Optimización de trayectorias en entornos discretos mediante autómatas celulares y el algoritmo A*

C. Acosta[1,*], **G. Carrillo**[1], **I. Martín**[1], **G. Rivadeneira**[1]

[1] *Facultad de Ingeniería de la Universidad Autónoma de Yucatán*

Fecha de recepción: 4 de enero de 2026 — Fecha de aceptación: 2 de septiembre de 2026

Resumen

En el presente trabajo se implementa el algoritmo de búsqueda heurística A-Star (A*) para la determinación de la ruta óptima en entornos de navegación tipo laberinto con condiciones de contorno (inicio y fin) predefinidas. El objetivo principal consiste en establecer la trayectoria de longitud mínima que conecte ambos puntos, garantizando la evasión estricta de los obstáculos presentes en el dominio. Se presentan tres casos de estudio con niveles de complejidad incremental, donde se demuestra que el núcleo algorítmico del método A* mantiene su consistencia y eficacia frente a diversas restricciones espaciales. Asimismo, se propone una metodología de implementación basada en la integración de marcos de datos (data frames) cuyos índices estén asociados a las filas y columnas de matrices construidas para este propósito, una estructura que optimiza la gestión de la información y mejora la eficiencia computacional del proceso de búsqueda.

Palabras Clave: Algoritmo A*, optimización de rutas, planificación de trayectorias, marcos de datos, heurística de búsqueda.

Trajectory optimization in discrete environments using cellular automata and the A* algorithm

Abstract

This paper implements the A-Star (A*) heuristic search algorithm to determine the optimal route in maze-like navigation environments with predefined boundary conditions (start and finish). The main objective is to establish the shortest path connecting both points, guaranteeing strict avoidance of obstacles present in the domain. Three case studies with increasing levels of complexity are presented, demonstrating that the algorithmic core of the A* method maintains its consistency and effectiveness under various spatial constraints. Furthermore, an implementation methodology is proposed based on the integration of data frames whose indices are associated with the rows and columns of matrices constructed for this purpose, a structure that optimizes information management and improves the computational efficiency of the search process.

Keywords: A* algorithm, route optimization, path planning, data frames, search heuristics.

1. Introducción

Un autómata celular (AC) se define como un modelo matemático aplicado a un sistema dinámico discreto, constituido por un arreglo regular de elementos denominados celdas. Cada unidad (celda o nodo) posee un estado finito dentro de un conjunto de valores posibles, los cuales evolucionan en intervalos de tiempo discretos mediante la aplicación de reglas de transición locales. El estado futuro de una celda individual está condicionado por su configuración actual y la de su vecindad inmediata (comúnmente bajo las configuraciones de Moore o Von Neumann). De este modo, un AC permite modelar comportamientos macroscópicos complejos a partir

*crenan@correo.uady.mx

Nota: Este artículo de investigación es parte de Ingeniería–Revista Académica de la Facultad de Ingeniería, Universidad Autónoma de Yucatán, Vol. 30, No. 2, 2026, ISSN: 2448-8364

de interacciones locales simplificadas. En términos operativos, el AC actúa como un procesador donde el estado de una celda y sus primeros vecinos ingresan con ciertos parámetros de entrada, se somete a las reglas de transición y resulta en una actualización de estado sincronizada para todo el conjunto. todas las celdas del arreglo se actualizan al mismo tiempo en cada paso discreto, cada una con un resultado particularizado [1, 2].

En el ámbito de la optimización de rutas entre dos puntos geográficos o nodos, la base fundamental fue establecida por E. W. Dijkstra (1959) [3]. Su planteamiento original abordó la resolución computacional de la ruta mínima en una red de nodos interconectados.

Posteriormente, este enfoque evolucionó hacia el algoritmo A*, propuesto en 1968 por P. Hart, J. Nilsson y B. Raphael [4] a diferencia de Dijkstra, el algoritmo A* optimiza la búsqueda mediante una función de evaluación que integra dos componentes de costo:

- **Costo real del grafo** $g(n)$: Representa la distancia o el gasto acumulado desde el nodo inicial hasta el nodo actual n.
- **Costo heurístico** $h(n)$: Estima la distancia mínima esperada desde el nodo hasta el objetivo. Es relevante notar que $h(n)$ es una estimación, ya que se calcula ignorando los obstáculos o paredes presentes en el dominio de búsqueda.

La suma de ambos parámetros define la función de aptitud total $f(n)$ (fig. 1), la cual permite priorizar la exploración de las celdas vecinas con menor costo proyectado.

Investigaciones recientes han explorado heurísticas alternativas inspiradas en sistemas biológicos. Por ejemplo, se ha utilizado un índice de quimiotaxis (denominado Smell Index) para simular el comportamiento del moho mucilaginoso *Physarum polycephalum* [5–7]. Este índice permite al organismo optimizar su gasto energético y la ubicación de sus extensiones para localizar nutrientes en entornos con obstáculos, resolviendo de manera indirecta problemas de transporte multiobjetivo. En el presente trabajo, las funciones $g(n)$ y $h(n)$ se determinan mediante la métrica de Manhattan (o distancia del taxista) [8–10], la cual se define matemáticamente como:

$$L\left(P_1, P_2\right) = |x_2 - x_1| + |y_2 - y_1| \tag{1}$$

En donde L (ec. 1) es la distancia entre los puntos evaluados P_1 y P_2 con coordenadas (x_1, y_1) y (x_2, y_2). La implementación de esta métrica permite operar exclusivamente con números enteros (si las coordenadas cumplen esta condición), optimizando el procesamiento computacional sin introducir errores metodológicos significativos en comparación con el uso de la distancia euclidiana tradicional.

Los cálculos realizados se basaron en un software desarrollado en Java que se implementó en una computadora MacBook pro M3 Max de 16 núcleos. Los laberintos utilizados son comunes en la literatura.

2. Descripción del método de búsqueda A*

El algoritmo de búsqueda A* [11–13] comienza con la definición de los nodos de origen (I) y destino (O), los cuales se sitúan dentro de un entorno discretizado. En este trabajo, se propone una **discretización matricial** del espacio en filas y columnas (fig. 2). Esta organización permite tratar el mapa o laberinto como un sistema de coordenadas cartesianas, optimizando el acceso indexado a cada celda y facilitando el cálculo de vecindades, independientemente de la geometría celular elegida (cuadrangular, hexagonal, triangular o circular).

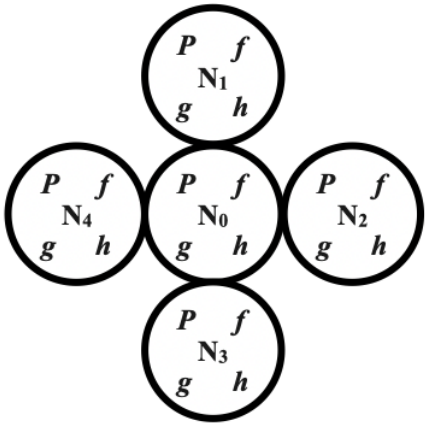


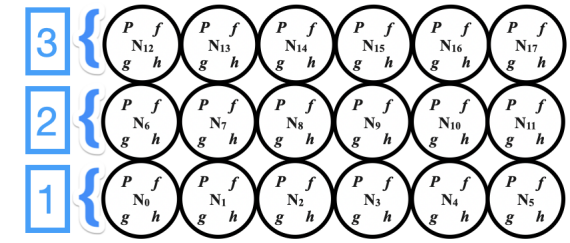
Figura 1: Los valores que toma N_0, N_1,..., dependen de la celda inicial, así si $N_0 = 00$, es decir la celda inicial, todas la demás siguen esta numeración y P en todas sus celdas vecinas es cero. Los valores de los parámetros g, h y f dependen de la distancia, para la celda inicial $g = 0$. El orden de recorrido mostrado es antihorario, pero también se puede utilizar un recorrido horario.

Cada celda se define no solo por su índice y por su posición, sino por un conjunto de atributos de estado que incluyen los costos g, h y f, además del puntero de precedencia (P). Este último elemento es crítico para la reconstrucción de la trayectoria óptima (backtracking) una vez que se alcanza el nodo objetivo.

Para la gestión eficiente de la información, se implementa un marco de datos (data frame) (fig. 2), que actúa como el registro central del autómata celular. La estructura de esta tabla se organiza de la siguiente manera:

- en una primera columna (0), el indice consecutivo de todas las celdas que conforman el espacio.
- en las columnas 1 y 2 (si el laberinto o mapa es bidimensional) las coordenadas de la celda.

- en la columna 3 pondremos el conjunto de los vecinos de la célula.
- en la columna 4, se marca el tipo de celda de que se trate, si es un obstáculo o pared (E) o celda inicial (I) o final (O) o nodo (N) propia para el proceso.
- en las columnas 5, 6, 7 y 8 los parámetros g, h y f además de la precedencia P.



Marco de datos

index	Posx	Posy	VecRel	tCell	gVal	hVal	fVal	nPred
0	0.0	0.0	{6,1,0,0}	N	0.0	0	0	0
1	100.0	0.0	{7,2,0,0}	N	0.0	0	0	0
2	200.0	0.0	{8,3,0,1}	N	0.0	0	0	0
3	300.0	0.0	{9,4,0,2}	N	0.0	0	0	0
4	400.0	0.0	{10,5,0,3}	N	0.0	0	0	0
5	500.0	0.0	{11,0,0,4}	N	0.0	0	0	0
6	600.0	0.0	{12,7,0,0}	N	0.0	0	0	0
7	700.0	0.0	{13,8,1,6}	N	0.0	0	0	0
8	800.0	0.0	{14,9,2,7}	N	0.0	0	0	0
9	900.0	0.0	{15,10,3,8}	N	0.0	0	0	0
10	0.0	100.0	{16,11,4,9}	N	0.0	0	0	0
11	100.0	100.0	{17,0,5,10}	N	0.0	0	0	0
12	200.0	100.0	{0,13,6,0}	N	0.0	0	0	0
13	300.0	100.0	{0,14,7,12}	N	0.0	0	0	0
14	400.0	100.0	{0,15,8,13}	N	0.0	0	0	0
15	500.0	100.0	{0,16,9,14}	N	0.0	0	0	0
16	600.0	100.0	{0,17,10,15}	N	0.0	0	0	0
17	700.0	100.0	{0,0,11,16}	N	0.0	0	0	0

Figura 2: Suponiendo que los primeros vecinos son cuatro y que están numerados como se muestra en las filas 1, 2 y 3, de la matriz, estas se convierten en columnas de la tabla, con el conjunto de primeros vecinos en la columna 3, el tipo de celda en la columna 4 (todas tipo nodo N en esta descripción) y los valores de g, h, y f, así como el nodo precedente en las columnas 5, 6, 7 y 8, ninguno de los cuales han sido evaluados, ya que necesitamos a las celdas inicial y final del laberinto o mapa.

El número de vecinos depende de como se conceptualice el espacio, ya que puede ser con celdas cuadradas, hexagonales o como en este caso circulares o aún triangulares.

2.1. Dinámica de búsqueda y estructuras de operación

El proceso de optimización se apoya en tres estructuras dinámicas que gestionan la frontera de exploración:

1. **openSet (Lista Abierta)**: Un subconjunto dinámico del marco de datos que almacena los nodos candidatos a ser explorados, así como sus variables de estado y precedencia. Se comporta como una cola de prioridad basada en el valor mínimo de f.
2. **closedSet (Lista Cerrada)**: Un arreglo (tabla de una sola columna) que contiene los índices de los nodos ya evaluados que forman parte de la exploración activa, evitando ciclos y redundancias.
3. **predSet**: Una lista operativa (tabla de una sola columna) dedicada exclusivamente a almacenar la secuencia de precedencias para derivar la ruta de mínimo costo.

2.2. Algoritmo de Ejecución

El procedimiento iterativo se describe a continuación:

- **Inicialización**: Se identifica la celda I y se añaden sus vecinos inmediatos al openSet. Para cada vecino, se calcula g (costo desde I hasta la celda vecina más la propia g de I), h (estimación al objetivo o celda final) y f, asignando a P como su predecesor. En donde la función de aptitud total f (ec. 2) es:

$$f(n) = h(n) + g(n) \tag{2}$$

- **Ciclo de Optimización**:
  a) Se selecciona del openSet el nodo con el **valor mínimo de** f (ec. 2).
  b) Dicho nodo se transfiere del openSet al closedSet.
  c) Se evalúan los vecinos del nodo seleccionado. Si un vecino no está en ninguna lista, se calculan sus costos y se añade al openSet.
  d) **Criterio de Actualización**: Si un vecino ya reside en el openSet o closedSet, se verifica si la nueva ruta ofrece un valor de g inferior. De ser así, se actualizan sus parámetros y su puntero de precedencia P. Así, el estado del AC se actualiza en función del menor valor, elegido entre los elemento del conjunto openSet, de la función de aptitud total f_m = *aptitud total mínima* (ec. 3), en términos de una función de estado dada por:

$$S_{t+1} = f_m\left(S_t, n\right) \tag{3}$$

- **Finalización**: El proceso se repite recursivamente hasta que en uno de los nodo se tenga el valor de $h = 0$ en el openSet, lo que coincide con el nodo objetivo O, momento en el cual se reconstruye la ruta óptima mediante la trazabilidad de los predecesores.

3. Implementación: Casos de estudio

La validación del algoritmo se realiza a través de tres aplicaciones con complejidad ascendente. Los dos primeros casos se resuelven mediante un análisis comparativo entre el cálculo manual y la ejecución computacional, mientras que el tercero se reserva exclusivamente para la evaluación del rendimiento del software diseñado.

3.1. Aplicación 1: Optimización con grafos dirigidos en un arreglo de 12 Nodos

El primer escenario consiste en un entorno discretizado con 12 nodos de interés, distribuidos en un plano de coordenadas cartesianas. El objetivo es determinar la trayectoria de costo mínimo entre el nodo de origen **S** ($ID : 66$) y el nodo de destino **K** ($ID : 4$) (fig. 3 a y b).

Estructura de datos y conectividad

Para la gestión de este escenario, se emplea el marco de datos denominado daFra (fig. 4 a), el cual integra la topología del entorno y las variables de estado, así como las precedencias:

- **Relaciones de adyacencia**: A diferencia de una rejilla abierta, este mapa utiliza aristas dirigidas. Por ejemplo, existe una transición permitida de $S \rightarrow A$, pero no una relación inversa, lo que simula restricciones de flujo o sentido en la ruta.
- **Gestión de celdas nulas**: El espacio total contiene celdas marcadas como Z (sin relaciones de vecindad). Aunque estas no participan activamente en la búsqueda, se mantienen en la estructura matricial para preservar el orden espacial y la integridad de la indexación, funcionando como un entorno de "relleno" o padding.

Resultados de la Simulación

Para optimizar el procesamiento, se filtraron las filas correspondientes a las celdas Z, centrando la operación del algoritmo en los 12 nodos interconectados.

- **Evaluación heurística**: El algoritmo procesó las vecindades (máximo 3 vecinos por nodo) evaluando las distancias con la métrica Manhattan (ec. 1) hacia el objetivo K.
- **Ruta Óptima**: El sistema identificó con éxito la secuencia de nodos que minimiza la función $f(n)$. En el análisis de resultados, la ruta resultante se resalta con una línea roja en el figura 4b), confirmando la coincidencia entre el cálculo manual y la salida del algoritmo computacional.

Análisis del marco de datos

La estructura del daFra permite observar cómo cada iteración actualiza los parámetros de los vecinos. En la visualización de los resultados (fig. 4 b), se observa que la ruta roja no solo evita el costo innecesario, sino que respeta estrictamente la direccionalidad impuesta por los vectores de vecindad, validando la robustez del modelo matricial propuesto.

3.2. Aplicación 2: Optimización de la ruta en un laberinto con dos paredes internas

El segundo escenario de estudio consiste en un dominio rectangular discretizado en una topología de **9 filas y 10 columnas** (90 celdas en total). En este diseño, los límites perimetrales y dos estructuras internas actúan como barreras infranqueables (paredes), integradas directamente como elementos restrictivos dentro de la matriz de estados.

Configuración de la vecindad y parámetros

A diferencia del primer ejemplo, este modelo utiliza una **vecindad de Moore (8 vecinos)** [1] para cada celda cuadrangular, permitiendo desplazamientos, del nodo a sus vecinos, tanto ortogonales como diagonales. Se elige una vecindad de 8 elementos a una celda cuadrada para ajustarse a las reglas de las estructuras cristalinas, en donde los vecinos se definen como los átomos, iones o moléculas más cercanos que rodean a un átomo o partícula central [14, 15].

- **Puntos de control**: Se definió la celda 22 como el nodo de origen (I) y la celda 62 como el nodo objetivo (O) (fig. 5 a y b).
- **Escalamiento de costos**: En la implementación computacional, los valores de las funciones g, h y f presentan un factor de escala de 100 respecto a las unidades métricas mostradas en la (fig. 5 a y b). Este ajuste se realizó para optimizar la representación visual y la precisión en la interfaz gráfica, sin alterar la proporcionalidad de la ruta óptima.

Análisis de Resultados y Validación

El proceso iterativo fue ejecutado de manera dual: mediante un cálculo manual exhaustivo y a través del algoritmo codificado.

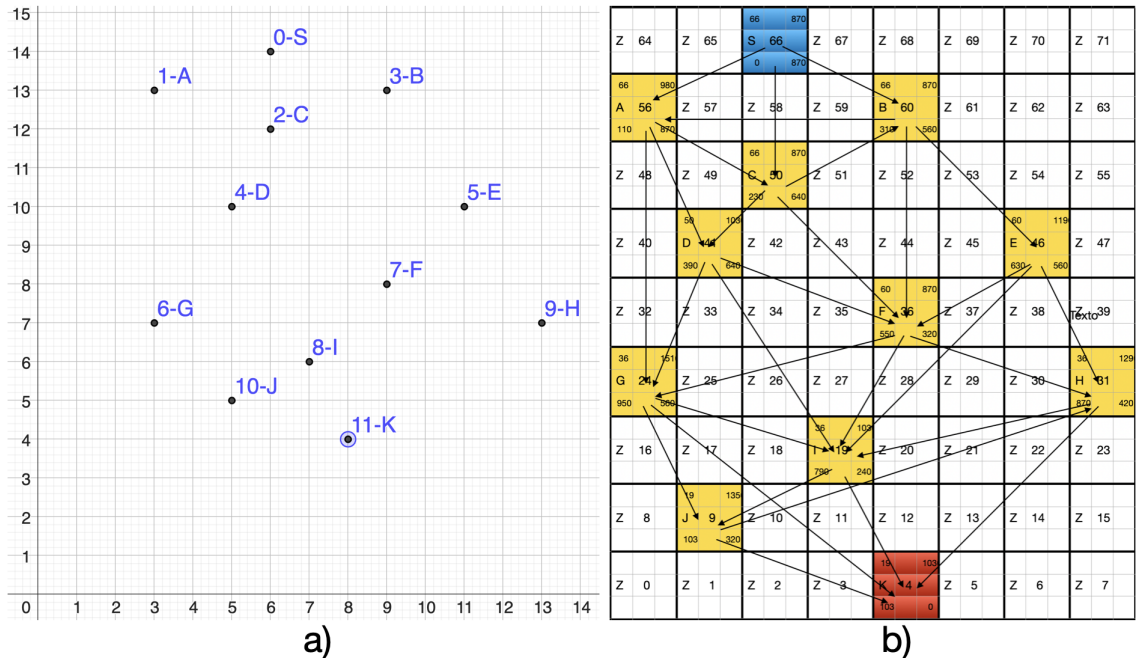
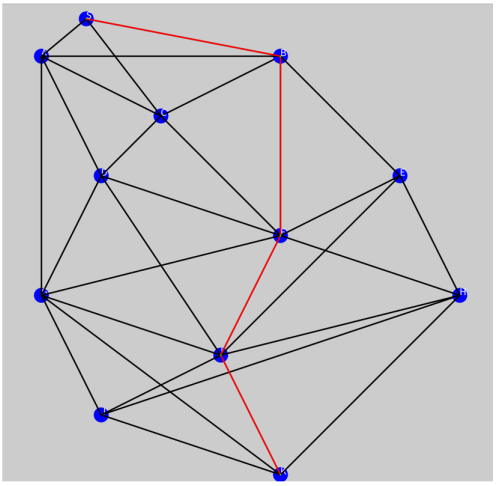


Figura 3: En a) se muestra la descripción espacial en coordenadas cartesianas con una numeración consecutiva y en b) se muestra la asociación entre índices de celdas con filas y columnas de una matriz asociada.

index	Posx	Posy	VecRel	tCell	gVal	hVal	fVal	nPred
4	410.0	10.0	{0,0,0}	K	1030.0	0.0	1030.0	19
9	170.0	90.0	{31, 4, 0}	J	1030.0	320.0	1350.0	19
19	330.0	170.0	{9, 4, 0}	I	790.0	240.0	1030.0	36
24	90.0	250.0	{4, 9, 19}	G	950.0	560.0	1510.0	36
31	650.0	250.0	{19, 4, 0}	H	870.0	480.0	1350.0	36
36	410.0	330.0	{24, 19, 31}	F	550.0	320.0	870.0	60
41	170.0	410.0	{24, 19, 36}	D	390.0	640.0	1030.0	50
46	570.0	410.0	{36, 19, 31}	E	630.0	560.0	1190.0	60
50	250.0	490.0	{41, 36, 60}	C	230.0	640.0	870.0	66
56	90.0	570.0	{24, 41, 50}	A	110.0	880.0	990.0	66
60	410.0	570.0	{56, 36, 46}	B	310.0	560.0	870.0	66
66	150.0	620.0	{56, 50, 60}	S	0.0	870.0	870.0	66

a)



b)

Figura 4: En a) se muestra la estructura del marco de datos, en donde se han eliminado las filas correspondientes a las celdas marcadas con Z que no están relacionadas con ninguna otra, solo se dejaron las 12 que si se asocian entre ellas. En b) se tiene el resultado marcado en rojo de la ruta óptima entre los puntos **S** y **K**.

- **Consistencia de Datos**: El marco de datos resultante (daFra) gestionó las 90 filas del espacio muestral. En la (fig. 6 a) se observa una sección representativa (filas 11 a 24) donde se registran las transiciones de estado y la asignación de precedencias (P).
- **Convergencia**: El algoritmo A* logró sortear las paredes internas, identificando la trayectoria más eficiente (Fig. 5b y 6b).
- **Correlación**: Se determinó una coincidencia exacta entre los resultados del procesamiento manual y el computarizado, lo que valida la fiabilidad de la lógica de programación y la correcta gestión de las listas openSet y closedSet en entornos con obstáculos físicos.

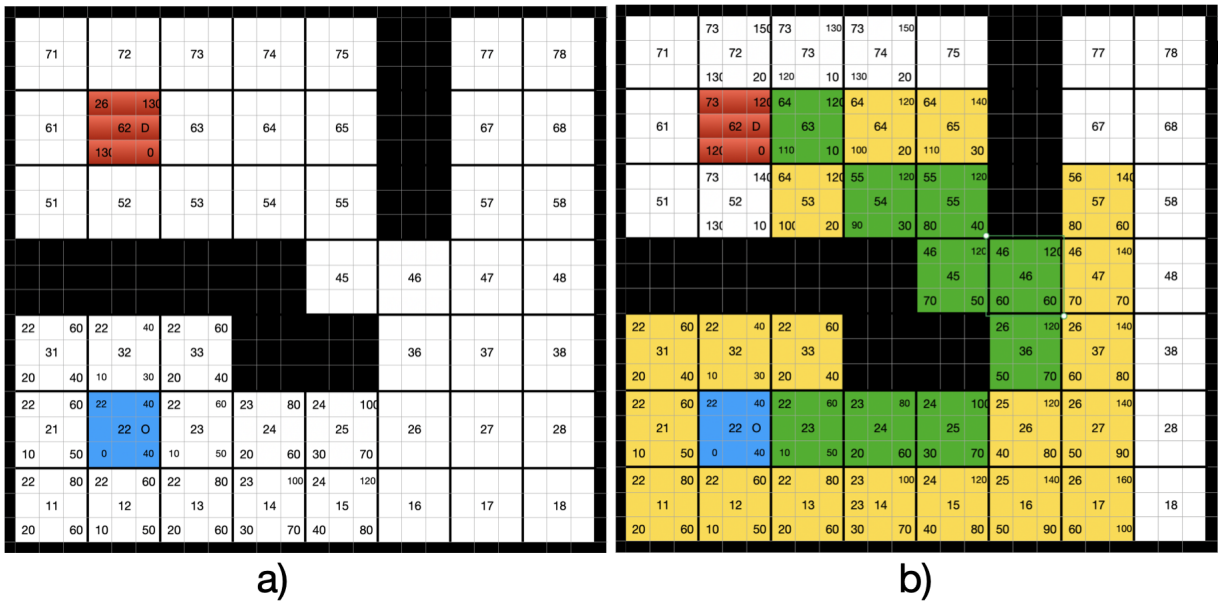


Figura 5: En a) define a la celda inicial O (ID : 22) y la celda final D (ID : 62). En b) se tiene la ruta óptima, coloreada en verde, definida por el proceso seguido en A*.

index	Posx	Posy	VecRel	tCell	gVal	hVal	fVal	nPred
11	100.0	100.0	{21,22,12,2,1,0,10,20}	N	200.0	600.0	800.0	22
12	200.0	100.0	{22,23,13,3,2,1,11,21}	N	100.0	500.0	600.0	22
13	300.0	100.0	{23,24,14,4,3,2,12,22}	N	200.0	600.0	800.0	22
14	400.0	100.0	{24,25,15,5,4,3,13,23}	N	300.0	700.0	1000.0	23
15	500.0	100.0	{25,26,16,6,5,4,14,24}	N	400.0	800.0	1200.0	24
16	600.0	100.0	{26,27,17,7,6,5,15,25}	N	500.0	900.0	1400.0	25
17	700.0	100.0	{27,28,18,8,7,6,16,26}	N	600.0	1000.0	1600.0	26
18	800.0	100.0	{28,29,19,9,8,7,17,27}	N	0.0	10000.0	10000.0	0
19	900.0	100.0	{0,0,0,0,0,0,0,0}	E	0.0	10000.0	10000.0	0
20	0.0	200.0	{0,0,0,0,0,0,0,0}	E	0.0	10000.0	10000.0	0
21	100.0	200.0	{31,32,22,12,11,10,20,30}	N	100.0	500.0	600.0	22
22	200.0	200.0	{32,33,23,13,12,11,21,31}	I	0.0	400.0	400.0	22
23	300.0	200.0	{33,34,24,14,13,12,22,32}	N	100.0	500.0	600.0	22
24	400.0	200.0	{34,35,25,15,14,13,23,33}	N	200.0	600.0	800.0	23

a)

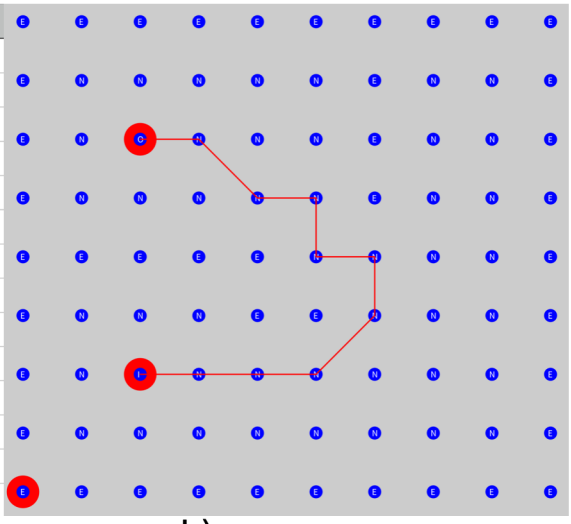


Figura 6: En a) se tiene el marco de datos generado, mostrándose solamente las filas 11 a 24, pero que en total se tienen 90 filas. En b) se muestra la ruta óptima definida por el proceso seguido en A*.

Notas sobre la robustez del modelo:

- La discretización matricial permitió que el algoritmo tratara las paredes simplemente como nodos con costo infinito o fuera del alcance de la vecindad permitida.
- La utilización de 8 vecinos incrementa la complejidad de la búsqueda frente a una vecindad de Von Neumann (4 vecinos), pero el modelo matricial propuesto mantuvo la eficiencia computacional esperada.

3.3. Aplicación 3: Optimización en entornos complejos de alta resolución

El tercer escenario de evaluación consiste en un laberinto de configuración avanzada, caracterizado por una red de barreras ortogonales (horizontales y verticales). Este entorno fue diseñado para poner a prueba la ca-

pacidad de respuesta del algoritmo ante obstáculos intrincados que generan múltiples "callejones sin salida" y rutas subóptimas (fig. 7 b).

Escalabilidad y Gestión de Datos

A diferencia de los casos anteriores, este dominio presenta una resolución significativamente mayor:

- **Dimensión del Espacio**: El entorno se discretizó en una matriz que genera un marco de datos de 4,000 celdas (filas).

- **Procesamiento**: Debido a la magnitud del espacio muestral, el análisis se realizó exclusivamente mediante la implementación computacional. La estructura de datos daFra (fig. 7 a) gestionó eficazmente la indexación y el cálculo de parámetros para la totalidad de los nodos, manteniendo la integridad de las precedencias incluso en las zonas de mayor confinamiento (fig. 7 b).

Dinámica de inicio variable y análisis de convergencia

Una de las pruebas críticas en este escenario fue la variación de las condiciones iniciales. Se ejecutaron simulaciones desplazando el punto de origen (I) a diversas coordenadas del laberinto, mientras se mantenía fijo el punto de destino (O).

Como se observa en la comparativa de la (fig. 8 a, b, c, y d), el algoritmo demostró una notable versatilidad:

- **Independencia de la posición inicial**: Independientemente de si el origen se ubicaba en los extremos o en cavidades internas (índices 82, 3682, 122 o 1860), el sistema convergió consistentemente hacia la trayectoria de menor costo.

- **Visualización del Espacio de Búsqueda**: En las representaciones gráficas, es posible identificar una serie de puntos de menor tamaño que tapizan diversas áreas del laberinto. Estos corresponden a los nodos procesados por el openSet y almacenados en el closedSet que, tras la evaluación de la función $f(n)$, fueron descartados para la trayectoria final. Esta "nube de puntosïlustra visualmente el proceso de exploración heurística y cómo el algoritmo discrimina las rutas ineficientes.

En la (fig. 9 a, b, c, y d) se puede observar que si tomamos 4 vecinos, es decir utilizamos la vecindad de von Neumann [1], se quita la posibilidad de las lineas transversales dejando solamente las ortogonales, lo cual hace que las distancias de viaje sean mayores.

Tabla 1: Rendimiento algorítmico en diversas complejidades topológicas (Baja densidad L-D, Semi-Estructurado S-E, Alta complejidad de Moore H-CM, Alta complejidad de Von Neumann H-CV). Los tiempos mostrados τ son promedios, dado que el proceso es en general aleatorio.

Scenario Configuration	Explored Nodes (N_E)	Optimal Path Cost (g_{final})	Execution Time (τ, ms)
Caso 1: L-D	8	9.26	1.3
Caso 2: S-S	21	9.66	6.0
Caso 3 a): H-CM	2728	186.15	100.8
Caso 3 b): H-CM	2718	150.91	103.9
Caso 3 c): H-CM	1160	86.35	29.1
Caso 3 d): H-CM	2715	166.64	97.2
Caso 3 a): H-CV	2913	209.0	105.4
Caso 3 b): H-CV	2817	172.05	94.2
Caso 3 c): H-CV	3100	101.00	26.1
Caso 3 d): H-CV	3095	193.00	90.7

4. Discusión de Resultados

El análisis de las tres aplicaciones experimentales muestra que el uso de una arquitectura basada en **marcos de datos (data frames)** vinculados a una **discretización matricial** redunda en la eficiencia del algoritmo A*.

Eficiencia del método matricial y estructura de datos

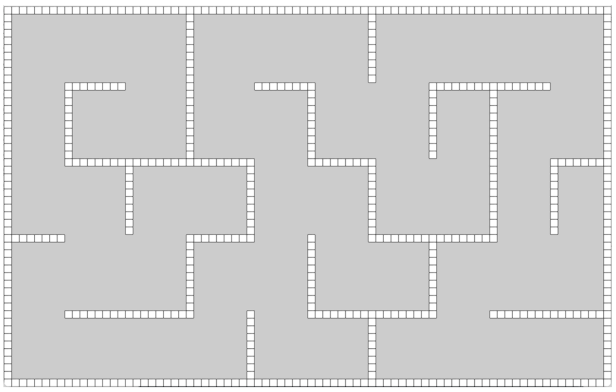
La mejora técnica de la propuesta radica en los siguientes puntos clave:

- **Optimización del acceso indexado**: Al asociar cada punto del espacio a una fila y columna específica en una matriz, el algoritmo logra un acceso indexado directo a cualquier celda. Esto elimina la necesidad de realizar búsquedas lineales costosas en cada iteración para localizar vecinos o actualizar estados.

- **Gestión de la información con marcos de datos**: El uso del marco de datos permite centralizar múltiples variables de estado (coordenadas, tipología, costos y precedencia) en una sola estructura relacional. Como se observó en la Aplicación 3 con 4,000 celdas, esta organización permite que el sistema mantenga la integridad de los datos incluso bajo una alta densidad de obstáculos.

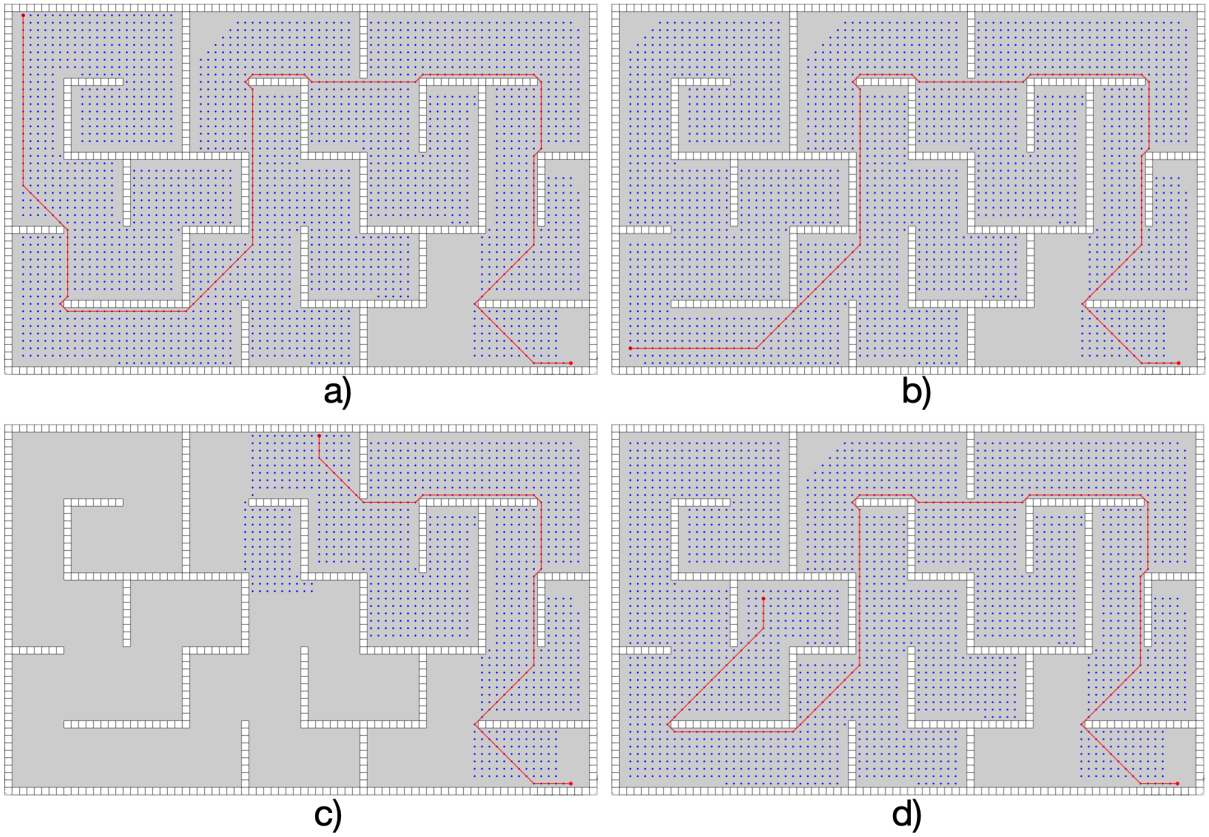
- **Evasión eficiente de obstáculos**: La metodología matricial permite tratar los obstáculos y paredes (marcadas como tipo E) simplemente como nodos con costos infinitos o inaccesibles dentro del conjunto de vecinos. Esto simplifica la lógica de navegación, ya que el núcleo del algoritmo A* no requiere modificaciones adicionales para sortear barreras complejas, como se evidenció en la coincidencia exacta entre los cálculos manuales y computarizados de las Aplicaciones 1 y 2.

80	10.0	30.0	{0,0,0,0,0,0,0,0}	E	0.0	10000.0	10000.0	0
81	30.0	30.0	{161,162,82,2,1,0,80,160}	N	20.0	2440.0	2460.0	82
82	50.0	30.0	{162,163,83,3,2,1,81,161}	N	0.0	2420.0	2420.0	82
83	70.0	30.0	{163,164,84,4,3,2,82,162}	N	20.0	2400.0	2420.0	82
84	90.0	30.0	{164,165,85,5,4,3,83,163}	N	40.0	2380.0	2420.0	83
85	110.0	30.0	{165,166,86,6,5,4,84,164}	N	60.0	2360.0	2420.0	84
86	130.0	30.0	{166,167,87,7,6,5,85,165}	N	80.0	2340.0	2420.0	85
87	150.0	30.0	{167,168,88,8,7,6,86,166}	N	100.0	2320.0	2420.0	86
88	170.0	30.0	{168,169,89,9,8,7,87,167}	N	120.0	2300.0	2420.0	87
89	190.0	30.0	{169,170,90,10,9,8,88,168}	N	140.0	2280.0	2420.0	88
90	210.0	30.0	{170,171,91,11,10,9,89,169}	N	160.0	2260.0	2420.0	89
91	230.0	30.0	{171,172,92,12,11,10,90,170}	N	180.0	2240.0	2420.0	90
92	250.0	30.0	{172,173,93,13,12,11,91,171}	N	200.0	2220.0	2420.0	91
93	270.0	30.0	{173,174,94,14,13,12,92,172}	N	220.0	2200.0	2420.0	92
94	290.0	30.0	{174,175,95,15,14,13,93,173}	N	240.0	2180.0	2420.0	93



a) b)

Figura 7: En a) se tiene el marco de datos generado, mostrándose solamente las filas 80 a 94, pero que en total se tienen 4000 filas. En b) se muestra el laberinto a resolver, solo paredes horizontales y verticales.



a) b) c) d)

Figura 8: Utilizando la vecindad de Moore (8 vecinos) en a) la celda inicial se ubico en el índice 82, en b) en el índice 3682 , en c) en el índice 122 y en d) en el índice 1860.

- **Eficiencia computacional**: El uso de una estructura matricial y tablas dinámicas permite que el algoritmo escale eficientemente en mapas de alta resolución.
- **Versatilidad geométrica**: Aunque el modelo es matricial, al asociarlo a un marco de datos la lógica de vecindad es adaptable a diferentes topologías celulares, lo que permite simular desplazamientos en entornos con restricciones físicas variadas.

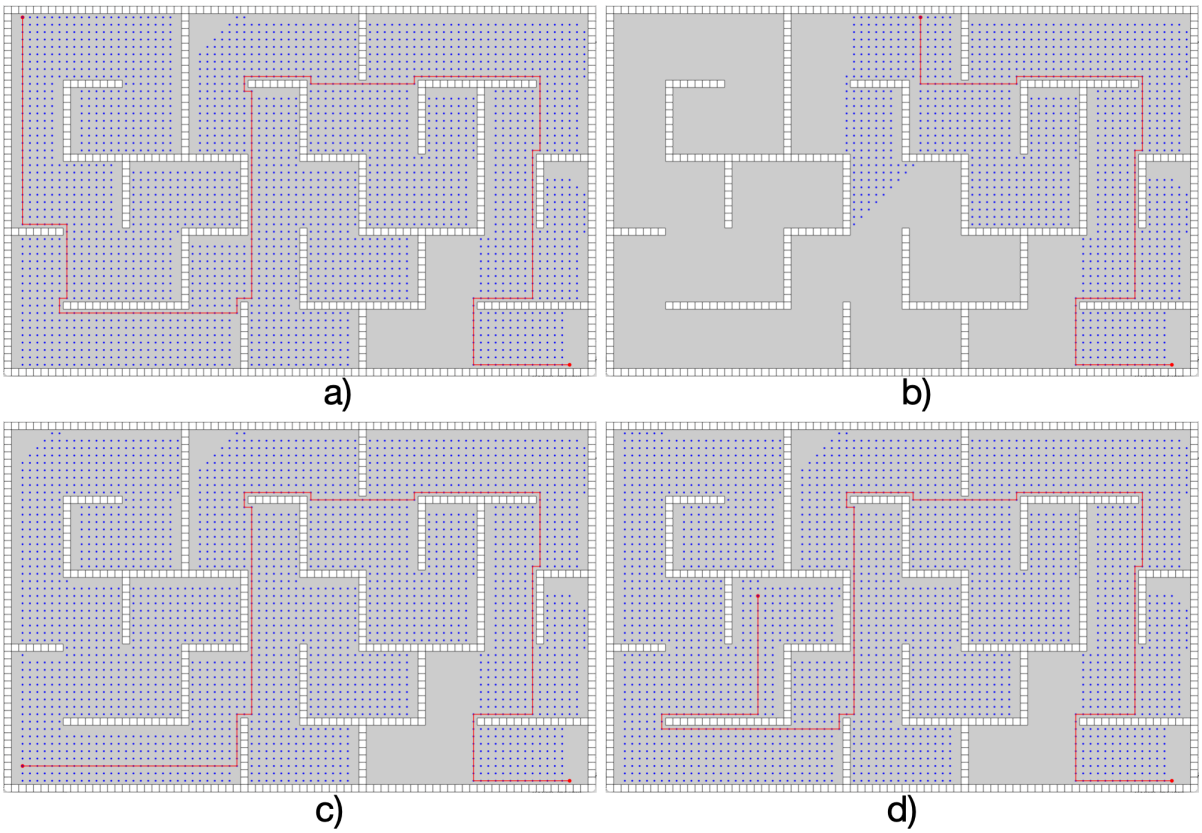


Figura 9: Utilizando la vecindad de von Neumann (4 vecinos) en a) la celda inicial se ubico en el índice 82, en b) en el índice 3682 , en c) en el índice 122 y en d) en el índice 1860.

5. Conclusiones generales

Tras la implementación y validación del modelo de optimización de rutas mediante Autómatas Celulares y el algoritmo A*, se concluye lo siguiente:

- **Robustez y versatilidad del algoritmo**: El algoritmo A* demostró una consistencia absoluta en la determinación de trayectorias de longitud mínima en escenarios de complejidad incremental, desde grafos dirigidos simples hasta laberintos de alta resolución con 4,000 nodos.

- **Validación de la Metodología Matricial**: La propuesta de utilizar marcos de datos indexados matricialmente es una estrategia altamente efectiva para el ordenamiento espacial. Este enfoque garantiza una gestión eficiente de la información, permitiendo que el algoritmo sea escalable y capaz de operar en tiempo real incluso en entornos saturados de obstáculos.

- **Independencia del punto de inicio**: Las pruebas realizadas con orígenes variables en la Aplicación 3 confirman la versatilidad del modelo, el cual converge hacia la ruta óptima de manera independiente a la ubicación inicial o la configuración del laberinto.

- **Aporte a la navegación heurística**: La integración de reglas de transición locales propias de los autómatas celulares con la búsqueda global del algoritmo A* proporciona un marco de trabajo sólido para la planificación de trayectorias en entornos discretos.

- **Jump Point Search**: Se conoce así a un algoritmo que mejora el A* en hasta un orden de magnitud [16], el cual no estamos implementando dado que los autómatas celulares hacen una priorización selectiva propia. Sin embargo, en trabajos futuros implementaremos esta técnica de búsqueda para revisar si induce una mejora en los procesos.

- **D* search algorithm**: Es otro algoritmo de búsqueda el cual está diseñado para vehículos autónomos en entornos de obstáculos dinámicos [17], el cual tampoco implementamos dado que los obstáculos con los que estamos tratando son fijos.

- **Medición de la eficiencia**: Se deja como un tra-

bajo futuro la medición comparativa de la eficiencia de este método con respecto a otros.

En resumen, este trabajo establece una base metodológica eficiente para el desarrollo de sistemas de navegación autónoma, donde la velocidad de procesamiento y la precisión en la evasión de obstáculos son requisitos críticos para la optimización de recursos energéticos y temporales.

Referencias

[1] N. Boccara, Modeling Complex Systems, 2nd Edition, Springer, 2010.

[2] D. A. Zaitsev, A generalized neighborhood for cellular automata, Theoretical Computer Science 666 (2017) 21–35. doi:https://doi.org/10.1016/j.tcs.2016.11.002.

[3] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1) (1959) 100–107. doi:https://ir.cwi.nl/pub/9256/9256D.pdf.

[4] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions Of Systems Science And Cybernetics 4 (2) (1968) 269–271. doi:https://doi.org/10.1109/TSSC.1968.300136.

[5] R. Kay, A. Mattacchione, C. K. . B. Hatton, Stepwise slime mould growth as a template for urban design, Nature Scientific Reports 12 (1322) (2022) 1–15. doi:https://doi.org/10.1038/s41598-022-05439-w.

[6] F. S. Gharehchopogh, A. Ucan, T. I. . B. Arasteh, G. Isik, Slime mould algorithm: A comprehensive survey of its variants and applications, Archives of Computational Methods in Engineering 30 (1322) (2023) 1–15. doi:https://doi.org/10.1007/s11831-023-09883-3.

[7] J. Jones, Characteristics of pattern formation and evolution in approximations of physarum transport networks, Artificial Life 16 (2) (2010) 127–153. doi:https://doi.org/10.1162/artl.2010.16.2.16202.

[8] E. F. Krause, Taxicab Geometry: An Adventure in Non-Euclidean Geometry, 1st Edition, Dover Books on Mathematics, 1987.

[9] P. E. Black, Manhattan distance, https://www.nist.gov/dads/HTML/manhattanDistance.html, accessed: 22 10 2025 (2019).

[10] M. Barile, Taxicab metric, https://mathworld.wolfram.com/TaxicabMetric.html, accessed: 22 10 2025 (2025).

[11] R. Fareh, M. Baziyad, M. H. Rahman, T. Rabie, M. Bettayeb, Investigating reduced path planning strategy for differential wheeled mobile robot, Robotica 38 (2) (2020) 1–21. doi:https://doi.org/10.1017/S0263574719000572.

[12] J. Chen, N. R. Baziyad, M. H. Sturtevant, Conditions for avoiding node re-expansions in bounded suboptimal search, Vol. 18, 2019, pp. 1220–1226. doi:https://dl.acm.org/doi/10.5555/3367032.3367206.

[13] A. V. Goldberg, H. Kaplan, R. F. Werneck, Reach for a*:efficient point-to-point shortest path algorithms, Tech. rep., accessed: 22 10 2025 (2005).

[14] C. J. Cramer, Essentials of Computational Chemistry, 2nd Edition, John Wiley & Sons, Ltd, 2004.

[15] P. Compeau, Biological Modeling A Short Tour, 1st Edition, Philomath Press, LLC, 2022.

[16] D. Harabor, A. Grastien, Online graph pruning for pathfinding on grid maps, Vol. 25, 2011, pp. 1114–1119. doi:https://doi.org/10.1609/aaai.v25i1.7994.

[17] R. N. Sarbini, I. Ahmad, R. O. Bura, L. Simbolon, Development of pathfinding using a-star and d-star lite algorithms in video game, Journal of Theoretical and Applied Information Technology 102 (3) (2024) 832–841. doi:https://www.jatit.org/volumes/Vol102No3/5Vol102No3.pdf.