Inteligencia Artificial

Estado del Arte: Collission-Free Path Planning

Rodrigo Alfaro Olmos

4 de julio de 2025

Evaluación

Resumen (5 %):	_____
Introducción (5 %):	_____
Definición del Problema (10 %):	_____
Estado del Arte (35 %):	_____
Modelo Matemático (20 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100 %):	_____

Resumen

Este informe aborda el problema de *Collision-Free Path Planning*, el cual consiste en encontrar trayectorias libres de colisiones para agentes que se mueven en entornos con obstáculos. Este problema es fundamental en la robótica y otros sistemas autónomos, como vehículos inteligentes y drones. Se presentan sus principales variantes, incluyendo planificación discreta y continua, con uno o múltiples agentes. Además, se revisan diversas técnicas de resolución, desde algoritmos clásicos como A*, hasta enfoques modernos basados en metaheurísticas y aprendizaje por refuerzo. Finalmente, se incluyen experimentos con el algoritmo de *Simulated Annealing*, evidenciando su capacidad para mejorar soluciones iniciales en distintos escenarios y mostrando cómo parámetros como la temperatura mínima pueden afectar significativamente el rendimiento y la eficiencia del algoritmo. El objetivo del documento es ofrecer una visión integral del problema y de las estrategias actuales para resolverlo en escenarios reales y complejos.

1. Introducción

El *Collision-Free Path Planning* es un problema de gran relevancia para la industria robótica, ya que busca encontrar un camino libre de colisiones entre dos puntos en un entorno determinado. Esta capacidad es fundamental para el funcionamiento de robots autónomos, vehículos no tripulados y sistemas de navegacón en espacios complejos.

El propósito de este informe es presentar una descripción detallada del problema, comenzando con su definición formal, continuando con un análisis del estado del arte es decir, los principales enfoques y algoritmos utilizados hasta la fecha y finalizando con su modelamiento matemático.

La motivación detrás de este trabajo radica en la amplia aplicabilidad del problema, especialmente en campos como la robótica, donde la capacidad de planificar trayectorias seguras y eficientes es esencial para la autonomía y la eficacia de los sistemas inteligentes.

2. Definición del Problema

El problema que se aborda en este trabajo corresponde al *Collision-Free Path Planning* en un entorno bidimensional. Este problema consiste en encontrar un camino entre dos puntos dados denominados inicio y términoâ dentro de un mapa 2-D que contiene un conjunto finito de obstáculos. El objetivo es trazar una trayectoria desde el punto de inicio al de término que cumpla con ciertas condiciones esenciales.

Primero, el camino debe ser libre de colisiones, es decir, no debe intersectar ningún obstáculo presente en el entorno. Segundo, la trayectoria debe ser continua y suave, de manera que pueda ser seguida por un agente móvil, como un robot o un vehículo autónomo [3]. Bajo estas condiciones, el objetivo principal del problema es encontrar la trayectoria más corta posible que satisfaga los criterios anteriores.

Este tipo de problema ha sido abordado en distintos contextos. Por ejemplo, en [3] y [4] se plantea el problema en un espacio bidimensional, enfocándose en la eficiencia computacional y la calidad de la ruta generada. Sin embargo, también existen variantes del problema que extienden su complejidad y aplicabilidad.

Una de estas variantes considera entornos tridimensionales, como en el caso de vehículos aéreos no tripulados (UAVs), donde es necesario planificar trayectorias en 3D evitando colisiones con obstáculos volumétricos [6]. Otro caso de interés ocurre cuando los obstáculos del entorno se mueven, como en aplicaciones industriales o de logística, lo que requiere soluciones que se adapten de manera dinámica a los cambios en el entorno [9, 2].

En la figura 1 vemos la representación gráfica del problema en un mapa 2-D, en donde tenemos en rojo los puntos de inicio y termino, en azul los obstáculos y la trayectoria representada como la linea blanca.

Además, este problema está estrechamente relacionado con otros dentro del área de la robótica y la inteligencia artificial, como el problema de localización y mapeo simultáneo (SLAM[5]), la navegación autónoma y la detección de colisiones. Todos estos comparten la necesidad de representar el entorno de manera eficiente, identificar restricciones espaciales, y generar acciones seguras y óptimas en contextos complejos.

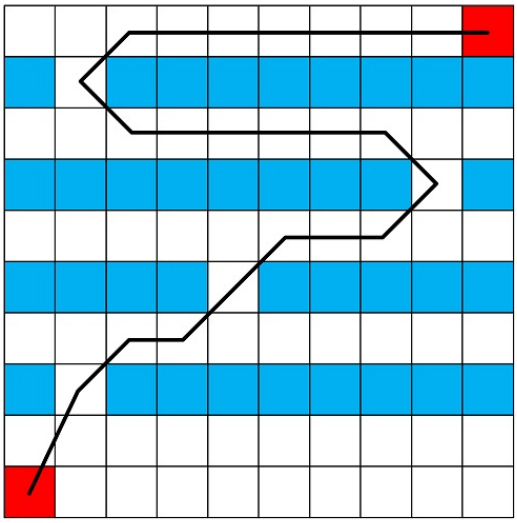


Figura 1: Representación Gráfica[8]

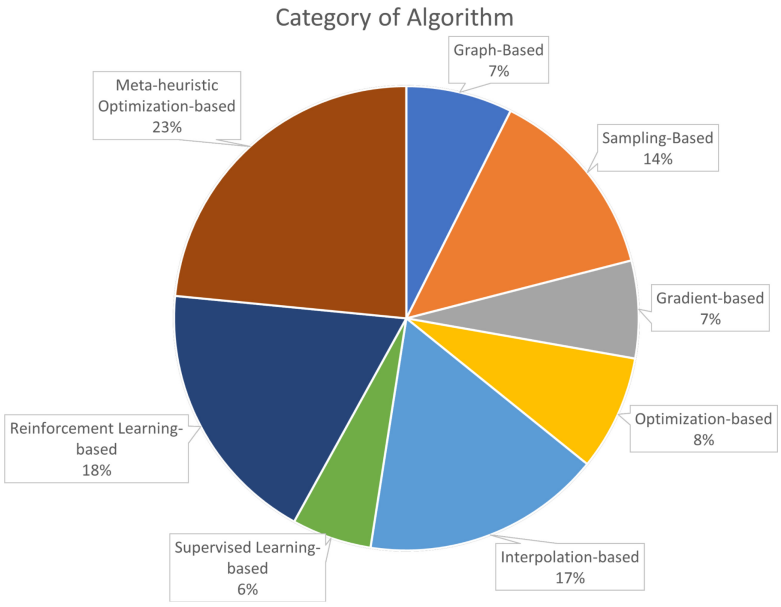
En resumen, el problema de planificación de rutas libres de colisión en entornos 2-D busca una solución que conecte dos puntos mediante una trayectoria corta, continua y suave, sin intersectar obstáculos. Sus variantes y aplicaciones son amplias, lo que lo convierte en un tema de investigación relevante tanto en la teoría como en la práctica.

3. Estado del Arte

Uno de los usos más relevantes del problema de planificación de rutas se encuentra en la conducción autónoma [5]. En este contexto, se han probado una gran variedad de algoritmos para resolver el problema, los cuales pueden agruparse en las siguientes categorías:

1. Metaheurísticas
2. Aprendizaje por refuerzo (Reinforcement Learning)
3. Aprendizaje supervisado
4. Métodos basados en interpolación
5. Métodos basados en optimización
6. Métodos basados en gradiente
7. Métodos basados en muestreo (Sampling-based)
8. Métodos basados en grafos

Entre estas, las categorías más populares han sido las metaheurísticas, los métodos basados en interpolación y el aprendizaje por refuerzo. Sin embargo, no existe una solución claramente superior a las demás en todos los contextos. Para espacios de búsqueda limitados, algoritmos clásicos como Dijkstra y A* siguen siendo los más utilizados, aunque presentan un alto costo



(e) Algorithms' categories percentage.

Figura 2: Categorización por uso de los algoritmos[5]

computacional en espacios de búsqueda más amplios.

También se han propuesto enfoques híbridos, pero, como se mencionó previamente, las metaheurísticas incluyendo algoritmos genéticos (GA), enjambre de abejas artificiales (ABC), optimización por enjambre de partículas (PSO), recocido simulado (SA), optimización por colonia de hormigas (ACO), y algoritmo de cuco (CSA) son las más empleadas en la literatura. Estas técnicas, al ser independientes del dominio del problema, permiten abordar espacios de búsqueda amplios y tienden a evitar quedarse atrapadas en óptimos locales.

Otro campo de aplicación importante es el de los vehículos aéreos no tripulados (UAV) [6]. En este ámbito, se ha propuesto el uso eficiente del algoritmo de campos potenciales artificiales (APF), dando origen a la versión extendida eAPF-CPP. Este algoritmo demostró ser efectivo al recorrer rutas en un tiempo promedio de 24.4 segundos, con una tasa de colisión del 8.56 %. Estos resultados avalan su superioridad sobre otros métodos, y se espera que futuras investigaciones lo integren con algoritmos de mapeo y localización simultánea (SLAM) para mejorar la localización de UAVs en entornos complejos.

Retrocediendo al año 2004, también se exploró el uso de algoritmos genéticos para resolver problemas de planificación de rutas con robots [4]. En este caso, el algoritmo fue levemente modificado para simplificar la representación de las variables, utilizando cromosomas de longitud fija. Las mutaciones consistían en aplicar inversiones (flips) aleatorias a los bits de los cromosomas. Se concluyó que, al emplear una representación eficiente basada en cadenas de longitud fija, el algoritmo requería menor poder de cómputo, manteniendo un rendimiento aceptable.

Este problema no se restringe únicamente al ámbito de la robótica, sino que también ha sido abordado en el sector de la construcción. En particular, se ha aplicado una solución al problema del movimiento óptimo de grúas industriales en entornos con restricciones espaciales, como bodegas o interiores con espacio reducido [9]. Aunque métodos clásicos como A* han

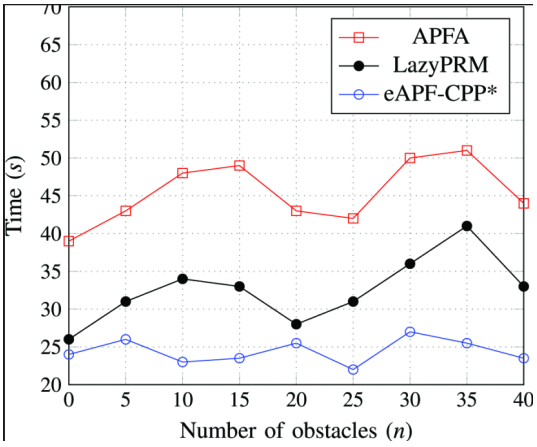


Figura 3: Numero de obstáculos vs tiempo eAPF-CPP[6]

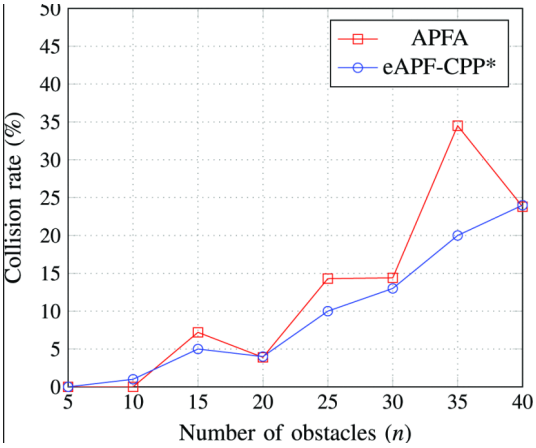


Figura 4: Numero de obstáculos vs tasa de colisión eAPF-CPP[6]

sido ampliamente utilizados, en el año 2021 se propusieron enfoques más modernos como los algoritmos genéticos y la optimización por colonia de hormigas (ACO), siendo este último el seleccionado para la implementación. Mediante simulaciones, se demostró que ACO âal incorporar variables naturales del dominioâ resuelve de forma eficiente el problema de planificación en bodegas. Se plantea como trabajo futuro la incorporación de un algoritmo adaptativo para seleccionar parámetros y un estudio más profundo sobre la influencia de los obstáculos en dichos entornos.

4. Modelo Matemático

4.1. Modelo Matemático de Asignación de Robots a Centros de Maquinado

A continuación, se presentan las notaciones y formulaciones matemáticas utilizadas para modelar la asignación de robots a tareas en centros de maquinado, minimizando costos de transporte y penalizaciones por retrasos. El siguiente modelo fue extraido de[1]

Índices

- $N = \{1, 2, ..., n\}$: Conjunto de centros de maquinado
- i, j, p: Contadores de centros de maquinado, donde $i \neq j$
- $V = \{1, 2, ..., v\}$: Conjunto de robots disponibles
- v, v': Contadores de robots, donde $v \neq v'$

Parámetros

- d_{ij}: Distancia entre los centros de maquinado i y j (metros)
- a_v: Velocidad promedio del robot v (m/s)
- g_v: Costo unitario de movimiento del robot v (\$/m)
- t_{iv}: Tiempo de llegada del robot v al centro i (segundos)
- P_l: Penalización por unidad de retraso (\$)

Variable de Decisión

- $x_{ijv} \in \{0, 1\}$: Toma el valor 1 si el robot v es asignado a la ruta de i a j, y 0 en caso contrario.
- t_{iv}: Variable de holgura que representa el tiempo de llegada del robot v al centro i.

Función Objetivo: Minimizar el costo del transporte y las penalizaciones por retraso

$$\text{mín} \sum_{i \in N} \sum_{j \in N, j \neq i} \sum_{v \in V} \left(g_v \cdot d_{ij} \cdot x_{ijv} + P_l \cdot \text{máx}(0, t_{iv} - t_{iv}^{\text{ref}}) \right) \tag{1}$$

Restricciones

$$t_{jv} \geq t_{iv} + \frac{d_{ij}}{a_v} - M(1 - x_{ijv}) \quad \text{(Restricción de intervalo de tiempo)} \quad (2)$$

$$\sum_{p \in N} x_{ipv} = \sum_{p \in N} x_{pjv} \quad \text{(Cada robot entra y sale de un centro)} \quad (3)$$

$$\sum_{j \in N} x_{0jv} = 1 \quad \text{(Robot parte del centro base)} \quad (4)$$

$$\sum_{v \in V} x_{ijv} \leq 1 \quad \text{(Sólo un robot por movimiento entre centros)} \quad (5)$$

$$\sum_{i \in N} x_{ijv} = 1 \quad \text{(Una entrada a cada centro)} \quad (6)$$

$$\sum_{j \in N} x_{ijv} = 1 \quad \text{(Una salida de cada centro)} \quad (7)$$

$$x_{ijv} \in \{0, 1\} \quad \text{(Variable binaria)} \quad (8)$$

4.2. Modelo Matemático para Planificación de Rutas en Almacenes

En este modelo, el objetivo es encontrar una ruta libre de colisiones desde la posición inicial del robot hasta un estante destino en un entorno de una bodega. El entorno puede adoptar distintos diseños: paralelos, verticales, horizontales o espina de pez. El siguiente modelo fue extraido de[3]

Función Objetivo

$$\text{máx}\, \text{Ruta Libre de Colisiones en } (W_1, W_2, W_3, W_4) = \text{Total } R.P.D \quad (9)$$

Definición de Distancia

$$\text{Total } R.P.D = \sqrt{(cx - Nxi)^2 + (cy - Nyi)^2} \quad (10)$$

Selección del Objetivo

$$gx, gy \propto \frac{1}{(Nxi, Nyi)} \quad (11)$$

$$\text{Verificar } (Nxi, Nyi) = \text{mín} \rightarrow (cx, cy) = \text{mín}(Nxi, Nyi) \quad (12)$$

$$ox \neq cx \wedge oy \neq cy \quad (13)$$

$$ox \neq Nx \wedge oy \neq Ny \quad (14)$$

$$cx = gx \wedge cy = gy \Rightarrow \text{Detener} \quad (15)$$

Definiciones

- W_1: Diseño de estanterías paralelas
- W_2: Diseño de estanterías verticales
- W_3: Diseño de estanterías horizontales
- W_4: Diseño espina de pez (fishbone)
- cx, cy: Coordenadas actuales del robot
- gx, gy: Coordenadas del objetivo (estante)
- Nxi, Nyi: Coordenadas vecinas al robot
- ox, oy: Coordenadas de obstáculos
- Nxi, Nyi: $\{(Nx_1, Ny_1), (Nx_2, Ny_2), (Nx_3, Ny_3), (Nx_4, Ny_4)\}$

5. Representación del Problema

Para la representación del problema utilizamos una matriz cuadrada, donde cada elemento corresponde a una coordenada (x, y), estableciendo un símil con un plano cartesiano discreto. Cada celda de la matriz representa un punto del espacio de búsqueda para el algoritmo de *Simulated Annealing*.

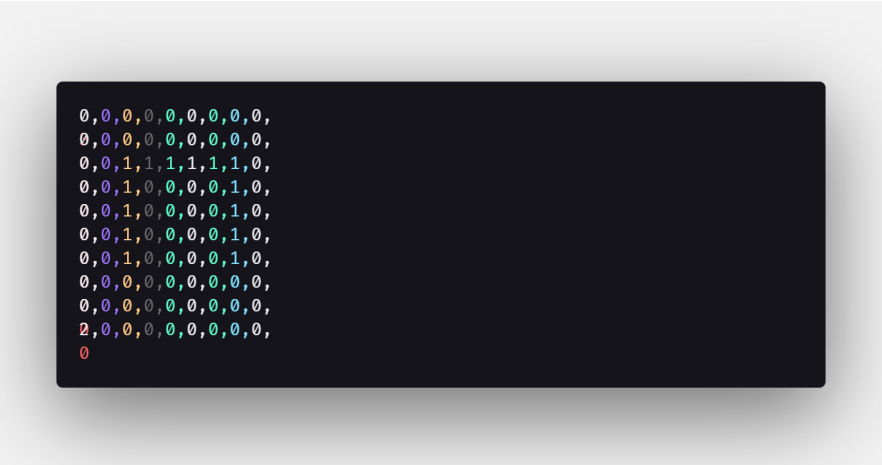


Figura 5: Ejemplo de instancia del problema, con inicio, fin y obstáculos.

En la Figura 5 se observa que cada celda puede adoptar uno de los siguientes valores:

- **0**: espacios válidos para la solución (representados como espacios blancos en el mapa 2D).
- **1**: obstáculos; celdas por las que la solución no puede pasar (espacios azules en el mapa 2D).
- **2** y **3**: punto de inicio y punto de término respectivamente (espacios rojos en el mapa 2D).

De esta manera, la matriz define completamente el entorno del problema y las restricciones que debe cumplir cualquier camino válido.

Finalmente, la solución se representa como un vector ordenado de coordenadas:

$$\text{Solución} = [(x_{\text{inicio}}, y_{\text{inicio}}), (x_2, y_2), \ldots, (x_{\text{final}}, y_{\text{final}})]\,,$$

donde el primer elemento corresponde al punto de inicio y el último elemento al punto de término. Esta representación permite evaluar la longitud del camino, validar la continuidad del mismo y garantizar que se cumplan las restricciones de paso (evitar obstáculos y mantener la conectividad entre puntos adyacentes).

6. Descripción del Algoritmo

Para resolver este problema, se nos asignó el algoritmo de *Simulated Annealing*, el cual fue implementado en C++ siguiendo un esquema clásico con los siguientes parámetros:

- **Temperatura inicial**: 100.
- **Decrecimiento de temperatura**: 5 % cada vez que se acepta una solución peor (en el código, `cooling rate` de 0.95).

- **Solución inicial**: se utiliza DFS para encontrar una solución inicial válida; en caso de que falle, se recurre a BFS como respaldo. Este enfoque cambió respecto a lo planteado en los avances, ya que la idea inicial âtrazar una línea recta entre inicio y fin con penalizaciones por chocar con obstáculosâ no arrojó resultados coherentes. A pesar de ello, se implementaron restricciones para que el movimiento respete los obstáculos.
- **Fin del algoritmo (temperatura mínima)**: cuando la temperatura es menor a 5.

Todos los parámetros se determinaron tras ejecutar el algoritmo y probar configuraciones clásicas, considerando que las instancias proporcionadas son lo suficientemente pequeñas como para que, por ejemplo, el algoritmo pudiera ejecutarse hasta una temperatura extremadamente baja (como 0.000000001). Sin embargo, se optó por parámetros tradicionales para evitar un tiempo de ejecución excesivo, ya que en un entorno real este tiempo juega un papel clave en la definición de los parámetros.

En cuanto al **movimiento**, nos basamos en un paper que emplea una versión optimizada de SA [7], por lo que nuestro movimiento consiste en perturbar un punto aleatorio de la solución (excluyendo el inicio y el fin). Esta perturbación se calcula como un desplazamiento aleatorio de -1, 0 o 1 en cada coordenada, manteniendo la restricción de que el punto modificado permanezca contiguo a sus vecinos y dentro de los límites de la matriz, logrando así un camino suave y válido.

Para **evaluar la solución**, simplemente calculamos la distancia euclidiana total entre los puntos consecutivos del camino. Como se mencionó previamente, si el camino atraviesa algún obstáculo, se aplica una penalización que invalida la solución.

Consideramos como **camino válido** aquel en el que todos los puntos de la solución estén conectados de forma contigua, permitiendo movimientos en direcciones cardinales e intercardinales (diagonales), es decir, admitimos como conexión tanto las casillas adyacentes en horizontal y vertical, como en diagonal.

En cuanto a la **ejecución del algoritmo**, este itera hasta que la temperatura sea menor a la temperatura mínima establecida, imprimiendo por pantalla información relevante sobre las iteraciones si así se indica en la firma de la función.

A nivel de implementación, todo lo descrito se encuentra modelado en la clase `SimulatedAnnealing`, que interactúa con nuestro `main` para parsear las instancias dadas en el formato previamente descrito.

7. Experimentos

Para los experimentos, se utilizaron los parámetros mencionados previamente para el algoritmo de *Simulated Annealing*, y se llevaron a cabo en un equipo con las siguientes especificaciones:

- CPU: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 6 núcleos y 12 hilos.
- GPU: NVIDIA GeForce GTX 1660 Ti (versión para portátiles).
- RAM: 16 GB DDR4.
- Almacenamiento: 512 GB SSD NVMe.
- Sistema Operativo: Ubuntu 24.04.2 LTS (a través de WSL2).
- Compilador: g++ 6.3.0.

Se ejecutaron los experimentos sobre las instancias proporcionadas, corriendo 1000 simulaciones por cada instancia. En cada ejecución se midieron la diferencia entre la evaluación de la solución inicial y la solución final encontrada, calculando la media de estas diferencias, así como el tiempo promedio de ejecución por instancia.

Los resultados se almacenaron en un archivo de texto y se procesaron posteriormente utilizando un *Jupyter Notebook*.

Adicionalmente, se repitió este procedimiento modificando el parámetro de la temperatura mínima a un valor de 0.001, con el objetivo de analizar cómo este cambio afecta el comportamiento y desempeño del algoritmo.

Ambos resultados de los simulaciones se guardaron en archivos .txt adjuntados con el codigo de la solución

8. Resultados

8.1. Diferencia entre soluciones - Δ

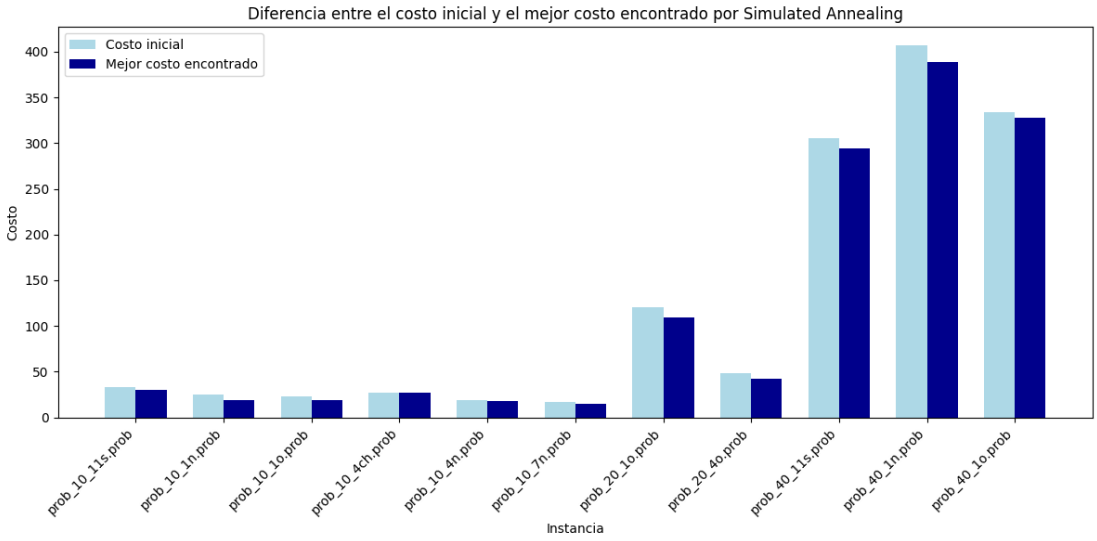


Figura 6: Diferencia entre soluciones

A priori, observamos que, si bien el algoritmo ayuda a encontrar un camino de menor costo, la diferencia no es tan pronunciada, especialmente en las instancias más pequeñas. Incluso, en una de ellas (10_4ch) no se logró mejorar la solución inicial. Aun así, aunque sea una mejora pequeña, en la mayoría de los casos se logra reducir el costo de la solución inicial.

Como mencionamos anteriormente, al utilizar una temperatura mínima más baja, se obtienen mejores resultados, como se muestra a continuación:

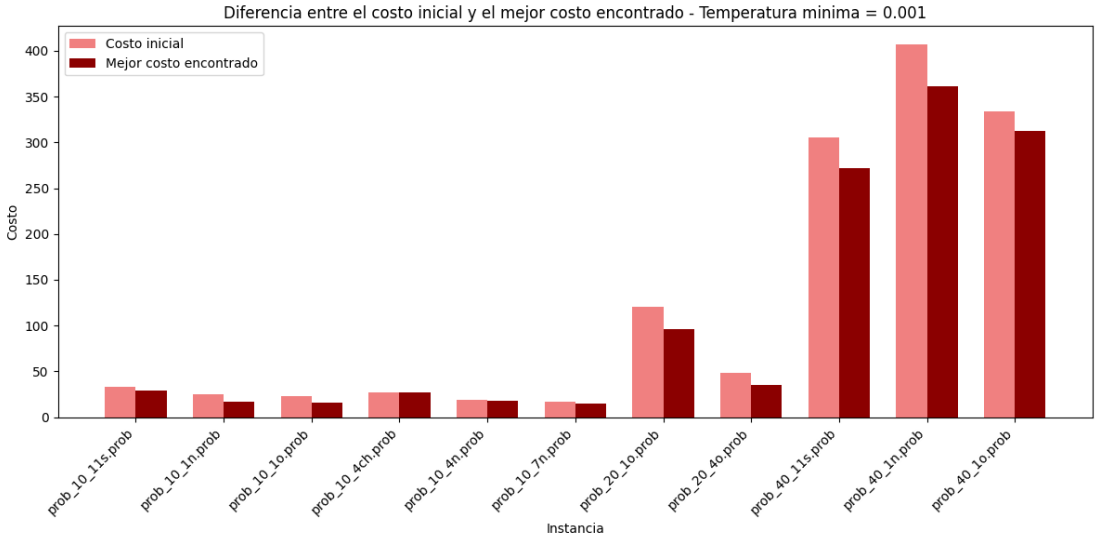


Figura 7: Diferencia entre soluciones - Temperatura mínima 0.001

8.2. Comparación entre diferencia de soluciones - Δ y porcentaje de mejora

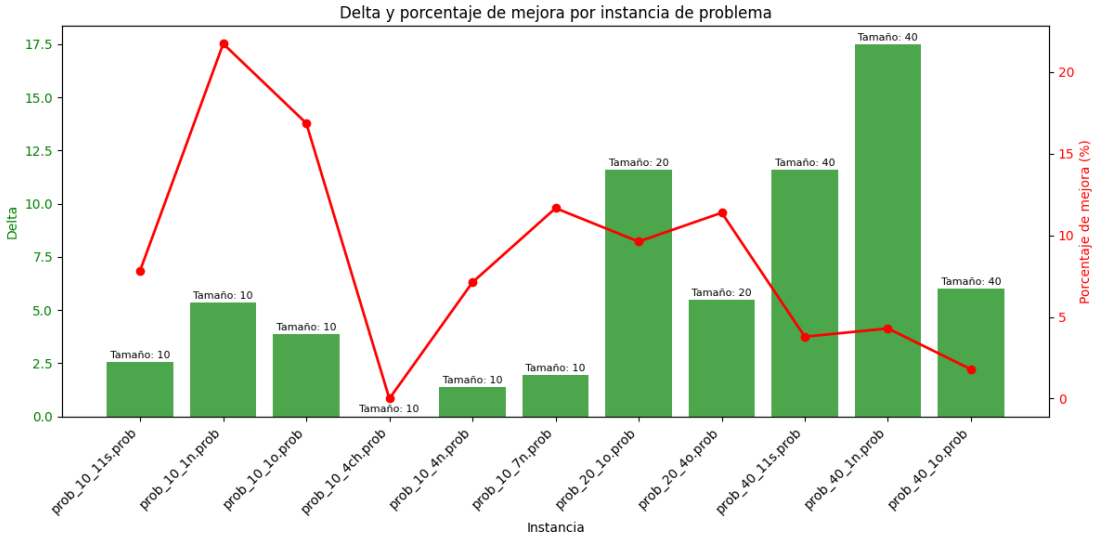


Figura 8: Comparación entre Δ y porcentaje de mejora

En cuanto al porcentaje de mejora (calculado como (Δ/costo inicial) × 100), observamos que el comportamiento se agrupa según el tamaño de la instancia: mientras mayor es la instancia, menor es el porcentaje de mejora promedio. Esto indica que, a medida que aumenta la complejidad del problema, el algoritmo tiene mayor dificultad para lograr mejoras significativas.

8.3. Tiempos de ejecución

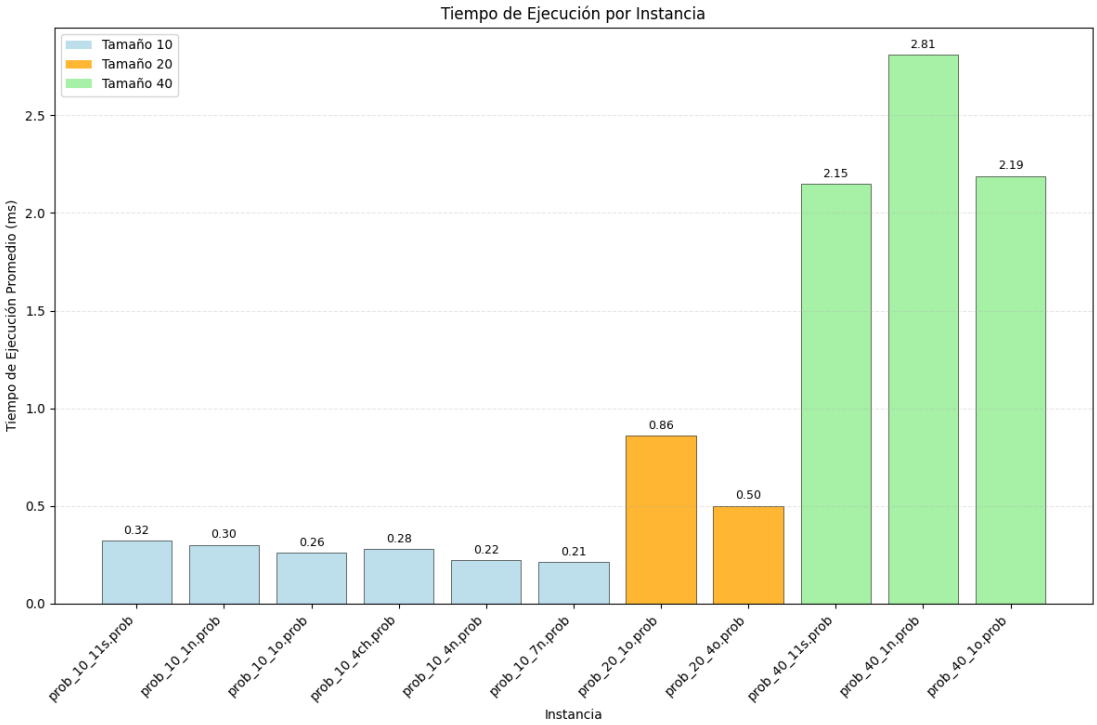


Figura 9: Tiempos de ejecución

Como se mencionó anteriormente, los tiempos de ejecución son relativamente bajos, ya que la generación de los resultados con 1000 simulaciones por instancia solo requirió un par de minutos. Si analizamos los tiempos de ejecución en detalle, vemos lo esperado: tiempos similares para instancias de igual tamaño, que aumentan conforme crece la instancia.

Un punto interesante que se observa al comparar con los tiempos obtenidos con una temperatura mínima más baja, es que existe una diferencia significativa. Esto era de esperar, ya que una temperatura mínima menor implica un mayor número de iteraciones, aumentando considerablemente el tiempo de ejecución. Este hallazgo es clave al pensar en un problema real, ya que evidencia que una elección incorrecta de los parámetros puede hacer que el algoritmo pierda una de las principales ventajas de las metaheurísticas: encontrar soluciones de buena calidad en un tiempo acotado.

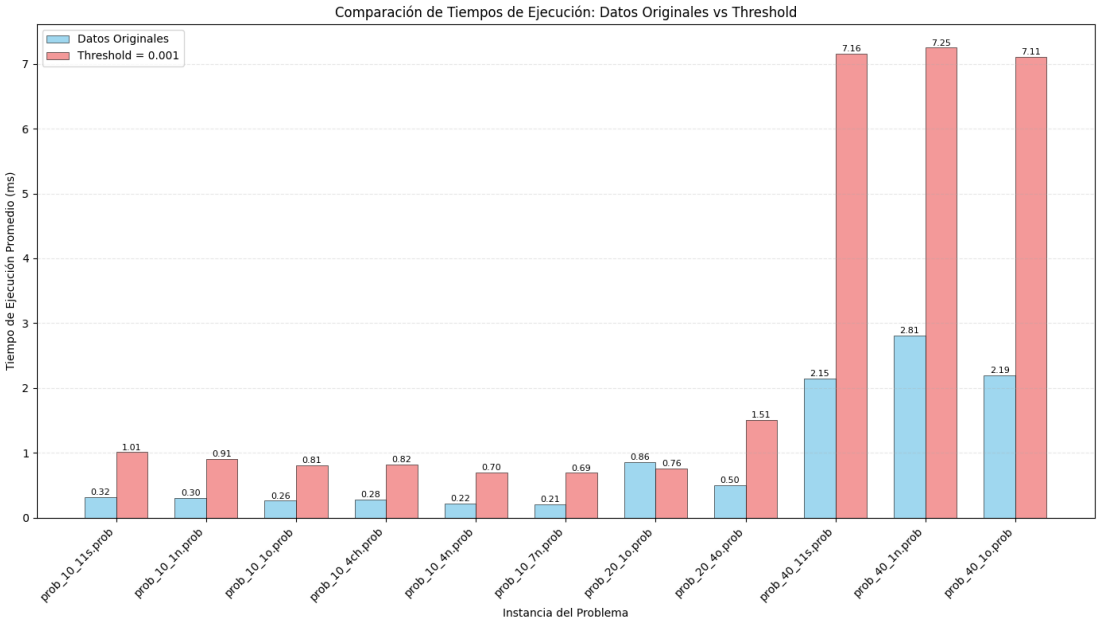


Figura 10: Tiempos de ejecución - Originales vs Temperatura mínima 0.001

A partir de los experimentos realizados, observamos que el algoritmo de SA encuentra una mejor solución para casi todas las instancias (excepto una), logrando mejoras que varían entre un 2 % y un 20 % en comparación con la solución inicial (obtenida mediante DFS o BFS). Además, se evidenció que, en nuestro caso, al aumentar el tamaño de las instancias, disminuye el porcentaje de mejora. También se comprobó que los tiempos de ejecución fueron lo suficientemente bajos como para realizar 1000 simulaciones por instancia en pocos minutos; sin embargo, también se demostró que, en un escenario real, una modificación en los parámetros (como la temperatura mínima) podría llevar a diferencias drásticas en el tiempo de ejecución, afectando la viabilidad del algoritmo como metaheurística eficiente.

9. Conclusiones

El problema de *Collision-Free Path Planning* representa un desafío fundamental en el campo de la inteligencia artificial y la robótica, debido a su aplicabilidad directa en sistemas autónomos como vehículos, robots móviles y UAVs. Asi como tambien en ámbitos fuera de la robótica, como se vio en el estado del arte con las grúas para bodegas industriales[9], demostrando la diversidad de enfoques existentes en donde nos podemos encontrar con el problema.

Se concluye que no existe un método único que supere al resto en todos los contextos, en cambio, la elección del algoritmo depende en gran medida del tipo de entorno, las restricciones computacionales y los requisitos de la aplicación. Mientras que algoritmos clásicos como A* y Dijkstra siguen siendo relevantes en entornos acotados y espacios de búsqueda no muy extensos, las técnicas metaheurísticas y de aprendizaje[5] se han consolidado como alternativas robustas en escenarios más complejos y dinámicos.

El análisis también permite observar una tendencia hacia la combinación de métodos (enfoques híbridos)[5], lo que permite aprovechar las fortalezas individuales de diferentes técnicas. Por último, se destaca la creciente relevancia de estos sistemas en aplicaciones prácticas, como lo es

la conducción automática en automóviles, lo que motiva la necesidad de continuar investigando y optimizando soluciones para el trazado de trayectorias libres de colisiones.

A partir de los experimentos realizados con el algoritmo de *Simulated Annealing*, se concluye que este método es capaz de mejorar la solución inicial obtenida mediante búsquedas como DFS o BFS en la mayoría de los casos, logrando reducciones en el costo entre un 2 % y un 20 % dependiendo del tamaño de la instancia. Se observó que el porcentaje de mejora tiende a disminuir a medida que crece el tamaño del problema, mientras que el tiempo de ejecución permanece bajo para configuraciones adecuadas de temperatura mínima. Sin embargo, una selección incorrecta de parámetros puede incrementar significativamente los tiempos de cómputo, lo que subraya la importancia de un ajuste cuidadoso para mantener la eficiencia de la metaheurística en aplicaciones reales.

10. Bibliografía

Referencias

[1] Fazlollahtabar Hamed and. An effective mathematical programming model for production of automatic robot path planning. *The Open Transportation Journal*, 13, 2019/03/26.

[2] Yanbo Kong, Yueyang Li, and Meng Li. Sequential path planning of multi food delivery robot to avoid collision. In *2024 IEEE 4th International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, volume 4, pages 994–998, 2024.

[3] N. Vimal Kumar and C. Selva Kumar. Development of collision free path planning algorithm for warehouse mobile robot. *Procedia Computer Science*, 133:456–463, 2018. International Conference on Robotics and Smart Manufacturing (RoSMa2018).

[4] G. Nagib and W. Gharieb. Path planning for a mobile robot using genetic algorithms. In *International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04.*, pages 185–189, 2004.

[5] Mohamed Reda, Ahmed Onsy, Amira Y. Haikal, and Ali Ghanbari. Path planning algorithms in the autonomous driving system: A comprehensive review. *Robotics and Autonomous Systems*, 174:104630, 2024.

[6] Praveen Kumar Selvam, Gunasekaran Raja, Vasantharaj Rajagopal, Kapal Dev, and Sebastian Knorr. Collision-free path planning for uavs using efficient artificial potential field algorithm. In *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, pages 1–5, 2021.

[7] Kun Shi, Luyao Yang, Zhengtian Wu, Baoping Jiang, and Qing Gao. Multi-robot dynamic path planning with priority based on simulated annealing. *Journal of the Franklin Institute*, 362(1):107396, 2025.

[8] Hyunwoo Shin and Junjae Chae. A performance review of collision-free path planning algorithms. *Electronics*, 9(2), 2020.

[9] Jatuporn Waikoonvet, Nattapong Suksabai, and Ittichote Chuckpaiwong. Collision-free path planning for overhead crane system using modified ant colony algorithm. In *2021 9th International Electrical Engineering Congress (iEECON)*, pages 325–328, 2021.