



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

INVESTIGACIÓN Y DESARROLLO DE MÉTODOS DE EVALUACIÓN Y OPTIMIZACIÓN DE RUTAS DE EVACUACIÓN SEGURAS EN UNA APLICACIÓN MÓVIL PARA ALERTAS DE CATÁSTROFES - ALERTIFY

Paulina Candia Poblete

paulina.candia@sansano.usm.cl

Gabriel Jara
Profesor Guía

Resumen: Este trabajo aborda la necesidad de optimizar las rutas de evacuación ante desastres naturales, como incendios forestales, terremotos y tsunamis, en regiones vulnerables. La propuesta consiste en desarrollar un sistema integrado en una aplicación móvil que ofrece rutas seguras en tiempo real, adaptadas según el tipo de emergencia. Para lograrlo, se implementan algoritmos de optimización, como A*, que permiten calcular las rutas más seguras y eficientes, considerando factores como la accesibilidad, la peligrosidad del entorno y la evolución del desastre. Se valida el sistema mediante simulaciones y pruebas en escenarios controlados, evaluando variables como el tiempo de evacuación, la distancia recorrida y la exposición a zonas de riesgo. Con esta herramienta, se busca mejorar la gestión de emergencias, fortalecer la resiliencia comunitaria y reducir las pérdidas humanas y materiales durante catástrofes naturales.

Palabras Clave: Evacuación, desastres naturales, rutas seguras, optimización, algoritmo A*, datos geoespaciales.



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: INVESTIGACIÓN Y DESARROLLO DE MÉTODOS DE EVALUACIÓN Y OPTIMIZACIÓN DE RUTAS DE EVACUACIÓN SEGURAS EN UNA APLICACIÓN MÓVIL PARA ALERTAS DE CATÁSTROFES - ALERTIFY

Nombre del candidato(a): Paulina Candia Poblete

Carrera / Grado: Ingeniería en Informática

Campus: Sede Viña del Mar **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Gabriel Alberto Jara Bulnes, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha:

Firma:

17-12-2025

Estudiante o Candidato(a):

Fecha: 15/12/2025

Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.



Índice

1	Introducción	3
1.1	Contexto y antecedentes	3
1.2	Definición del problema.....	3
1.3	Descripción de solución	4
1.4	Objetivos	5
1.4.1	Objetivo General.....	5
1.4.2	Objetivos Específicos	5
1.5	Justificación del proyecto	5
1.6	Metodología	7
1.7	Organización del informe en capítulos.....	7
2	Marco Teórico.....	8
2.1	Descripción de algoritmos de optimización de rutas.....	9
2.1.1	Dijkstra.....	9
2.1.2	A* (A Estrella).....	11
2.1.3	JPS	14
2.1.4	Otros Algoritmos.....	16
2.2	Tecnologías y <i>frameworks</i> utilizados	17
3	Diseño e Implementación.....	19
3.1	Método de optimización de ruta	19
3.2	Desarrollo del prototipo.....	21
3.3	Parámetros de penalización.....	22
4	Validación	23
4.1	Metodología de validación	23
4.2	Escenario simulado	23
4.3	Librerías utilizadas	24
4.4	Simulaciones realizadas	24
5	Evaluación.....	32
5.1	Métricas de evaluación	32
5.2	Resultados comparativos	33
5.3	Análisis e interpretación	33
5.4	Conclusiones del proceso de evaluación.....	34
5.5	Propuesta de mejora	34
6	Conclusiones	36
7	Agradecimientos.....	37
8	Referencias	37

1 Introducción

1.1 Contexto y antecedentes

Los desastres naturales, como incendios forestales, terremotos y tsunamis, representan una amenaza creciente para la seguridad de las comunidades, especialmente en regiones vulnerables como América Latina y el Caribe. En las últimas décadas, la frecuencia y severidad de estos eventos han aumentado debido al cambio climático y a la rápida urbanización no planificada, lo que ha resultado en un incremento considerable en las pérdidas humanas, económicas y ambientales. En Chile, la recurrencia de incendios forestales, sumada a la alta actividad sísmica y la amenaza constante de tsunamis, ha expuesto la necesidad urgente de mejorar la gestión del riesgo y la planificación de evacuaciones.

Los desastres naturales, como incendios forestales, terremotos y tsunamis, representan un desafío creciente para las comunidades, especialmente en regiones vulnerables como Chile y América Latina. Su frecuencia e intensidad, han aumentado considerablemente, incrementando la exposición de la población al riesgo, dificultando la respuesta efectiva durante una catástrofe y aumentando las pérdidas humanas y materiales.

Este trabajo de investigación se enfoca en el desarrollo de un prototipo de sistema de optimización de rutas de evacuación, integrado en una aplicación móvil para alertas de emergencias, que combine información geoespacial, análisis de riesgos y datos en tiempo real para guiar de manera eficiente a la población hacia zonas seguras. Al implementar esta solución, se busca contribuir a la reducción del riesgo y a la mejora de la resiliencia comunitaria frente a desastres naturales, optimizando los recursos disponibles y facilitando la toma de decisiones críticas en situaciones de emergencia.

1.2 Definición del problema.

Frente a este escenario, resulta esencial el desarrollo de herramientas tecnológicas que permitan a las autoridades y a la población reaccionar de manera eficiente ante situaciones de emergencia. Aunque sistemas como el Sistema de Alerta de Emergencias (SAE) han demostrado ser efectivos para la difusión de alertas, estos mecanismos se limitan a notificar sobre la ocurrencia de un evento, sin proporcionar información detallada sobre rutas de evacuación seguras. Las rutas predefinidas no se adaptan a condiciones dinámicas como bloqueos viales, propagación de incendios o inestabilidad del terreno tras un terremoto. Esto genera confusión y aumenta el riesgo durante emergencias.

La capacidad de ofrecer rutas seguras y adaptadas en tiempo real es fundamental para minimizar las pérdidas humanas durante una catástrofe. Las rutas predefinidas, utilizadas tradicionalmente, no contemplan la capacidad de adaptarse a situaciones dinámicas como el bloqueo de carreteras, el avance de un incendio o la inestabilidad del terreno tras un terremoto. La ausencia de datos en tiempo real sobre el estado de las vías y las amenazas presentes complica la planificación y ejecución de evacuaciones, incrementando el riesgo para las comunidades afectadas y dificultando la toma de decisiones durante una emergencia.

El problema central que aborda esta investigación es la falta de herramientas que permitan identificar rutas seguras durante emergencias, donde el riesgo es alto y las personas no saben por dónde escapar ni cómo llegar a lugares seguros. Actualmente, los algoritmos tradicionales de optimización de rutas (como Dijkstra o A* estándar) solo minimizan distancia o tiempo, sin considerar factores críticos de seguridad en contextos de evacuación. En situaciones de emergencia, como incendios forestales, terremotos o tsunamis, una ruta óptima en términos de distancia puede ser altamente peligrosa si atraviesa zonas de riesgo (ej. áreas con propagación de fuego, derrumbes o inundaciones).

1.3 Descripción de solución

Se propone desarrollar un prototipo de sistema de optimización de rutas de evacuación integrado en Alertify, que combine:

- Algoritmos de búsqueda de caminos, modificados para incorporar seguridad.
- Datos geoespaciales y de riesgo en tiempo real (ej. propagación de incendios, bloqueos viales).
- Geolocalización para adaptar rutas según la ubicación del usuario y la evolución del desastre.

El sistema recalculará rutas dinámicamente para evitar zonas de riesgo emergentes, optimizando tiempo de evacuación y reduciendo exposición a peligros.

El prototipo aprovechará la geolocalización para determinar la posición del usuario y calcular rutas considerando factores como accesibilidad, propagación del desastre y congestión vial. Además, incorporará la capacidad de ajustar rutas dinámicamente en respuesta a cambios en las condiciones, evitando zonas de riesgo emergentes.

Este enfoque dinámico no solo optimiza el tiempo de evacuación, sino que también reduce la exposición a peligros, mejorando significativamente la capacidad de respuesta durante emergencias. Con esta solución, se busca contribuir a la reducción del riesgo y a la protección de vidas humanas, facilitando una toma de decisiones informada y efectiva en momentos críticos.

Este trabajo se enfoca en la exploración y adaptación de modelos de optimización de rutas para incorporar parámetros de seguridad. La solución propuesta incluye:

- Modificación de la función de costo en algoritmos de optimización, incorporando penalizaciones por riesgo (ej. cercanía a incendios, bloqueos viales) en la función de costo del algoritmo.
- Priorizar rutas que eviten zonas peligrosas, incluso si esto implica un aumento en la distancia total.
- Utilizar heurísticas de seguridad para dirigir la búsqueda hacia zonas seguras.
- Implementar el algoritmo modificado en un entorno simulado.
- Evaluar su rendimiento mediante métricas como tiempo de evacuación, distancia recorrida y exposición a zonas de riesgo.

1.4 Objetivos

1.4.1 Objetivo General

Este trabajo tiene por objetivo la investigación, evaluación y desarrollo de un prototipo de modelo de optimización de ruta que pueda ser integrada a una aplicación móvil que recibe alertas diferenciadas ante catástrofes, dando prioridad a la seguridad de los usuarios de la aplicación.

1.4.2 Objetivos Específicos

- Investigar y comparar diferentes métodos de optimización de rutas, considerando variables de seguridad, tiempo de evacuación y riesgos inherentes a cada tipo de catástrofe.
- Desarrollar un prototipo que implemente un sistema básico de optimización de rutas en tiempo real utilizando datos de riesgo.
- Evaluar el rendimiento de las rutas generadas por el prototipo utilizando simulaciones y escenarios de emergencia.
- Proponer mejoras y ajustes basados en los resultados de la evaluación, considerando tanto el rendimiento técnico como la usabilidad del sistema.

1.5 Justificación del proyecto

El desarrollo de un sistema de evaluación y optimización de rutas de evacuación seguras en una aplicación móvil representa una solución innovadora a la creciente necesidad de responder eficazmente a desastres naturales. Esta investigación adquiere especial relevancia al abordar un problema crítico para la seguridad de las comunidades, aplicando conocimientos avanzados en algoritmos de búsqueda de camino, sistemas dinámicos y tecnología móvil. Este proyecto no solo aborda un problema crítico para la seguridad de las comunidades, sino que también se fundamenta en la aplicación de conocimientos avanzados en optimización de rutas, sistemas dinámicos y tecnología móvil, generando aportes significativos en términos de relevancia técnica, beneficios tangibles e innovación.

El sistema aborda una necesidad crítica: proporcionar rutas de evacuación seguras y adaptativas en tiempo real. A diferencia de sistemas tradicionales, el enfoque propuesto integra algoritmos avanzados, mejorando la capacidad de respuesta y reduciendo riesgos. Además, contribuye al desarrollo de soluciones tecnológicas innovadoras para la gestión de emergencias.

El proyecto incorpora características únicas que lo diferencian de los sistemas tradicionales de evacuación:

- Adaptación en tiempo real, permitiendo recalcular rutas según condiciones dinámicas, como incendios en propagación o bloqueos viales, representa un avance crucial frente a sistemas estáticos.
- Alertas diferenciadas las cuales generan notificaciones personalizadas según el tipo de desastre permitiendo tomar decisiones más informadas y efectivas.

- El uso de una aplicación móvil accesible y fácil de usar, permitiendo el acceso a herramientas avanzadas, convirtiéndolas en un recurso esencial para comunidades vulnerables.

Además, el proyecto destaca por la implementación de algoritmos avanzados de búsqueda de camino que permiten calcular rutas de evacuación eficientes y adaptables en tiempo real. Estos métodos integran datos dinámicos, como geolocalización y análisis de riesgos, lo que requiere conocimientos sólidos en programación, estructuras de datos y sistemas de información geográfica (SIG). Además, la utilización de modelos heurísticos demuestra la capacidad para optimizar procesos complejos en condiciones de incertidumbre, asegurando soluciones robustas y técnicamente viables para la gestión de emergencias.

Los algoritmos de búsqueda de camino han demostrado ser herramientas fundamentales en la planificación de rutas óptimas en diversos contextos. Sin embargo, su aplicación en escenarios de emergencia presenta desafíos únicos que requieren adaptaciones específicas. Este trabajo explora precisamente esas adaptaciones, modificando la función de costo tradicional para incorporar parámetros de seguridad que consideren factores como la accesibilidad, la peligrosidad del entorno y la evolución dinámica de los desastres. Esta adaptación representa un impacto práctico directo en la capacidad de respuesta ante emergencias basado en los conocimientos prácticos y teóricos que se tienen en el área de optimización.

La investigación en algoritmos de búsqueda de camino para rutas de evacuación es valiosa porque permite desarrollar soluciones tecnológicas que pueden salvar vidas y reducir el impacto de los desastres naturales. Al integrar estos algoritmos con datos geoespaciales y de riesgo en tiempo real, se crea un sistema potencialmente capaz de adaptarse a condiciones cambiantes, como la propagación de incendios, bloqueos viales o inestabilidad del terreno. Esta capacidad de adaptación es crucial en situaciones de emergencia, donde cada segundo cuenta y las decisiones deben tomarse con información precisa y actualizada.

Esta propuesta no solo aplica conocimientos avanzados adquiridos en el ámbito académico, sino que también responde a necesidades reales con un enfoque práctico, innovador y orientado a generar un impacto positivo en la gestión de desastres y la seguridad de las comunidades, proporcionando rutas de evacuación seguras, facilitando la planificación y coordinación durante desastres, mejorando la eficacia en la gestión de emergencias reduciendo los tiempos de respuesta y minimizando la exposición a riesgos para las comunidades lo cual contribuye a reducir pérdidas humanas y materiales, fortaleciendo la resiliencia comunitaria y optimizando el uso de recursos disponibles. Este enfoque multidimensional garantiza un impacto positivo tanto en la preparación como en la respuesta ante catástrofes.

1.6 Metodología

Se planea crear, implementar y evaluar un prototipo básico de algoritmo de optimización seleccionado para calcular la ruta más segura hacia un punto de evacuación.

La metodología para esta investigación contempla las siguientes etapas.

- **Revisión bibliográfica:** Se revisarán algoritmos genéricos de optimización de rutas a partir de una revisión bibliográfica y técnica. Este análisis permitirá identificar sus principios, fortalezas y limitaciones en el contexto de evacuaciones seguras.
- **Selección de algoritmo:** Se definirán criterios técnicos para la selección del algoritmo a adaptar, considerando aspectos como capacidad de respuesta en condiciones dinámicas, eficiencia computacional y capacidad de integración de variables de seguridad.
- **Adaptación del algoritmo:** Se procederá a la adaptación del algoritmo seleccionado a las necesidades específicas del problema, incorporando factores como zonas de riesgo y bloqueos.
- **Desarrollo del prototipo:** Se desarrollará un prototipo funcional que permita probar el algoritmo en un entorno simulado.
- **Validación:** Se validará el prototipo mediante escenarios diseñados con datos simulados, evaluando la efectividad de las rutas generadas con métodos tradicionales y proponiendo mejoras para optimizar su desempeño.

1.7 Organización del informe en capítulos

En los capítulos subsiguientes se explicará:

- **Marco Teórico:** Este capítulo examina estudios previos y tecnologías relevantes, como el uso de algoritmos de optimización, sistemas de alerta temprana y herramientas de información geoespacial. Se analizan los desafíos y avances en la planificación de rutas de evacuación.
- **Diseño e Implementación:** En este capítulo se presentan las características técnicas del prototipo, incluyendo la integración de algoritmos, datos en tiempo real y geolocalización para calcular rutas seguras. Además, se explican las decisiones tomadas durante la implementación y las métricas utilizadas para evaluar su funcionalidad.
- **Validación:** Se detallan las simulaciones realizadas para probar el sistema en escenarios de emergencia.
- **Evaluación:** Métricas utilizadas para evaluar su funcionalidad y se describen los resultados obtenidos, como tiempos de evacuación, distancias recorridas y seguridad de las rutas, comparándolos con métodos tradicionales.
- **Conclusiones:** Este capítulo explica las áreas de mejora del algoritmo basado en los resultados obtenidos. Se sugieren ajustes técnicos.

2 Marco Teórico

La optimización de rutas de evacuación ha sido un área de investigación ampliamente estudiada en el campo de la planificación urbana y la gestión de desastres. El uso de algoritmos ha sido central en muchos de estos estudios debido a su capacidad para encontrar rutas óptimas en grafos de gran escala. La revisión bibliográfica realizada para efectos de esta tesina sobre algoritmos de búsqueda de caminos aplicados a escenarios de emergencia reveló que las soluciones especializadas en este ámbito son, en su mayoría, adaptaciones o variaciones de tres algoritmos fundamentales: Dijkstra [1], A estrella (A*) [2] y *Jump Point Search* (JPS) [3]. Esta observación se fundamenta en el análisis de estudios previos que abordan la optimización de rutas en contextos dinámicos y de alto riesgo [4], [5], [6], [7], [8], [9]. Dada esta predominancia en la literatura se decidió evaluar estos tres algoritmos como base para el desarrollo de este trabajo. Esta selección permite cubrir un espectro amplio de enfoques algorítmicos, desde soluciones clásicas hasta optimizaciones modernas, proporcionando así un marco sólido para la comparación y adaptación a las necesidades específicas de las rutas de evacuación.

La planificación de rutas de evacuación segura durante situaciones de emergencia no solo implica encontrar la ruta más corta, sino también identificar y mitigar los riesgos que pueden comprometer la seguridad de las personas en movimiento. En áreas vulnerables a desastres naturales, como regiones montañosas o sísmicamente activas, los derrumbes y deslizamientos de tierra son riesgos significativos que pueden bloquear o destruir rutas de evacuación. Durante un terremoto o fuertes lluvias, las carreteras que atraviesan colinas o montañas pueden volverse peligrosamente inestables. Estudios sugieren que el uso de Sistemas de Información Geográfica (SIG) [10] y sensores geotécnicos [11] en tiempo real puede proporcionar datos valiosos para detectar la inestabilidad del terreno, permitiendo recalcular rutas de evacuación antes de que estas sean bloqueadas. La incorporación de esta información en tiempo real a los sistemas de optimización es fundamental para evitar que las rutas se vean afectadas por derrumbes.

Los incendios forestales representan un riesgo complejo para las rutas de evacuación debido a su capacidad de propagarse rápidamente, cambiando de dirección según el viento y el terreno. En este contexto, el riesgo no solo proviene del fuego en sí, sino también del humo, que puede reducir la visibilidad y afectar la salud de los evacuados. Se ha analizado el potencial de la integración de datos en tiempo real provenientes de sensores de fuego y viento [12] en los sistemas de optimización, permitiendo ajustar las rutas dinámicamente, guiando a las personas lejos de fuegos activos y áreas peligrosas [13].

Las condiciones meteorológicas adversas, como tormentas, lluvias intensas o fuertes vientos, pueden complicar significativamente las rutas de evacuación. Las carreteras pueden volverse intransitables por inundaciones repentinas, y la visibilidad puede verse reducida por la niebla o las tormentas de arena. Los modelos de predicción del tiempo y el uso de sensores distribuidos permiten a los sistemas de optimización prever el impacto de estas condiciones meteorológicas en las rutas de evacuación. Se han desarrollado estudios que integran datos meteorológicos satelitales [14] en tiempo real, permitiendo que las rutas de evacuación se ajusten a las condiciones del clima.

Uno de los pilares fundamentales en la gestión de emergencias es la difusión de alertas tempranas a través de medios tecnológicos. En esta línea, sistemas como el Sistema de Alerta de Emergencias (SAE) [15] en Chile han demostrado ser efectivos para advertir a la población sobre riesgos inminentes. Si bien la notificación temprana es clave, su impacto puede verse limitado cuando no va acompañada de información clara sobre rutas de evacuación adaptadas a las condiciones específicas de la emergencia. En otros países, sistemas como *Shake Alert* [16] en Estados Unidos han mostrado la importancia de combinar alertas tempranas con indicaciones de acción inmediatas.

2.1 Descripción de algoritmos de optimización de rutas

Los algoritmos de optimización de rutas buscan caminos óptimos en grafos. Sin embargo, en el contexto de evacuaciones, es necesario modificar la función de costo para incorporar parámetros de seguridad.

2.1.1 Dijkstra

El algoritmo de Dijkstra, desarrollado por Edsger Dijkstra en 1956, es un método ampliamente utilizado por su capacidad para encontrar la ruta más corta desde un nodo inicial a todos los demás nodos en un grafo ponderado, donde cada arista tiene un peso. Ha sido aplicado en escenarios de evacuación para garantizar que las personas puedan escapar rápidamente[9] de áreas peligrosas y es ampliamente utilizado en sistemas de navegación y planificación de rutas. Se ha utilizado tradicionalmente para calcular rutas más cortas en escenarios de evacuación. Este algoritmo explora exhaustivamente todas las rutas posibles desde el nodo inicial hasta todos los nodos, es útil para encontrar la ruta más corta en mapas pequeños o cuando no se dispone de una estimación, sin embargo, presenta limitaciones en términos de eficiencia cuando se enfrenta a redes de gran tamaño y en situaciones donde la planificación debe realizarse en tiempo real. En el contexto de evacuaciones, su aplicación se basa en modelar ciudades o edificios como grafos, donde los nodos representan puntos clave (como salidas, intersecciones o refugios) y las aristas son rutas posibles (calles, pasillos, escaleras). Los pesos en las aristas reflejan distancias, tiempos de viaje o riesgos asociados, como zonas inundables o con obstáculos.

El algoritmo calcula las distancias mínimas desde un nodo inicial hasta todos los demás nodos en un grafo. Su funcionamiento se basa en evaluar, paso a paso, los caminos más cortos, asegurando que cada nodo visitado tenga la distancia mínima acumulada desde el nodo inicial. El proceso se desarrolla de la siguiente manera:

1. En la inicialización se establece la distancia de cada nodo como infinita, excepto el nodo inicial, cuya distancia se define como cero. Este nodo será el punto de partida.
2. En cada iteración, se elige el nodo con la distancia acumulada más corta entre los que aún no han sido visitados. Este nodo garantiza tener la menor distancia posible al nodo inicial debido a la forma en que se construye el algoritmo. A partir del nodo seleccionado, se examinan sus vecinos. Si la distancia desde el nodo inicial a través del nodo actual es menor que la distancia previamente registrada para un vecino se actualiza con este nuevo valor. Este proceso se repite iterativamente, en cada paso, se añade un nuevo nodo al conjunto de nodos cuya distancia mínima ya ha sido determinada.
3. Gradualmente, todas las rutas se optimizan hasta que se han visitado todos los nodos y se han calculado las distancias mínimas y se entregan los resultados.

De esta manera, al finalizar el algoritmo, se obtiene un conjunto de rutas que representan la distancia más corta desde el nodo inicial hasta cada uno de los nodos del grafo, podemos ver este proceso en el ejemplo a continuación:

```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY # Distancia mínima conocida desde el nodo
                             fuente hasta v (inicialmente infinito)
        prev[v] ← UNDEFINED # nodo predecesor de v en el camino más
                             corto (inicialmente indefinido)
        add v to Q # Todos los nodos se añaden al conjunto Q para ser
                    procesados
    dist[source] ← 0 # La distancia del nodo fuente a sí mismo es 0

    while Q is not empty:
        u ← vertex in Q with minimum dist[u]
        Q.remove(u)

        # Para cada arco (u, v) que sale de u
        for each arc (u, v) in Q:
            # Calcular la distancia alternativa desde el nodo fuente
            # hasta v, pasando por u
            alt ← dist[u] + Graph.Edges(u, v)
            # Si esta distancia alternativa es menor que la distancia
            # conocida hasta v, actualizar la distancia y el predecesor
            # de v
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Pseudocódigo 1: Algoritmo Dijkstra[17]

2.1.2 A* (A Estrella)

En contraste, el algoritmo A*, es una mejora de Dijkstra que optimiza la búsqueda al enfocarse en el objetivo usando heurísticas para dirigir la búsqueda hacia el objetivo más rápidamente, lo que reduce significativamente el tiempo de cómputo por lo que es más eficiente para encontrar la ruta más corta en términos de tiempo y recursos, especialmente en mapas grandes o en situaciones donde el tiempo de cálculo es crítico. Varias investigaciones[9], [18] han demostrado que A* es útil para evadir obstáculos en tiempo real, donde es necesario replantear rutas constantemente a medida que cambia el entorno.

El funcionamiento de este algoritmo se puede resumir en:

1. Se determina la función $f(n)=g(n)+h(n)$, donde a cada nodo (n) se le asigna inicialmente un costo total $f(n)$, compuesto por:
 - $g(n)$: Costo acumulado desde el nodo de inicio hasta el nodo actual (inicialmente $g=0$ para el nodo de partida).
 - $h(n)$: Heurística que estima el costo desde el nodo actual hasta el objetivo.
2. Se calcula el costo total. Se selecciona el nodo no visitado con el menor valor de $f(n)$. Este valor indica la mejor estimación del costo total para llegar al objetivo pasando por ese nodo. Se examinan todos los nodos vecinos del nodo seleccionado. Se calcula el nuevo valor de $g(n)$ para cada vecino, actualizando el costo acumulado si se encuentra una ruta más corta. Si se descubre que la nueva ruta hacia un vecino tiene un costo total $f(n)$ más bajo que el previamente registrado, se actualiza este valor y se establece el nodo actual como predecesor en la ruta.
3. Se continúa seleccionando y evaluando nodos hasta llegar al nodo objetivo o hasta que todos los caminos posibles hayan sido explorados. Se entrega los resultados.

De esta manera, al finalizar el algoritmo, se obtiene la ruta más corta desde el nodo inicial hasta el destino específico, optimizando la búsqueda mediante el uso de una heurística que guía el proceso hacia el objetivo de manera eficiente y, por tanto, si se aplica la heurística apropiada se podría llegar a una ruta óptima. Podemos ver este proceso en el ejemplo a continuación:

```
function reconstruct_path(cameFrom, current):
    total_path = {current}
    while current in cameFrom.Keys:
        current = cameFrom[current]
        total_path.prepend(current)
    return total_path

# A* encuentra una ruta desde el nodo inicial (start) hasta el
# objetivo (goal).
# h es la función heurística. h(n) estima el costo para llegar al
# objetivo desde el nodo n.function A_Star(start, goal, h)
# Conjunto de nodos descubiertos que pueden necesitar ser
# (re)expandidos.
# Inicialmente, solo se conoce el nodo de inicio.
# Normalmente se implementa como un montículo mínimo o cola de
# prioridad en lugar de un conjunto hash.
openSet = {start}

# Para un nodo n, cameFrom[n] es el nodo que lo precede en el camino
# más económico conocido desde el inicio hasta n.
cameFrom = an empty map

# Para un nodo n, gScore[n] es el costo conocido actualmente del
# camino más económico desde el inicio hasta n.
gScore = map with default value of Infinity
gScore[start] = 0

# Para un nodo n, fScore[n] = gScore[n] + h(n). fScore[n] representa
# nuestra mejor estimación actual de qué tan económico podría ser un
# camino desde el inicio hasta el objetivo si pasa por n.
fScore = map with default value of Infinity
fScore[start] = h(start)

while openSet is not empty:
    # Esta operación puede realizarse en tiempo  $O(\log(N))$  si
    # conjunto_abierto es un montículo mínimo o cola de prioridad
    current = the node in openSet having the lowest fScore[] value
```

```
if current == goal:
    return reconstruct_path(cameFrom, current)

openSet.Remove(current)

for each neighbor of current:
    # d(current, vecino) es el peso de la arista desde current
    hasta vecino
    # tentative_gScore es la distancia desde el inicio hasta el
    vecino pasando por current
    tentative_gScore = gScore[current] + d(current, neighbor)
    if tentative_gScore < gScore[neighbor]:
        # Si este camino hacia el vecino es mejor que cualquier
        otro anterior regístralo
        cameFrom[neighbor] = current
        gScore[neighbor] = tentative_gScore
        fScore[neighbor] = tentative_gScore + h(neighbor)
        if neighbor not in openSet:
            openSet.add(neighbor)

# El conjunto abierto está vacío pero nunca se alcanzó el objetivo
return failure
```

Pseudocódigo 2: Algoritmo A Estrella[19]

2.1.3 JPS

Técnicas como JPS y algoritmos jerárquicos han sido propuestas para reducir el riesgo en zonas marítimas desconocidas y en situaciones de guerra, donde los cálculos en tiempo real son críticos para guiar evacuaciones seguras [7], [8]. Estos enfoques combinan la rapidez de A* con optimizaciones específicas que permiten manejar grandes volúmenes de datos geoespaciales, haciendo más eficientes las rutas calculadas.

JPS es una optimización del algoritmo A* diseñada para mejorar la eficiencia en la búsqueda de rutas en cuadrículas de costo uniforme. En una cuadrícula, múltiples caminos pueden tener costos idénticos debido a la simetría. JPS elimina estas redundancias al identificar puntos de salto (*Jump Points*) clave, desde los cuales la búsqueda puede avanzar directamente, omitiendo nodos intermedios que no afectan la decisión del camino óptimo. En lugar de explorar todos los vecinos, se utiliza la función salto para avanzar en línea recta hasta encontrar un punto de interés (punto de salto). Son nodos que cumplen criterios específicos, como ser puntos de inicio o destino, tener vecinos forzados (nodos que deben ser considerados debido a obstáculos) o estar en direcciones diagonales con ciertas condiciones. Al identificar y omitir nodos redundantes, JPS acelera el proceso de búsqueda sin comprometer la optimalidad del camino encontrado. Su funcionamiento se puede simplificar a:

1. Inicialización: Se configura el conjunto de nodos abiertos, cerrados y las tablas de costos de manera similar a A*.
2. Búsqueda de puntos de salto: Se busca nodos que actúen como anclas en la búsqueda. Los nodos irrelevantes intermedios son ignorados.
3. Poda del Espacio de Búsqueda: Al avanzar en la cuadrícula, JPS evalúa direcciones y determina si puede continuar en línea recta o si debe cambiar de dirección al encontrar un punto de salto o un obstáculo. Esto reduce significativamente el número de nodos evaluados en comparación con A*.
4. Actualización de costos: Si se encuentra un nuevo punto de salto con un menor costo acumulado, se actualiza su costo, el costo total estimado y el nodo predecesor.
5. Forzar vecinos: La función salto también detecta cambios en la dirección debido a obstáculos (vecinos bloqueados), asegurando que se exploren solo los caminos esenciales.
6. Reconstrucción de la ruta: Una vez alcanzado el nodo objetivo, se reconstruye la ruta siguiendo los predecesores desde el objetivo hasta el nodo inicial. Se entrega los resultados.

Podemos ver este proceso en el ejemplo a continuación:

```
function jumpPointSearch(start, goal, grid):
    openSet = PriorityQueue()
    openSet.add(start, heuristic(start, goal))
    cameFrom = empty map

    gScore = map with default value Infinity
    gScore[start] = 0

    fScore = map with default value Infinity
    fScore[start] = heuristic(start, goal)

    while openSet is not empty:
        current = openSet.pop()

        if current == goal:
            return reconstructPath(cameFrom, current)

        for each neighbor in getPrunedNeighbors(current,
            cameFrom.get(current)):
            jumpPoint = jump(neighbor, direction, goal, grid)

            if jumpPoint is not null:
                tentative_g = gScore[current] + distance(current,
                    jumpPoint)

                if tentative_g < gScore[jumpPoint]:
                    cameFrom[jumpPoint] = current
                    gScore[jumpPoint] = tentative_g
                    fScore[jumpPoint] = tentative_g +
                        heuristic(jumpPoint, goal)

                    if jumpPoint not in openSet:
                        openSet.add(jumpPoint,
                            fScore[jumpPoint])

    return failure

function jump(node, direction, goal, grid):
    nextNode = node + direction

    if nextNode is blocked or out of bounds:
        return null

    if nextNode == goal:
        return nextNode

    # Verificar si el nodo actual es un "punto de salto forzado"
    if hasForcedNeighbor(nextNode, direction, grid):
        return nextNode
```

```
if direction is diagonal:
    # Verificar si hay un punto de salto en la componente
    horizontal de la dirección
    if jump(nextNode, horizontal_component(direction), goal,
    grid) is not null:
        return nextNode
    if jump(nextNode, vertical_component(direction), goal, grid)
    is not null:
        return nextNode

return jump(nextNode, direction, goal, grid)

function getPrunedNeighbors(node, parent):
    if parent is null:
        return all walkable neighbors

    neighbors = []
    direction = normalize(node - parent)

    # Añadir vecinos naturales (a los lados)
    for each dir in getNaturalDirections(direction):
        neighbor = node + dir
        if neighbor is walkable:
            neighbors.add(neighbor)

    # Añadir vecinos forzados (con obstáculos)
    for each dir in getForcedDirections(node, direction, grid):
        neighbor = node + dir
        if neighbor is walkable:
            neighbors.add(neighbor)

    return neighbors
```

Pseudocódigo 3: Algoritmo JPS

2.1.4 Otros Algoritmos

Estos algoritmos son herramientas fundamentales en la planificación de rutas de evacuación, garantizando rutas seguras y eficientes durante una emergencia. Sin embargo, la investigación más reciente ha comenzado a explorar variaciones más avanzadas de estos algoritmos tales como algoritmos jerárquicos [5] para reducir la complejidad computacional en entornos de cuadrícula, lo que es útil en simulaciones de evacuación en tiempo real en áreas urbanas. Además, el algoritmo *D-Lite** [6], que permite replanificar rutas en entornos dinámicos, ha mostrado ser eficaz en escenarios donde la infraestructura o las condiciones cambian rápidamente, como durante terremotos o incendios forestales.

2.2 Tecnologías y *frameworks* utilizados

Para el desarrollo del prototipo de Alertify se emplearán los siguientes *frameworks* y herramientas clave, seleccionados por su capacidad para manejar datos en tiempo real, procesamiento geoespacial y comunicación eficiente entre cliente y servidor:

- a. *Firebase*: es una plataforma de desarrollo de aplicaciones móviles y web proporcionada por Google. Su infraestructura en la nube ofrece servicios escalables que facilitan la gestión de datos en tiempo real, autenticación y almacenamiento.

Se utilizará como base de datos en tiempo real para almacenar y sincronizar datos como la ubicación de los usuarios, el estado de las rutas y alertas de emergencia. Su capacidad para actualizar información en tiempo real asegura que los usuarios reciban rutas actualizadas al instante. Facilita la identificación y autenticación de los usuarios, garantizando un acceso seguro y restringido a la aplicación.

Se usarán mensajes *push* para enviar notificaciones inmediatas sobre cambios en las condiciones de las rutas o nuevas alertas de emergencia.

Ventajas técnicas:

- Sincronización en tiempo real, ideal para aplicaciones críticas como la gestión de emergencias.
 - Amplio ecosistema que permite integración con otras herramientas de *Google*.
- b. *Google Maps API* es un conjunto de herramientas geoespaciales que permite integrar mapas interactivos y datos de navegación en aplicaciones. Ofrece funciones como geolocalización, trazado de rutas y cálculo de distancias.

Permitirá identificar la geolocalización actual de los usuarios, un componente esencial para calcular rutas personalizadas hacia las zonas seguras. Se utilizará el trazado de rutas para marcar rutas óptimas utilizando datos de tráfico, distancias y condiciones viales.

Ventajas técnicas:

- Precisión en datos geoespaciales y de navegación.
 - Amplio soporte técnico y documentación detallada para desarrolladores.
- c. El desarrollo del algoritmo de optimización de rutas requiere un lenguaje de programación flexible y potente, capaz de integrar algoritmos complejos y adaptarse a condiciones dinámicas. Python es una opción ideal debido a su versatilidad y a la disponibilidad de librerías especializadas, como:
- Librerías matemáticas (ej. Math, NumPy) para cálculos avanzados, como la distancia euclidiana entre nodos o la estimación de costos en la función heurística del algoritmo.
 - Librerías de visualización (ej. Pygame, Matplotlib) para representar gráficamente el grafo de rutas, las zonas de riesgo y las rutas de evacuación generadas, facilitando la validación y el análisis del sistema.

- Librerías de manejo de tiempo (ej. Time) para medir el rendimiento del algoritmo y evaluar su eficiencia en términos de tiempo de cómputo, un aspecto crítico en situaciones de emergencia.
- d. *Websockets* es un protocolo de comunicación que establece una conexión persistente y bidireccional entre el cliente y el servidor. Es ideal para aplicaciones que requieren una transferencia continua de datos en tiempo real.

Websockets se empleará para mantener una conexión activa entre el cliente y el servidor, permitiendo la actualización instantánea de las rutas y el intercambio de información crítica como nuevos riesgos detectados o cambios en las condiciones de evacuación. Facilitará el envío de notificaciones en tiempo real a los usuarios sobre cambios en las rutas o emergencias.

Ventajas técnicas:

- Latencia baja y transferencia eficiente de datos.
- Capacidad para manejar múltiples conexiones simultáneamente, lo que es crucial en situaciones de emergencia con muchos usuarios activos.
- Mejora la experiencia del usuario al garantizar actualizaciones inmediatas.

En el contexto de esta investigación, nos centraremos en el uso de Google Maps API como herramienta principal para la visualización geoespacial y el cálculo de rutas, complementado con el desarrollo del algoritmo de optimización en Python. Google Maps API permite integrar mapas interactivos y datos de navegación, lo que facilita la identificación de la geolocalización de los usuarios y el trazado de rutas personalizadas hacia zonas seguras. Sin embargo, debido a las limitaciones de la API para implementar algoritmos personalizados de búsqueda de rutas, se utilizará Python para desarrollar y probar el modelo de optimización de manera independiente.

Python, junto con librerías como Pygame para la visualización y Math para los cálculos matemáticos, nos permitirá simular escenarios de emergencia y evaluar el rendimiento del algoritmo A* modificado. Esta combinación de herramientas asegura que podamos validar la efectividad del modelo propuesto en un entorno controlado, sentando las bases para futuras implementaciones en plataformas móviles o sistemas de alerta temprana.

3 Diseño e Implementación

3.1 Método de optimización de ruta

Podemos resumir los métodos de optimización vistos en el capítulo 2 en la siguiente tabla:

	Dijkstra	A*	JPS
Principio Básico	Encuentra la ruta más corta evaluando todos los nodos accesibles desde el inicial.	Utiliza una heurística para priorizar la búsqueda hacia el objetivo.	Optimiza A* en cuadrículas de costo uniforme eliminando nodos redundantes.
Estrategia de Búsqueda	Explora exhaustivamente todas las rutas posibles, independientemente del objetivo.	Prioriza los nodos según la heurística y el costo acumulado.	Salta sobre nodos redundantes en la cuadrícula, evaluando solo "puntos de salto".
Uso de Heurística	No utiliza heurística.	Utiliza heurísticas para estimar el costo al objetivo, como la distancia Manhattan o Euclidiana.	Utiliza heurística para dirigirse hacia el objetivo, pero optimiza la evaluación en cuadrículas.
Espacio de Aplicación	Grafos generales con cualquier tipo de ponderación.	Grafos generales y cuadrículas, adecuado para entornos dinámicos.	Cuadrículas con costos uniformes o discretos.
Eficiencia Computacional	Relativamente lento en grafos grandes debido a la evaluación exhaustiva.	Más eficiente que Dijkstra gracias a la heurística.	Más rápido que A* en cuadrículas al reducir el número de nodos evaluados.
Complejidad Espacial	Alta: mantiene datos para todos los nodos visitados.	Moderada: depende de la heurística y del tamaño del conjunto abierto.	Baja: gracias a la eliminación de nodos intermedios.
Ventajas	- Encuentra rutas óptimas independientemente de la heurística. - Aplicable a cualquier tipo de grafo.	- Flexibilidad en diferentes tipos de grafos y cuadrículas. - Rápido en entornos dinámicos con buena heurística.	- Excelente en cuadrículas de costo uniforme. - Significativamente más rápido al evitar nodos redundantes.
Limitaciones	- Ineficiente en grafos grandes o tiempo real. - No tiene dirección hacia el objetivo.	- Requiere una buena heurística para maximizar su eficiencia.	- Limitado a cuadrículas uniformes. - No es tan efectivo en grafos generales.
Casos de Uso Comunes	- Redes de carreteras con diferentes pesos. - Problemas donde no se puede usar heurística.	- Sistemas de navegación en tiempo real. - Juegos, robótica y simulación de evacuación.	- Juegos y simulación en cuadrículas. - Navegación en entornos con patrones repetitivos o uniformes.

En base a esto podemos decir que:

- a. Dijkstra es un método adecuado para grafos donde no se dispone de heurísticas o para aplicaciones generales que requieren resultados garantizados. No se considera adecuado para el proyecto debido a las siguientes limitaciones clave en este contexto:
 - Falta de enfoque direccional: Dijkstra explora todos los nodos del grafo de manera uniforme, sin priorizar direcciones específicas. En una evacuación, es crucial guiar la búsqueda hacia zonas seguras (ej: salidas de emergencia) de manera eficiente, algo que Dijkstra no puede hacer sin una adaptación compleja.
 - Ineficiencia en grafos grandes: Dijkstra no utiliza heurísticas, lo que lo hace menos eficiente en grafos extensos (como edificios grandes o ciudades). En emergencias, el tiempo de cómputo es crítico, y Dijkstra podría explorar rutas irrelevantes antes de encontrar una salida segura.
 - Sensibilidad a cambios dinámicos: En emergencias, las condiciones del grafo pueden cambiar rápidamente (ej: rutas bloqueadas, nuevos riesgos). Dijkstra no está diseñado para adaptarse dinámicamente a estos cambios sin reiniciar el proceso.
- b. Aunque JPS es una optimización eficiente de A* para cuadrículas de costo uniforme, no se considera adecuado para este proyecto. Esto se debe a que:
 - Asume costos uniformes: JPS está diseñado para entornos donde el costo de viajar entre cuadrículas es uniforme. En el contexto de evacuaciones, el riesgo no es uniforme: algunas áreas pueden ser extremadamente peligrosas (ej. zonas con incendios activos), mientras que otras pueden ser seguras. JPS no puede incorporar esta variabilidad en los costos, lo que limita su utilidad para generar rutas seguras.
 - Implementación compleja: La implementación es más compleja que A* ya que es una optimización específica para cuadrillas uniformes y requiere la implementación de reglas para identificar saltos y podar movimientos.
- c. A* es versátil y eficiente, aplicable en una variedad de contextos, especialmente entornos con diferentes tipos de grafos. Ventajas de A* en casos de evacuaciones:
 - Es genérico y aplicable a cualquier tipo de grafo, ya sea una red de calles, un edificio con pasillos irregulares, o un espacio con obstáculos no alineados a una cuadrícula. Esto es crucial en evacuaciones, donde los entornos suelen ser heterogéneos y no necesariamente cuadrangulares (ej: edificios con pasillos diagonales, plazas abiertas, o áreas con geometrías complejas).
 - Posee buen desempeño en mapas grandes al concentrar la búsqueda en zonas relevantes, haciéndolo adecuado para buscar rutas de evacuación en zonas complejas.

Por todo lo señalado, se considera A* la alternativa ideal para el prototipo de planificación de rutas de evacuación porque combina eficiencia en la búsqueda de rutas óptimas con la capacidad de adaptación a cambios dinámicos, capacidad de adaptación a cambios en tiempo real y personalización mediante heurísticas, además de ser computacionalmente más eficiente que Dijkstra y al ser basado en grafo tiene mejor adaptabilidad a la geografía de caminos que JPS. Su heurística personalizada permite no solo encontrar la ruta más corta, sino también la más segura, integrando factores como

riesgos de derrumbes, incendios o bloqueos de carreteras, lo cual es crucial para minimizar riesgos durante emergencias. Esto es fundamental para ajustarse a las condiciones cambiantes durante desastres naturales y garantizar una evacuación rápida y efectiva.

Además, A^* es escalable y puede manejar redes complejas como las de áreas urbanas, siendo altamente adecuado para aplicaciones en tiempo real. La gran cantidad de bibliotecas disponibles facilita su implementación rápida y eficiente, algo crítico considerando los plazos ajustados del proyecto.

3.2 Desarrollo del prototipo.

Durante el desarrollo del prototipo, se identificaron ciertas limitaciones técnicas relacionadas con el uso de la *API* de *Google Maps*. En particular, se evidenció la imposibilidad de implementar un algoritmo propio de búsqueda, como A^* mejorado, utilizando los datos generados por dicha *API*[20], principalmente porque sus términos de servicio prohíben descargar, almacenar o reutilizar sus datos para crear sistemas alternativos, como bases de datos o algoritmos de navegación personalizados. Además, no brinda acceso directo al grafo completo de carreteras y conexiones necesario para desarrollar estos algoritmos, ya que solo ofrece funcionalidades específicas (como cálculo de rutas) a través de sus endpoints. Esto, sumado a las restricciones técnicas y legales que impiden la extracción o ingeniería inversa de sus datos, limita su uso a lo que la *API* expone directamente, sin posibilidad de manipular la información subyacente para propósitos externos.

Google Maps realiza sus búsquedas de caminos internamente y entrega como resultado la ruta óptima ya calculada. Sin embargo, no expone la representación del grafo utilizado en sus procesos de búsqueda, lo cual es necesario para la implementación y prueba de algoritmos como A^* . Esta representación es esencial para evaluar diferentes caminos posibles y ajustar las rutas dinámicamente según las condiciones en tiempo real.

Al no poder acceder a los nodos y aristas utilizados por la *API*, se imposibilita la aplicación de algoritmos personalizados de búsqueda de rutas. Esto afecta la capacidad del sistema para realizar pruebas comparativas entre rutas generadas por el algoritmo A^* y las ofrecidas por la *API*. Esto imposibilita probar y ajustar rutas de evacuación en tiempo real.

Por ello, se optó por desarrollar el prototipo en Python, un lenguaje flexible que facilita la implementación de algoritmos de búsqueda de rutas. Además, se utilizó Pygame para visualizar el grafo, las zonas de riesgo y las rutas generadas. Esta elección permite:

- Implementar y ajustar libremente el algoritmo A^* con la función de costo mejorada, sin depender de sistemas externos.
- Simular escenarios como incendios o bloqueos viales, para evaluar cómo el algoritmo adapta las rutas en tiempo real.
- Visualizar de manera clara el grafo, las zonas de riesgo y las rutas generadas, facilitando el análisis de su comportamiento.
- Realizar pruebas comparativas que validen la eficiencia y seguridad de las rutas propuestas.

La decisión de utilizar Python y Pygame para el desarrollo del prototipo responde a la necesidad de superar las limitaciones de *Google Maps*, al tiempo que se aprovechan las ventajas de un entorno flexible, visual y fácil de implementar. Esta elección no solo permite validar el algoritmo A^* con función

de costo mejorada en condiciones controladas, sino que también sienta las bases para futuras expansiones, como la integración con sistemas de alerta temprana o la adaptación a plataformas móviles. En última instancia, esta combinación tecnológica garantiza que el prototipo cumpla con su objetivo principal: demostrar la efectividad de rutas de evacuación seguras y adaptativas en escenarios de emergencia.

3.3 Parámetros de penalización

El algoritmo A* tradicional se basa en una función de costo que combina el costo acumulado desde el nodo inicial ($g(n)$) y una heurística que estima el costo restante hasta el objetivo ($h(n)$), generalmente representada como:

$$f(n) = g(n) + h(n)$$

Sin embargo, en escenarios de evacuación, donde la seguridad es tan crítica como la distancia, esta función debe adaptarse para incorporar factores de riesgo dinámicos. Por ello, se propone una función de costo mejorada que integra una penalización por riesgo, transformando la ecuación en:

$$f(n) = g(n) + h(n) + riesgo(n)$$

Para modelar el riesgo se definió un factor de penalización proporcional a la distancia del nodo respecto al foco del incendio. La penalización se aplicó mediante la fórmula:

Donde:

$$C_{modificado} = C_{base} + \alpha \cdot 1/(d + 1)$$

- $C_{modificado}$ es el nuevo costo total del nodo.
- C_{base} es el costo del desplazamiento estándar $f(n) = g(n) + h(n)$.
- α es el coeficiente de penalización, ajustado experimentalmente.
- d es la distancia euclidiana del nodo respecto al foco del incendio.

De este modo, los nodos más cercanos al fuego se vuelven menos atractivos para la búsqueda, desviando la ruta hacia zonas más seguras.

4 Validación

4.1 Metodología de validación

Para comprobar la efectividad del sistema desarrollado y evaluar el comportamiento del algoritmo propuesto, se llevaron a cabo simulaciones en entornos controlados que representaron escenarios de emergencia, considerando condiciones dinámicas como la presencia y propagación del fuego, bloqueos de rutas y variaciones en la densidad de nodos disponibles para la evacuación.

El proceso de validación se centró en comparar el rendimiento del algoritmo A* tradicional con una versión modificada del mismo algoritmo, adaptada para contextos de emergencia. Esta versión penaliza el acercamiento a nodos que se encuentran dentro de un radio de influencia del fuego, lo que permite generar rutas más seguras, incluso si la distancia total recorrida resulta ligeramente mayor.

Para poder medir la funcionalidad de este prototipo utilizaremos las siguientes métricas de evaluación:

- Tiempo de cómputo del algoritmo: Tiempo requerido por el algoritmo para calcular la ruta óptima de evacuación desde el punto de inicio hasta el destino seguro. Este indicador evalúa la eficiencia computacional del sistema en condiciones.
- Distancia total de la ruta: desde la ubicación inicial del usuario hasta el punto de evacuación.
- Seguridad de la ruta calculada: evaluando la ruta en función de zonas de riesgo identificadas asignando un índice de seguridad que refleje la exposición a estos riesgos
- Eficiencia computacional: número de nodos explorados por el algoritmo durante la búsqueda de la ruta más segura, lo que permite evaluar la optimización del proceso y su capacidad para encontrar soluciones sin un consumo excesivo de recursos.

4.2 Escenario simulado

El entorno de simulación consistió en una cuadrícula de 50x50 nodos que representa una zona urbana simplificada con diferentes puntos de riesgo y una zona segura. En este escenario se definieron:

- Punto de inicio: ubicación del usuario o grupo de evacuación.
- Punto de destino: zona segura designada fuera del área de peligro.
- Áreas de riesgo: regiones de la cuadrícula que simulan incendios activos, las cuales influyen en la penalización de los nodos cercanos.
- Nodos bloqueados: calles o sectores intransitables por obstrucción o daño estructural.

Se implementaron dos versiones del sistema:

- A* clásico: calcula la ruta más corta posible entre el punto inicial y el destino sin considerar factores de riesgo.
- A* modificado: incluye una función de penalización que incrementa el costo de los nodos cercanos a fuentes de peligro (fuego) o zonas con alta concentración de riesgo.

4.3 Librerías utilizadas

Para la implementación del algoritmo de optimización de rutas A*, se utilizó el lenguaje de programación Python debido a su capacidad para integrar fácilmente algoritmos complejos y su facilidad para visualizar el funcionamiento de estos. Esta elección de lenguaje asegura que el prototipo sea escalable y adaptable a futuras mejoras, como la integración con APIs de mapas alternativas o la incorporación de datos dinámicos de sensores en tiempo real. A continuación, se describen las principales librerías empleadas y su propósito específico en el desarrollo del prototipo:

- **Math:** Utilizada para realizar cálculos matemáticos básicos, como la distancia euclidiana entre nodos, que es clave para la heurística del algoritmo A*. Esta librería facilita la implementación de funciones matemáticas necesarias para estimar costos y priorizar rutas.
- **Pygame:** Librería utilizada para la visualización interactiva de los escenarios de simulación y las rutas generadas por el algoritmo. Pygame permitió crear una interfaz gráfica sencilla para representar el mapa de nodos, las zonas de riesgo (como incendios o bloqueos) y las rutas de evacuación calculadas. Esto facilitó la validación visual de los resultados y la demostración del funcionamiento del prototipo en un entorno controlado.
- **Time:** Empleada para medir el tiempo de ejecución de las simulaciones y del algoritmo, permitiendo evaluar la eficiencia del prototipo en términos de rendimiento temporal computacional. Esto es crucial para validar que el sistema pueda operar en tiempo real, un requisito esencial en contextos de emergencia.

4.4 Simulaciones realizadas

Se llevaron a cabo simulaciones representativas con distintos patrones de propagación del fuego y ubicaciones del punto de inicio. En cada escenario, se ejecutó el algoritmo de optimización de rutas y midió su capacidad para generar rutas seguras y eficientes. Cada simulación fue ejecutada utilizando tanto el algoritmo A* clásico (A* Original) como el A* modificado (A* modified).

Se registraron las siguientes métricas:

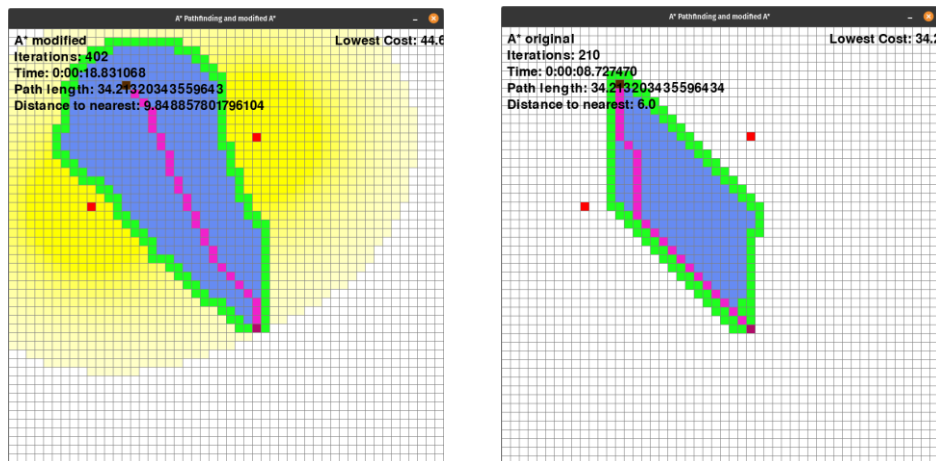
- **Tiempo total de cómputo:** cuánto tiempo toma encontrar la ruta de evacuación desde la ubicación inicial hasta el punto de seguridad. Se llamó "*time*".
- **Longitud de la ruta obtenida:** longitud total de la ruta generada. Se llamó "*path length*".
- **Distancia mínima al fuego durante la evacuación:** Distancia mínima a áreas peligrosas, como cercanía a incendios. Se llamó "*distance to nearest*".
- **Número de nodos explorados:** cantidad de nodos que revisó el algoritmo al buscar la ruta más segura. Se llamó "*iterations*".

Los colores que se ven en las visualizaciones son representaciones gráficas de:

- **Negro:** Barreras, son obstáculos que el algoritmo no puede atravesar.
- **Rojo:** Celdas bloqueadas, representan áreas de "fuego" o zonas bloqueadas.

- Amarillo: Celdas con peso, alrededor de las áreas de “fuego” o zonas bloqueadas. Estas celdas tienen un costo adicional (penalización) para pasar por ellas.
- Marrón: Punto de inicio del camino.
- Morado: Punto final del camino.
- Verde y azul: Celdas revisadas por el algoritmo.
- Rosa: Línea que representa el camino encontrado.

A continuación, se presentan los resultados de algunas de las simulaciones realizadas para comparar el comportamiento del algoritmo *A* original* y el *A* modificado* en diferentes escenarios de evacuación. Cada escenario simula condiciones específicas de riesgo, como la presencia de focos de incendio, bloqueos parciales y zonas de peligro dispersas.



Escenario 1: Focos de incendio a medio camino

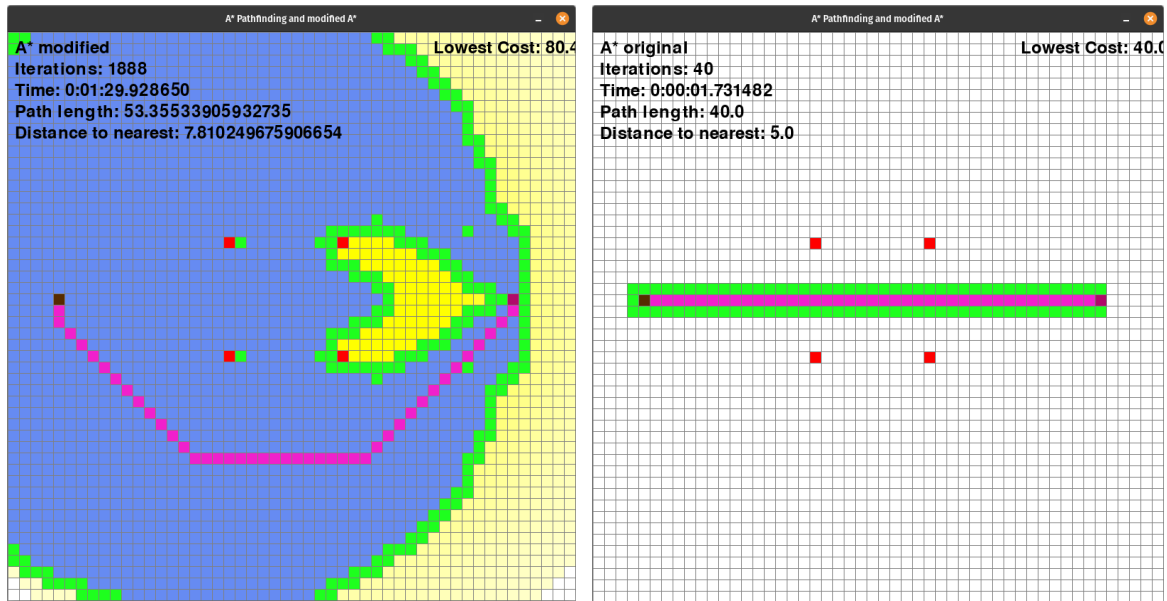
Se simula un mapa con dos focos de incendio ubicados a medio camino entre el punto de inicio (marrón) y el punto de destino (morado).

El objetivo es evaluar cómo cada algoritmo evita las zonas de riesgo y calcula rutas alternativas.

Métrica	A* Modificado	A* Original
Iteraciones	402	210
Tiempo de cómputo	00:18.8	00:08.7
Longitud del camino	34.21	34.21
Distancia mínima al fuego	9.848	6

A* Modificado realiza más iteraciones y tarda más tiempo en calcular la ruta, pero logra una mayor distancia al fuego, lo que indica una ruta más segura.

A* Original encuentra una ruta más rápidamente, pero con mayor exposición al riesgo.



Escenario 2: Focos múltiples dispersos

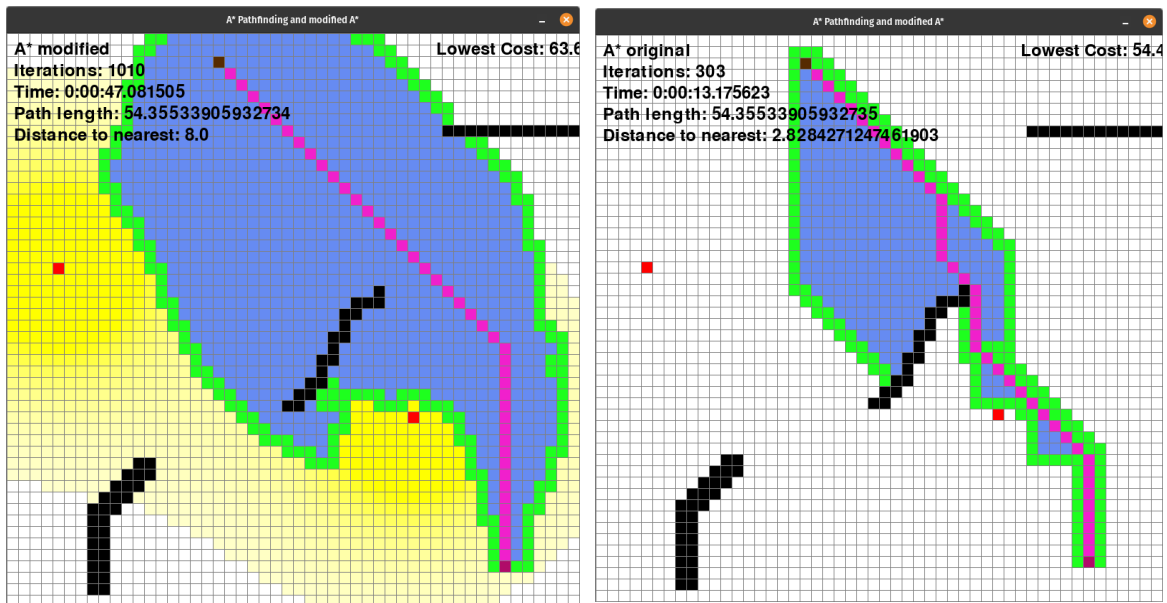
Se simula un mapa con múltiples focos de incendio dispersos en diferentes áreas.

El objetivo es evaluar cómo los algoritmos manejan la presencia de varios puntos de riesgo.

Métrica	A* Modificado	A* Original
Iteraciones	1888	40
Tiempo de cómputo	01:29.9	00:01.7
Longitud del camino	53.35	40
Distancia mínima al fuego	7.810	5

A* Modificado nuevamente realiza más iteraciones y tarda más tiempo, pero logra una mayor distancia mínima al fuego, lo que indica una ruta más segura.

A* Original es más rápido, pero la ruta resultante tiene una mayor exposición al riesgo.



Escenario 3: Bloqueos parciales y focos dispersos

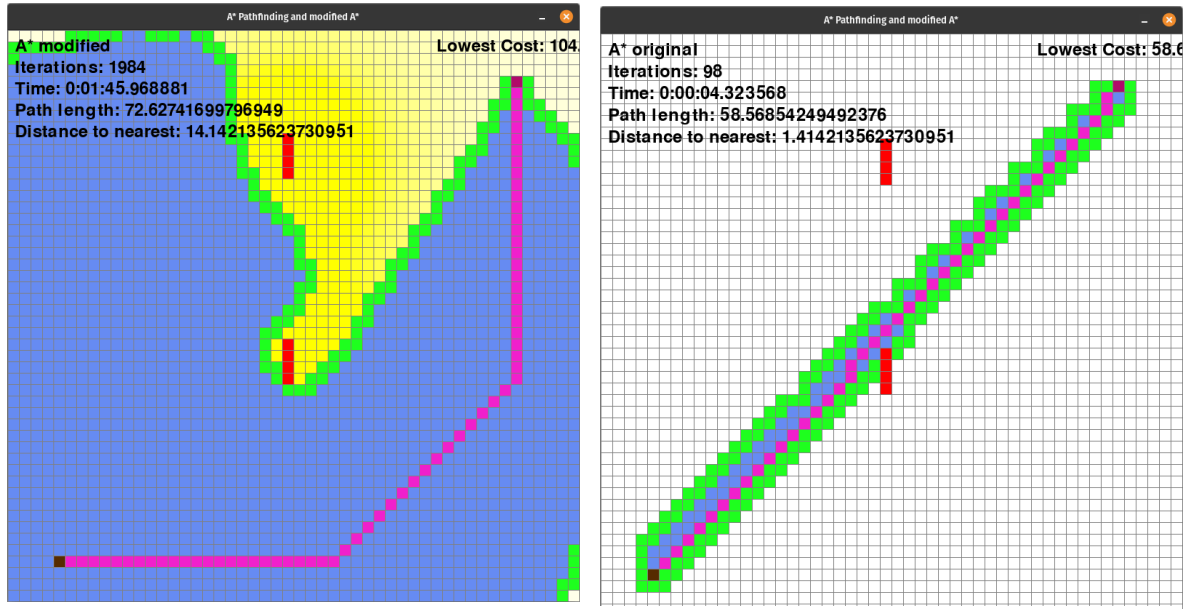
Se simula un mapa con bloqueos parciales y focos de incendio dispersos.

El objetivo es evaluar cómo los algoritmos manejan obstáculos y zonas de riesgo combinadas.

Métrica	A* Modificado	A* Original
Iteraciones	1010	303
Tiempo de cómputo	00:47.1	00:13.2
Longitud del camino	54.35	54.35
Distancia mínima al fuego	8	2.824

A* Modificado prioriza la seguridad, evitando zonas de riesgo y bloqueos, aunque puede tardar más en calcular la ruta.

A* Original es más rápido, pero se acerca más a las zonas de peligro.



Escenario 4: Bloqueo parcial con corredor libre

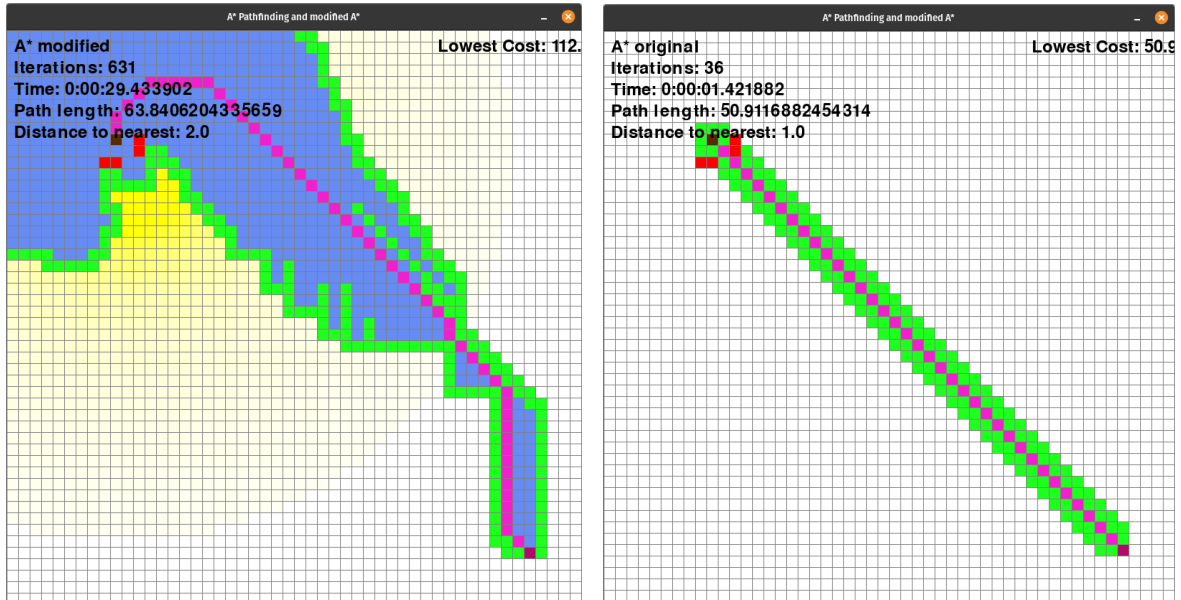
Se simula un mapa con un bloqueo parcial y un corredor libre que conecta el punto de inicio con el destino.

El objetivo es evaluar cómo los algoritmos aprovechan el corredor libre para evitar riesgos.

Métrica	A* Modificado	A* Original
Iteraciones	1984	98
Tiempo de cómputo	01:45.9	00:04.3
Longitud del camino	72.62	58.56
Distancia mínima al fuego	14.142	1.414

A* Modificado aprovecha el sector libre para evitar zonas de riesgo, aunque puede realizar más iteraciones.

A* Original puede no priorizar la seguridad, optando por rutas más cortas pero potencialmente más peligrosas.



Escenario 5: Zona de fuego cerca de inicio

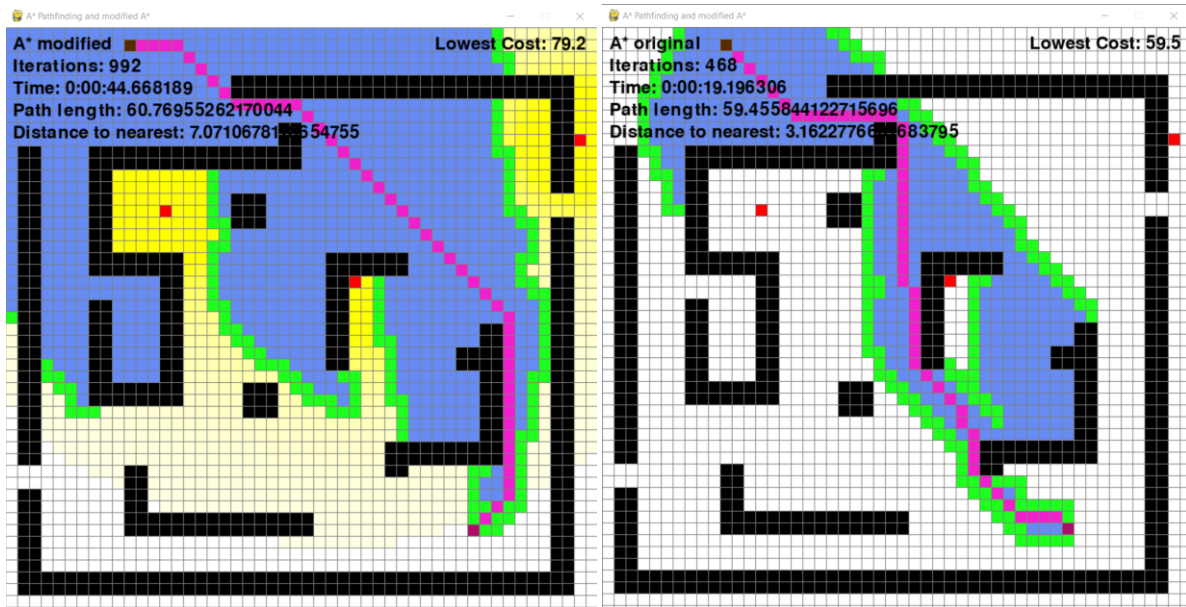
Se simula un mapa con una zona de fuego cercana al punto de inicio.

El objetivo es evaluar cómo los algoritmos evitan el riesgo desde el principio del recorrido

Métrica	A* Modificado	A* Original
Iteraciones	631	36
Tiempo de cómputo	00:29.4	00:01.4
Longitud del camino	63.84	50.91
Distancia mínima al fuego	2	1

A* Modificado evita la zona de fuego desde el inicio, garantizando una ruta más segura.

A* Original no evita adecuadamente el riesgo inicial, priorizando la longitud del camino.



Escenario 6: Laberinto

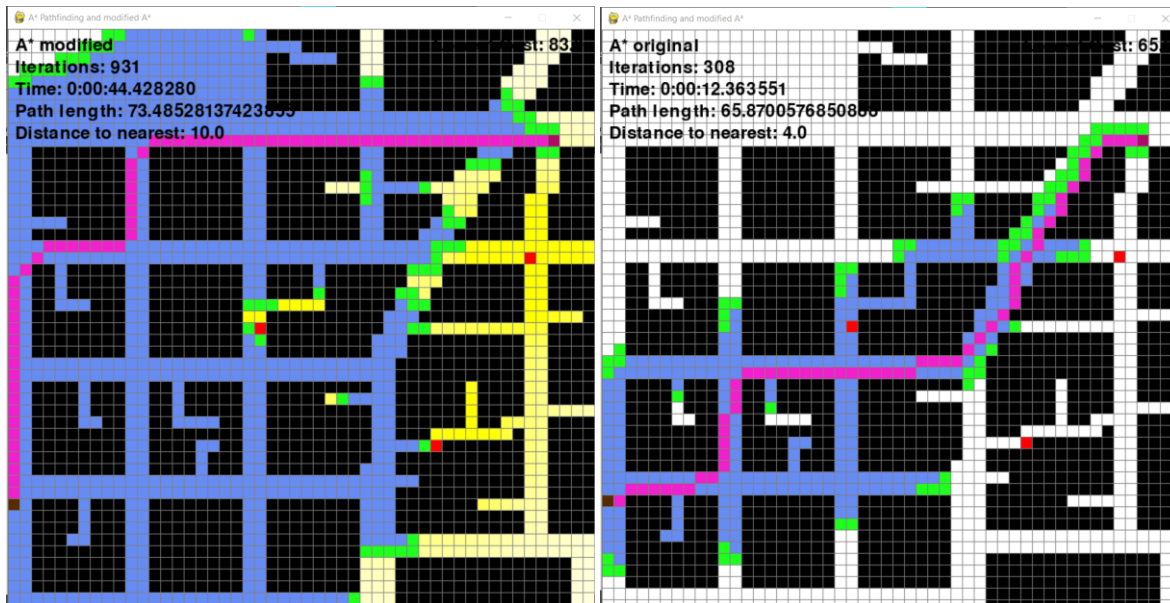
Se simula un laberinto con tres zonas de fuego.

El objetivo es evaluar cómo los algoritmos actúan con caminos más cerrados.

Métrica	A* Modificado	A* Original
Iteraciones	992	468
Tiempo de cómputo	00:44.7	00:19.2
Longitud del camino	60.8	59.5
Distancia mínima al fuego	7	3

A* Modificado se aleja de las zonas de fuego, garantizando una ruta más segura.

A* Original prioriza la longitud del camino.



Escenario 7: Viña del mar

Se generó un mapa que simula las calles de viña del mar, entre Av Libertad y 4 Oriente, para poder tener una visualización de cómo se comporta el algoritmo en un ambiente más restringido.

El objetivo es evaluar cómo los algoritmos trabajan en un mapa realista.

Métrica	A* Modificado	A* Original
Iteraciones	931	308
Tiempo de cómputo	00:44.4	00:12.4
Longitud del camino	73.5	65.9
Distancia mínima al fuego	10	4

A* Modificado evita las calles cercanas a las zonas de fuego.

A* Original pasa entre medio de las zonas de fuego, aumentando el riesgo.

5 Evaluación

5.1 Métricas de evaluación

Para evaluar el desempeño del sistema de optimización de rutas de evacuación, se definieron métricas cuantitativas que permiten medir tanto la eficiencia computacional como la seguridad de las rutas generadas. Estas métricas son fundamentales para comparar el rendimiento del algoritmo A* original con el algoritmo A* modificado, especialmente en contextos de emergencia donde la seguridad y la rapidez son críticas.

- a. Tiempo de evacuación (T_e): Representa el tiempo estimado que un usuario tardaría en completar la ruta calculada.
 - Un menor tiempo indica mayor eficiencia.
- b. Distancia total recorrida (D_r): Corresponde a la longitud total del camino encontrado.
 - En A*, el objetivo es minimizar D_r .
 - En A* modificado, se acepta un incremento moderado de D_r si esto implica una mayor distancia al peligro.
- c. Distancia mínima al peligro (D_p): Mide la proximidad más corta entre el usuario y una zona de riesgo durante la ruta.
 - Un menor valor de D_p implica mayor exposición al peligro.
 - En contextos de emergencia, se prioriza aumentar D_p incluso si la distancia total crece.
- d. Número de nodos explorados (N_n): Evalúa la eficiencia computacional. Un menor número de nodos explorados refleja un algoritmo más eficiente.

5.2 Resultados comparativos

Tras la ejecución de las simulaciones en los cinco escenarios descritos, se observó que el algoritmo A modificado* generó rutas más seguras en todos los casos evaluados. Los principales hallazgos fueron:

- Aumento de la distancia recorrida: En promedio, un 20% mayor que el A* clásico, debido a los desvíos hacia zonas seguras.
- Reducción significativa del riesgo: La distancia mínima al peligro (D_p) se incrementó en un 250%, reduciendo sustancialmente la exposición directa al peligro.
- Tiempo de cómputo mayor: Aunque el tiempo de cómputo fue 15 veces mayor en promedio, la eficiencia en términos de seguridad justifica este costo.
- Exploración más dirigida: Aunque el número de nodos explorados aumentó, el algoritmo demostró una búsqueda más inteligente hacia zonas seguras.

5.3 Análisis e interpretación

Los resultados de las simulaciones confirman que la incorporación de penalizaciones asociadas a factores de riesgo en la función de costo del algoritmo A* permite priorizar rutas más seguras sin sacrificar de manera significativa la eficiencia. En todos los escenarios evaluados, el algoritmo A* modificado demostró una capacidad superior para evitar zonas de riesgo.

- Eficiencia computacional: Aunque el algoritmo A* modificado realizó un mayor número de iteraciones y tardó más tiempo en calcular las rutas, este incremento en el tiempo de cómputo es justificable dado a que la seguridad de la ruta es prioritaria, y un pequeño retraso en el cálculo de la ruta es un costo aceptable si esto implica una reducción significativa en la exposición al peligro.
- Seguridad de las rutas: La distancia mínima al fuego (D_p) fue consistentemente mayor en las rutas generadas por el algoritmo A* modificado, lo que indica una menor exposición a zonas de riesgo. En escenarios donde la propagación del fuego bloquea rutas principales, el A* modificado demostró una capacidad superior para redirigir la búsqueda hacia alternativas viables, evitando zonas peligrosas.
- Longitud del camino: En la mayoría de los escenarios, la longitud del camino (D_r) fue ligeramente mayor en las rutas generadas por el A* modificado. Sin embargo, este incremento fue moderado y se compensó con una mayor seguridad.

5.4 Conclusiones del proceso de evaluación

El proceso de evaluación permitió comprobar que el algoritmo A* modificado prioriza la seguridad de las rutas. Las principales conclusiones son las siguientes:

- **Penalización efectiva:** La penalización aplicada a los nodos cercanos a peligros activos resultó efectiva para incrementar la distancia mínima de exposición al riesgo sin deteriorar significativamente la velocidad de cálculo. Esta característica es especialmente valiosa en escenarios donde las condiciones pueden cambiar rápidamente, como en incendios forestales o terremotos.
- **Viabilidad técnica:** El proceso de validación y evaluación respalda la viabilidad técnica del algoritmo A* modificado como base para futuros sistemas de optimización de rutas seguras en contextos de emergencia.

5.5 Propuesta de mejora

Aunque los resultados obtenidos en este trabajo representan un avance significativo en el desarrollo de un prototipo funcional para la optimización de rutas de evacuación, el sistema aún se encuentra en una etapa temprana de desarrollo y no está listo para su implementación en condiciones reales. Los experimentos realizados, basados en una cuadrícula simplificada con obstáculos estáticos, demuestran la viabilidad del enfoque propuesto, pero no reflejan la complejidad de un entorno real, donde factores como la topografía tridimensional, restricciones de tránsito, direcciones de calles y condiciones dinámicas (como cambios en el flujo vehicular o la propagación de incendios) introducen desafíos adicionales que deben ser abordados.

Para avanzar hacia un producto mínimo viable que pueda ser implementado en una aplicación como Alertify, se requieren las siguientes mejoras y pasos futuros:

- **Optimización de la eficiencia computacional:** Aunque el algoritmo A* modificado es efectivo en términos de seguridad, su eficiencia computacional puede mejorarse. Una posible dirección es explorar técnicas de optimización que reduzcan el número de iteraciones necesarias sin comprometer la seguridad de las rutas. Esto podría incluir el uso de heurísticas más avanzadas o la implementación de algoritmos híbridos que combinen las ventajas de A* con otras técnicas de búsqueda.
- **Integración de datos en tiempo real:** La efectividad del sistema depende en gran medida de la calidad y actualización de los datos de riesgo. Una mejora clave sería integrar fuentes de datos en tiempo real, como sensores de incendio, sistemas de monitoreo sísmico o información meteorológica. Esto permitiría al algoritmo adaptarse a condiciones cambiantes de manera más precisa y oportuna.
- **Expansión a otros tipos de desastres:** El sistema actual está optimizado principalmente para incendios forestales. Sin embargo, su marco teórico es aplicable a otros tipos de desastres, como terremotos, tsunamis o inundaciones. Una propuesta de mejora sería adaptar el algoritmo para incorporar parámetros de riesgo específicos de estos escenarios, como la inestabilidad del terreno o la altura del agua en caso de inundaciones.
- **Adaptación a una representación de grafo de la geografía de una ciudad:** Antes de llevar el sistema a producción en Alertify, es fundamental adaptar el algoritmo a una representación de grafo que modele la geografía real de una ciudad, aunque sea a pequeña escala. Esto permitirá



evaluar su comportamiento en un entorno más cercano a la realidad, identificando posibles ajustes necesarios para garantizar su eficacia en condiciones reales.

- Implementación en plataformas móviles: Finalmente, una mejora crítica sería la implementación del algoritmo en una plataforma móvil real, como la aplicación Alertify. Esto permitiría evaluar su rendimiento en condiciones reales de uso, así como su capacidad para integrarse con otras funcionalidades de la aplicación, como notificaciones de alerta temprana o sistemas de geolocalización.

Este trabajo sienta las bases para un sistema de optimización de rutas de evacuación, pero aún se encuentra en una fase de prototipo académico. Los pasos futuros deben enfocarse en superar las limitaciones del modelo simplificado, incorporando datos reales y validando el sistema en condiciones cercanas a la realidad. Solo entonces será posible desarrollar un producto mínimo viable listo para su implementación en aplicaciones como Alertify, contribuyendo efectivamente a la gestión de emergencias y la seguridad de las comunidades.

6 Conclusiones

Este trabajo de investigación abordó el desarrollo de un sistema de optimización de rutas de evacuación seguras, integrado en una aplicación móvil para alertas de catástrofes naturales, como incendios forestales, terremotos y tsunamis. El objetivo principal fue investigar, evaluar y desarrollar un prototipo que priorice la seguridad de los usuarios en situaciones de emergencia, mediante la adaptación del algoritmo A* para incorporar parámetros de riesgo en su función de costo. El prototipo, desarrollado en Python y utilizando Pygame para la visualización, permitió simular escenarios de emergencia y evaluar el rendimiento del algoritmo A* modificado frente al algoritmo A* original.

Los resultados de las simulaciones demostraron que el algoritmo A* modificado genera rutas más seguras, aunque con un ligero incremento en la longitud del camino y el tiempo de cómputo. El algoritmo A* modificado logró una mayor distancia mínima a las zonas de peligro, reduciendo la exposición al riesgo en todos los escenarios, lo que justifica el mayor tiempo de cómputo y la longitud de ruta ligeramente mayor.

Este trabajo aporta al desarrollo de estrategias y herramientas para la gestión de emergencias, especialmente en la optimización de rutas de evacuación, ofreciendo una base sólida para futuras investigaciones. La integración de datos en tiempo real, la optimización de la eficiencia computacional y la validación con usuarios reales son áreas clave que podrían mejorar aún más la efectividad del sistema. Además, la adaptación del algoritmo para incorporar parámetros de riesgo específicos de otros tipos de desastres ampliaría su aplicabilidad y lo convertiría en una herramienta más versátil para la gestión de emergencias en diferentes contextos.

En conclusión, este trabajo ha fortalecido el conocimiento en optimización de rutas y desarrollo de sistemas de gestión de emergencias, destacando la necesidad de soluciones tecnológicas dinámicas y adaptativas, confirmando que la integración de algoritmos avanzados con datos en tiempo real representa un avance significativo en la capacidad de respuesta ante desastres, mejorando la capacidad de respuesta, la toma de decisiones y la protección de la población, contribuyendo a la protección de vidas y recursos en situaciones críticas.

7 Agradecimientos

Agradezco a Dr. Java Winchester Churrasquito por acompañarme durante todo este proceso, a Sofía Rodríguez Muñoz por responder mis dudas respecto algoritmos y Anghelo Carvajal Carvajal por ayudar en el *debugging*.

Agradezco a mis compañeros de grupo que me apoyaron en los momentos mas estresantes de este proceso, Cristian González y Camilo Leal.

8 Referencias

- [1] E. W. Dijkstra, "A note on two problems in connexion with graphs.," *Numerische mathematik* 1.1, pp. 269–271, 1959.
- [2] P. E. Hart, Nils J. Nilsson, and Bertram Raphael, "A formal basis for the heuristic determination of minimum cost paths.," *Systems Science and Cybernetics, IEEE Transactions on* 4.2, pp. 100–107, 1968.
- [3] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps. In Proceedings of the AAAI conference on artificial intelligence ,," vol. 25, no. 1, pp. 1114–1119, 2011.
- [4] Y. Zhu, G. Zhang, R. Chu, H. Xiao, Y. Yang, and X. Wu, "Research on escape route planning analysis in forest fire scenes based on the improved A* algorithm," *Ecol Indic*, vol. 166, p. 112355, Sep. 2024, doi: 10.1016/j.ecolind.2024.112355.
- [5] X. Zheng, X. Tu, and Q. Yang, "Improved JPS Algorithm Using New Jump Point for Path Planning of Mobile Robot," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, IEEE, Aug. 2019, pp. 2463–2468. doi: 10.1109/ICMA.2019.8816410.
- [6] A. Stentz, "Optimal and efficient path planning for partially-known environments. In Proceedings of the 1994 IEEE international conference on robotics and automation ,," *IEEE*, pp. 3310–3317, 1994.
- [7] N. Feng, G. Yang, W. Meng, and G. Hou, "Fusion of Improved JPS and A* Algorithms for Tactical Path Planning," in *2024 6th International Conference on Natural Language Processing (ICNLP)*, IEEE, Mar. 2024, pp. 185–191. doi: 10.1109/ICNLP60986.2024.10692902.
- [8] F. Huang and J. Wei, "Weighted Jump Point Search Algorithm in Ship Static Safe Path Finding," in *2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, IEEE, Jan. 2022, pp. 755–759. doi: 10.1109/ICCECE54139.2022.9712689.
- [9] T. Wang, R. Huang, L. Li, W. Xu, and J. Nie, "The Application of the Shortest Path Algorithm in the Evacuation System," in *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, IEEE, Sep. 2011, pp. 250–253. doi: 10.1109/ICM.2011.119.
- [10] S. Wang *et al.*, "Mapping the landscape and roadmap of geospatial artificial intelligence (GeoAI) in quantitative human geography: An extensive systematic review," *International Journal of Applied Earth Observation and Geoinformation*, vol. 128, p. 103734, Apr. 2024, doi: 10.1016/j.jag.2024.103734.
- [11] D. S. Tudose, I. Deaconu, and A. Musat, "Geo-dynamic monitoring using wireless sensor networks," in *2016 15th RoEduNet Conference: Networking in Education and Research*, IEEE, Sep. 2016, pp. 1–6. doi: 10.1109/RoEduNet.2016.7753235.
- [12] H. Soliman, K. Sudan, and A. Mishra, "A smart forest-fire early detection sensory system: Another approach of utilizing wireless sensor and neural networks," in *2010 IEEE Sensors*, IEEE, Nov. 2010, pp. 1900–1904. doi: 10.1109/ICSENS.2010.5690033.
- [13] W. Zeng, J. Ni, W. Xu, S. Du, and S. Ye, "Wildfire Risk Assessment Model for Transmission Lines Based on Matter-Element Extension Theory," in *2022 4th International Conference on Electrical*



- Engineering and Control Technologies (CEECT)*, IEEE, Dec. 2022, pp. 444–449. doi: 10.1109/CEECT55960.2022.10030688.
- [14] R. Salvi, K. Shinde, J. Jadhav, S. Ubale, and J. Katkar, “Weather Prediction and Easy Evacuation using Satellite Communication,” in *2019 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)*, IEEE, Mar. 2019, pp. 66–68. doi: 10.1109/WiSPNET45539.2019.9032760.
- [15] “<https://www.subtel.gob.cl/sae/>.”
- [16] “<https://www.shakealert.org/>.”
- [17] K. S. P. Mehlhorn, *Algorithms and Data Structures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-77978-0.
- [18] S. Ji and L. Yang, “An Extension of Dynamic Programming Algorithm in Robotic Path Planning,” in *2012 International Conference on Computer Science and Service System*, IEEE, Aug. 2012, pp. 1709–1712. doi: 10.1109/CSSS.2012.427.
- [19] A. Botea, M. Muller, and J. Schaeffer, “Near Optimal Hierarchical Path-Finding,” University of Alberta, Edmonton, Alberta, Canada, 2004.
- [20] “<https://developers.google.com/maps/documentation/routes/>.”