectiva de mujeres que lo habitan o transitan cotidianamente. A partir de lo anterior, la autora propone ciertas estrategias que pueden ayudar a darle solución a esta problemática. Giraldo, J., 2018. La construcción social de los miedos de las mujeres asociados al uso del espacio público del centro de Medellín, Medellín.
* **“Modelado basado en grafos y el algoritmo de Dijkstra para la búsqueda de rutas de vehículos en autopistas”** Este artículo se basa en la construcción de rutas vehiculares que pueden cambiar o recalcular su trayectoria con base a cambios en las condiciones de la carretera. La implementación de esta estrategia es abordada con la implementación de grafos y el algoritmo de Dijkstra, la cual es muy utilizada en la búsqueda de los caminos más cortos. La conclusión es un modelo matemático que mediante la utilización de ciertos factores es capaz de resolver dicho problema. Kirono, S. Arifianto, M. Putra, R. y Musoleh, A. (2018). Graph-based Modeling and Dijkstra Algorithm for Searching Vehicle Routes on Highways. International Journal of Mechanical Engineering and Technology (IJMET)
* **“Comparación de la función de ‘Evitar rutas peligrosas’ de Waze en las ciudades de Rio de Janeiro e Israel”** Este artículo presenta un estudio comparativo de la función Waze Avoid Areas (WADA) en Jerusalén, Río de Janeiro y EE. UU, abordando el cómo funciona la característica, y cuáles son sus defectos. El estudio arroja que la implementación del mismo algoritmo en diferentes ciudades y contextos no es eficaz y resulta en efectos negativos. Carraro, V. (2019). Grounding the digital: a comparison of Waze ‘avoid dangerous areas’ feature in Jerusalem, Rio de Janeiro and the US. GeoJournal.

## **3. MATERIALES Y MÉTODOS**

En esta sección, explicamos cómo se recogieron y procesaron los datos y, después, diferentes alternativas de algoritmos del camino más corto restringido para abordar el acoso sexual callejero.

## **3.1 Recogida y tratamiento de datos**

El mapa de Medellín se obtuvo de Open Street Maps (OSM) [[1]](#footnote-1)y se descargó utilizando la API[[2]](#footnote-2) OSMnx de Python. La (i) longitud de cada segmento, en metros; (2) la indicación de si el segmento es de un solo sentido o no, y (3) las representaciones binarias conocidas de las geometrías se obtuvieron de los metadatos proporcionados por OSM.

Para este proyecto, se calculó la combinación lineal (CL) que captura la máxima varianza entre (i) la fracción de hogares que se sienten inseguros y (ii) la fracción de hogares con ingresos inferiores a un salario mínimo. Estos datos se obtuvieron de la encuesta de calidad de vida de Medellín, de 2017. La CL se normalizó, utilizando el máximo y el mínimo, para obtener valores entre 0 y 1. La CL se obtuvo mediante el análisis de componentes principales. El riesgo de acoso se define como uno menos la CL normalizada. La Figura 1 presenta el riesgo de acoso calculado. El mapa está disponible en GitHub[[3]](#footnote-3).

**Figura 1.** Riesgo de acoso sexual calculado como una combinación lineal de la fracción de hogares que se sienten inseguros y la fracción de hogares con ingresos inferiores a un salario mínimo, obtenida de la Encuesta de Calidad de Vida de Medellín, de 2017.

## **3.2 Alternativas de camino más corto con restricciones**

## A continuación, presentamos diferentes algoritmos utilizados para el camino más corto restringido.

**3.2.1 Algoritmo de Búsqueda en Anchura (BFS)**

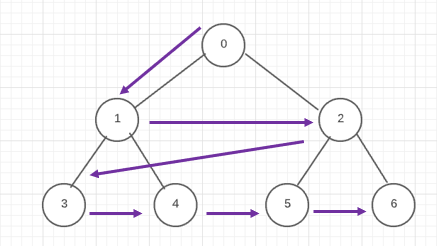
El BFS busca hacer el recorrido por niveles, es decir que en el BFS trataremos el grafo como un árbol. Tomando como raíz un nodo inicial.

Este recorrido implementa una cola que permite almacenar temporalmente todos los nodos de un nivel, para ser procesados antes de pasar al siguiente nivel hasta que la cola esté vacía, De esta forma:

Inicialmente se debe declarar la estructura de cola, luego agregar el nodo raíz (esto se hace porque se necesita tener al menos un elemento en la cola, dado que la condición de salida es que esté vacía); marcamos el nodo raíz como visitado.

Cada vez que un nodo es visitado, debe ser desencolado, para luego agregar a la cola todos los nodos del siguiente nivel y los marcamos como visitados antes de comenzar el ciclo de nuevo, en el que procesaremos estos nuevos nodos que hemos agregado a la cola.

Si encontramos el elemento buscado, la función BFS retornará al “main”.



La complejidad temporal es O(V) donde V es el número de nodos, debe atravesar todos los nodos.

La completitud del espacio también es O(V), ya que en el peor de los casos debe mantener todos los vértices en la cola.

**3.2.2 Algoritmo de Búsqueda en Profundidad (DFS):**

En este caso se busca recorrer el grafo desde la raíz hasta los nodos extremos u hojas por cada una de las ramas; los niveles de cada nodo no serán importantes.

La forma más intuitiva de hacer este algoritmo es a través de la recursividad, de lo contrario tendríamos que usar en lugar de una cola una pila.

Para llevar a cabo el algoritmo, deben ser pasados por parámetro el nodo a buscar y el nodo actual (El nodo que está siendo visitado en cada ambiente de recursión), que en la primera llamada será el nodo raíz.

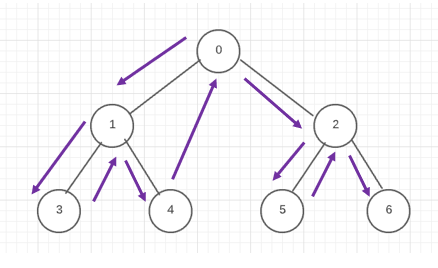
En cada llamada recursiva marcaremos el nodo actual como visitado y luego verificamos si es el nodo buscado para salir de la recursión, este será nuestro caso base. De no ser el nodo requerido, se hace la llamada recursiva con todos los nodos hijos del nodo actual, pero en este caso, a diferencia del recorrido BFS, no se visitarán todos los hijos de forma consecutiva, sino que el algoritmo recorrerá en profundidad hasta llegar a un nodo extremo o nodo hoja, antes de retornar al ambiente de recursión en donde se encuentran los otros nodos hijos.

El orden en que se eligen las ramas en un recorrido DFS está determinado por el tipo de recorrido de procesamiento de árbol que se haya elegido, estos pueden ser:

**Pre-orden:** Se procesa primero la raíz, luego la rama izquierda y luego las ramas siguientes hasta llegar a la que se encuentra más a la derecha.

**Post-orden:** Se procesa el árbol desde las ramas izquierdas hasta la que se encuentra más a la derecha. Finalmente se procesa el nodo raíz

**Simétrico o In-orden:** Se procesa la rama de la izquierda, luego el nodo raíz y luego la rama derecha.



La complejidad temporal es nuevamente O(|V|), necesita atravesar todos los nodos.

Complejidad del espacio: depende de la implementación, una implementación recursiva puede tener una complejidad de espacio O(h) [el peor de los casos], donde h es la profundidad máxima de su árbol.

Usar una solución iterativa con una pila es en realidad lo mismo que BFS, solo usar una pila en lugar de una cola, por lo que obtienes tanto O(|V|) como complejidad de tiempo y espacio.

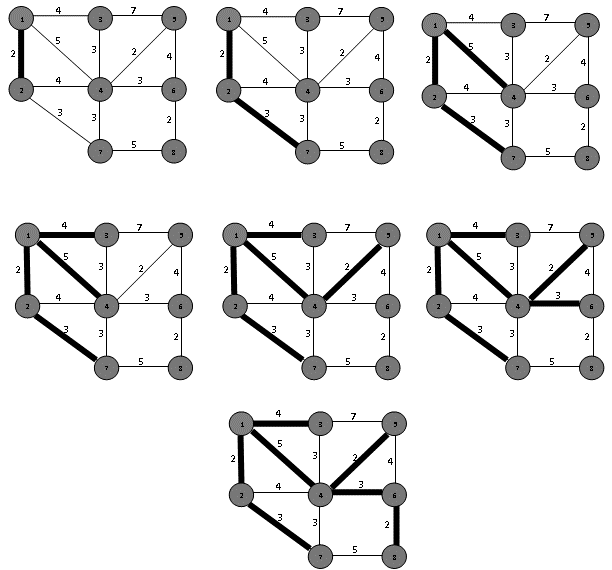
**Tips:**

* La búsqueda en anchura se recomienda cuando lo que se necesita es buscar el camino más corto en grafos no ponderados.
* La búsqueda en profundidad se puede utilizar para detectar ciclos en un grafo, determinar si un grafo es conexo o no y cuántas componentes conexas tiene, determinar puntos de articulación y biconexión de grafos, entre otras cosas.
* Cuando no tiene importancia el orden en que visitemos los nodos y aristas del grafo, se puede usar cualquiera de los dos algoritmos.

**3.2.3 Dijkstra’s shortest path algorithm:**

El algoritmo de Dijkstra es muy similar al algoritmo de Prim para el árbol de expansión mínimo; donde se genera un SPT (árbol de ruta más corta) con una fuente determinada como raíz. Mantenemos dos conjuntos, un conjunto contiene vértices incluidos en el árbol de ruta más corta, el otro conjunto incluye vértices que aún no están incluidos en el árbol de ruta más corta. En cada paso del algoritmo, encontramos un vértice que está en el conjunto de aún no incluido y tiene una distancia mínima de la fuente.

Los pasos utilizados en el algoritmo de Dijkstra para encontrar el camino más corto desde un único vértice fuente a todos los demás vértices del grafo son:

1. Cree un conjunto, inicialmente vacio, spt\_set (conjunto de árboles de ruta más corta) que realice un seguimiento de los vértices incluidos en el árbol de ruta más corta, es decir, cuya distancia mínima desde la fuente se calcula y finaliza.

2. Asigne un valor de distancia a todos los vértices en el gráfico de entrada. Inicialice todos los valores de distancia como INFINITOS. Asigne el valor de distancia como 0 para el vértice de origen para que se elija primero.

3. Mientras que spt\_set no incluye todos los vértices:

a. Elija un vértice “u” que no esté en spt\_set y tenga un valor de distancia mínimo.

b) Incluir “u” en spt\_set.

c. Actualizar el valor de distancia de todos los vértices adyacentes de “u”. Para actualizar los valores de distancia; itere a través de todos los vértices adyacentes. Para cada vértice adyacente “v”, si la suma del valor de la distancia de “u” (desde la fuente) y el peso del borde u-v es menor que el valor de la distancia de v, actualice el valor de la distancia de v.

Teorema: El algoritmo de Dijkstra realiza O(V\*\*2) operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo y no dirigido con n vértices, es decir:

Tiempo de ejecución = O(|A|.𝑻\_𝒅𝒌+|v|.𝑻\_𝒅𝒎)

|A|: Número de aristas

𝑻\_𝒅𝒌: Complejidad de disminuir clave

|V|: Numero de vértices

𝑻\_𝒅𝒎: Complejidad de extraer mínimo

**3.2.4 Algoritmo PathFinding A\*:**

Pathfinding A\* plantea que:

Movimiento = Esfuerzo + Heurística

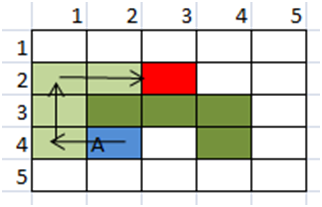
EL A\* funcione de manera muy similar a Dijkstra, la única diferencia está en cómo sopesas (u ordenas) a los candidatos. Con Dijkstra es:

Cost\_of\_Edge + Cost\_of\_previous\_Node

Con A \* es:

Cost\_of\_Edge + Cost\_of\_previous\_Node + Estimated\_Cost\_to\_reach\_Target\_from(Node)

Donde Estimated\_Cost\_to\_reach\_Target\_from comúnmente se llama una función heurística. Esta es una función que intentará estimar el costo para alcanzar el nodo objetivo. Una buena función heurística logrará que se tengan que visitar menos nodos para encontrar el objetivo. Si bien el algoritmo de Dijkstra se expandiría a todos los lados, A\* buscará (gracias a la heurística) en la dirección del objetivo.



**Complejidad:** El espacio requerido por A\* para ser ejecutado es su mayor problema. Dado que tiene que almacenar todos los posibles siguientes nodos de cada estado, la cantidad de memoria que requerirá será exponencial con respecto al tamaño del problema, Para solucionar este problema, se han propuesto diversas variaciones de este algoritmo, como pueden ser RTA\*, IDA\* o SMA\*.

Los autores agradecen al profesor Juan Carlos Duque, de la Universidad EAFIT, por facilitar los datos de la Encuesta de Calidad de Vida de Medellín, de 2017, procesados en un *Shapefile*.

## **4. DISEÑO E IMPLEMENTACIÓN DEL ALGORITMO**

## A continuación, explicamos las estructuras de datos y los algoritmos utilizados en este trabajo. Las implementaciones de las estructuras de datos y los algoritmos están disponibles en Github[[4]](#footnote-4).

## **4.1 Estructuras de datos**

## Para la representación del grafo utilizamos dos listas de adyacencia. Una de ellas almacenará los vértices, y la otra las aristas con sus respectivos atributos, donde incluimos “risk” y ”lenght”; Decidimos lista en lugar de matrices para disminuir un poco la complejidad en caso de que no se trate de un grafo completo, es decir, nuestro peor caso, en donde la lista de adyacencia se vería como una matriz, por otro lado fueron necesarias dos listas ya que cada vértice tiene demasiados atributos asociados y al separarlos se hace un poco más intuitiva la construcción del grafo.

## La estructura de los datos se presenta en la Figura 2.

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente**Figura 2:** Un ejemplo de mapa de calles se presenta en (a) y su representación como lista de adyacencia en (b).

**4.2 Algoritmos**

En este trabajo, proponemos algoritmos para el problema del camino más corto restringido. El primer algoritmo calcula el camino más corto sin superar un riesgo medio ponderado de acoso *r*. El segundo algoritmo calcula el camino con el menor riesgo medio ponderado de acoso sin superar una distancia *d*.

**4.2.1 Primer algoritmo**

A la hora de encontrar el camino más corto restringido con ayuda del algoritmo de Dijkstra, utilizamos una matriz inicialmente compuesta de infinitos, de n\*n, donde n representa el número de vértices del grafo; Allí estaremos analizando uno por uno cada vértice hasta llegar el vértice destino, en el momento en el que el vértice actual sea igual destino el programa se termina, lo que nos ayuda a disminuir la complejidad en algunos casos. La idea es analizar los sucesores del vértice actual y elegir el que tenga menor distancia, es por eso que inicialmente la tabla está compuesta por infinitos, porque a medida de que encontramos un valor menor al ya registrado en la tabla, debemos actualizarla.

El algoritmo se ejemplifica en la Figura 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Paso 1 | Paso 2 | … | Paso n |
| Vértice 1 | Inf | Inf | Inf | Inf |
| Vértice 2 | Inf | Inf | Inf | Inf |
| … | Inf | Inf | Inf | Inf |
| Vértice n | Inf | Inf | Inf | Inf |

**Figura 3**

# **REFERENCIAS**

* Bibliadelprogramador.com. 2022. *Algoritmos de Búsqueda en Anchura (BFS) y Búsqueda en Profundidad (DFS)*. [online] Available at: <https://www.bibliadelprogramador.com/2014/04/algoritmos-de-busqueda-en-anchura-bfs-y.html> [Accessed 19 February 2022].
* La tecnología cambia la vida futura – Gobetech.com. 2022. *¿Cómo se puede encontrar la complejidad del tiempo para el recorrido DFS?*. [online] Available at: <https://tech.gobetech.com/22258/como-se-puede-encontrar-la-complejidad-del-tiempo-para-el-recorrido-dfs.html> [Accessed 19 February 2022].
* GeeksforGeeks. 2022. Dijsktra's algorithm. [online] Available at: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/> [Accessed 19 February 2022].
* Cristalab. 2022. Algoritmo A\* para encontrar el camino más corto en IA. [online] Available at: <https://www.cristalab.com/tutoriales/algoritmo-a-para-encontrar-el-camino-mas-corto-en-ia-c91591l/> [Accessed 19 February 2022].

1. <https://www.openstreetmap.org/> [↑](#footnote-ref-1)
2. https://osmnx.readthedocs.io/ [↑](#footnote-ref-2)
3. [https://github.com/mauriciotoro/ST0245Eafit/tree/master/  
   proyecto/Datasets/](https://github.com/mauriciotoro/ST0245Eafit/tree/master/proyecto/Datasets)  [↑](#footnote-ref-3)
4. http://www.github.com/ ????????? /.../proyecto/ [↑](#footnote-ref-4)