PRÁCTICA 2

SATISFACCIÓN DE RESTRICCIONES Y BÚSQUEDA HEURÍSTICA

Alumno	NIA
Iván González Portero	100522281
Vanesa Elena Ionescu	100522287

ÍNDICE

PARTE 1

MODELADO

Un problema de satisfacción de restricciones se define mediante R = (X, D, C), donde:

- $X = \{x_{ij}\}$ es el conjunto de variables.

  Dado el tablero de dimensiones n x n, definimos un conjunto de variables donde cada variable representa una celda.

  $$X = \{x_{ij}\}\ donde\ i \in [0, n-1], j \in [0, n-1]$$

  Cada variable $\{x_{ij}\}$ representa el contenido de la celda en la fila i y la columna j.

- $D = \{D_{ij}\}$ representa los dominios de cada variable.

  El dominio de cada variable depende de si la celda está preasignada o no, y en caso de estarlo, de qué color es el disco.

  - Si la celda (i,j) contiene un disco negro ("x"): $\{D_{ij}\}$ = {1}
  - Si la celda (i,j) contiene un disco blanco ("o"): $\{D_{ij}\}$ = {0}
  - Si la celda (i,j) está vacía ("."): $\{D_{ij}\}$ = {0,1}

  Teniendo en cuenta que 0 representa disco blanco y 1 representa disco negro.

- $C = \{C_k\}$ es el conjunto de restricciones del problema.

  C1. **"El número de discos blancos y negros en cada fila, y en cada columna, debe ser el mismo."**

  Para cada fila, $i \in [0, n-1]$ → el número de discos negros (1) debe ser exactamente n/2:

  $$\sum_{j=0}^{n-1} x_{ij} = \frac{n}{2}$$

  Para cada columna, $j \in [0, n-1]$ → el número de discos blancos (0) debe ser exactamente n/2:

  $$\sum_{i=0}^{n-1} x_{ij} = \frac{n}{2}$$

  C2. **"No es posible disponer más de dos discos del mismo color consecutivamente en una fila o columna."**
  No pueden existir tres celdas consecutivas horizontales con el mismo color. Esto se modela asegurando que la suma de cualquier triplete adyacente no sea ni 0 (tres blancos) ni 3 (tres negros). Por tanto, la suma debe estar estrictamente entre 0 y 3.

  $$1 \leq x_{ij} + x_{ij+1} + x_{ij+2} \leq 2 \rightarrow \text{Para cada fila}$$
  $$1 \leq x_{ij} + x_{i+1j} + x_{i+2j} \leq 2 \rightarrow \text{Para cada columna}$$

  C3. **"No debe quedar ninguna posición vacía"** → ya se cumple debido a que recorremos todas las celdas de cada fila y cada columna

PARTE 2

MODELADO

Podemos definir nuestro problema como un grafo dirigido G = (V,E) donde:

- V → es el conjunto de vértices
  - v ∈ V es una función de coordenadas (latitud, longitud)
- E → conjunto de arcos con costes
- C: V x V → función de coste (distancia en metros)

Y donde nuestro objetivo es encontrar el camino de menor coste desde $v_{inicial}$ a v_{final}

ALGORITMO SELECCIONADO → A*

La elección de A* se fundamenta en cuatro pilares teóricos que garantizan la calidad y eficiencia de la solución:

- Optimalidad → El algoritmo garantiza encontrar la solución óptima (el camino de coste mínimo) siempre que la función heurística h(n) utilizada sea admisible. En esta implementación, utilizaremos la distancia Haversine (explicada en el apartado siguiente), que hace que se cumpla que $h(n) \leq h^*(n)$ asegurando así que el algoritmo sea admisible también.

- Completitud → Dado que el grafo de entrada es finito (número limitado de vértices y arcos definidos en los ficheros .gr y .co) y los costes de los arcos son estrictamente no negativos (distancias físicas en metros), el algoritmo A* es completo. Esto garantiza que, si existe una ruta conectada entre el vértice origen y el destino, el algoritmo la encontrará y terminará su ejecución.

- Eficiencia (la compararemos con Dijkstra) → A* es óptimamente eficiente para una heurística dada. A diferencia del algoritmo de Dijkstra, que explora los nodos radialmente en todas direcciones (equivalente a A* con h(n) = 0), A* utiliza la información heurística para "guiar" la búsqueda hacia el objetivo. Esto permite podar gran parte del espacio de búsqueda, reduciendo drásticamente el número de nodos expandidos y, en algunos casos (en los que no haya barreras geográficas como bahías o lagos), el tiempo de ejecución y el consumo de memoria.

- Aplicabilidad al Dominio → Este problema es ideal para A* porque disponemos de información espacial explícita. Los ficheros de coordenadas (.co) proporcionan la latitud y longitud de cada vértice. Esta información geométrica permite calcular estimaciones precisas del coste restante.

Función de Evaluación

El algoritmo ordena la exploración de los nodos mediante la minimización de la función de evaluación f(n), definida como:

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

Donde cada componente representa:

1. g(n) (Coste Real Acumulado) → Representa el coste exacto del camino recorrido desde el nodo inicial hasta el nodo actual n. Este valor se calcula sumando los pesos de los arcos (distancias en metros especificadas en el fichero .gr) que conforman la ruta actual.

$$g(n) = \sum_{i=inicio}^{n} coste_{arco}(i, i+1)$$

2. h(n) (Estimación Heurística) → Es la estimación del coste mínimo restante desde el nodo n hasta el nodo objetivo. Se calcula utilizando la distancia de Haversine, que mide la distancia del arco de círculo máximo sobre la esfera terrestre entre las coordenadas del nodo n y el destino.

$$h(n) = Haversine(n, objetivo)$$

3. f(n) (Coste Total Estimado) → Representa la estimación del coste total del camino más barato que pasa por n. El algoritmo utilizará una estructura de cubos (Dial's Bucket) para rastrear el índice del cubo no vacío más bajo, asegurando así la eficiencia en la selección del siguiente estado a expandir.

HEURÍSTICA ELEGIDA → DISTANCIA HAVERSINE

Como hemos dicho en el apartado anterior, hemos elegido la distancia Haversine. Antes de llegar a esta conclusión, habíamos considerado otras dos heurísticas, que justificaremos a continuación el porqué de su descarte:

- Distancia Euclídea (Teorema de Pitágoras) → $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ → Descartada porque, aunque sí que es verdad que mide la línea recta más corta entre dos puntos, lo hace en un plano 2D, donde las coordenadas son cartesianas. Como nuestras coordenadas son Latitud/Longitud, el resultado sería erróneo porque la Tierra no es plana.
- Distancia Manhattan → $|x_2 - x_1| + |y_2 - y_1|$ → Descartada porque podría no ser admisible ya que, al usarse en mapas de carreteras, podría sobreestimar la distancia real, ya que puede haber carreteras que vayan en diagonal.

Se ha elegido la distancia Haversine porque se fundamenta en las propiedades matemáticas de consistencia y admisibilidad, las cuales son críticas para garantizar tanto la optimalidad de la solución como la eficiencia computacional del algoritmo.

Fórmula de Haversine

$$d = 2r\,arcsen\left(\sqrt{sen^2\left(\frac{lat2 - lat1}{2}\right) + \cos(lat1)\cos(lat2)\,sen^2\left(\frac{lon2 - lon1}{2}\right)}\right)$$

- Consistencia → La distancia Haversine es consistente, ya que para cualquier nodo n, y cualquier sucesor n' generado por una acción con coste c(n.n') se cumple:

$$h(n) \leq c(n, n') + h(n')$$

  Donde:
  - H(n) es la distancia en línea recta desde n al objetivo
  - H(n') es la distancia en línea recta desde el vecino n' al objetivo
  - C(n,n') es la distancia física real por carretera entre n y n'
- Admisibilidad → "toda heurística consistente es admisible". La admisibilidad requiere que h(n) nunca sobreestime el coste real ($h^*(n)$). Dado que la distancia Haversine representa el límite inferior físico de la distancia entre dos coordenadas, y no se puede viajar más rápido que en línea recta) se cumple que:

$$h_{Haversine}(n) \leq h^*(n)$$

- Optimalidad → Al ser admisible, el algoritmo A* garantiza la optimalidad (la primera solución completa que se encuentre será la de menor coste posible).

En algoritmos A* es posible llegar a un nodo por un camino subóptimo, cerrarlo, y posteriormente descubrir un camino mejor hasta ese mismo nodo. Esto obliga a reabrir el nodo, lo cual degrada el rendimiento. Pero gracias a la propiedad de consistencia de nuestra heurística, se asegura que el primer camino encontrado hacia cualquier nodo es el óptimo, permitiendo el uso eficiente de una Lista Cerrada sin necesidad de re-expandir nodos ya visitados.

DECISIONES DE IMPLEMENTACIÓN Y ESTRUCTURAS DE DATOS

Al realizar el trabajo en Python, hemos diseñado estructuras de datos específicas que minimizan la complejidad temporal de las operaciones más frecuentes_

- Lista abierta basada en Dial's Bucket con conjuntos (Sets) → utilizamos un diccionario donde las claves son los costes enteros de la función f(n) y los valores son sets que contienen los nodos con dicho coste. En A* es frecuente encontrar caminos mejores a nodos que ya están en la lista abierta, requiriendo su actualización rápida sin necesidad de recorrer toda la estructura. Dado que las distancias son float, realizamos una conversión a entero para indexar los buckets eficientemente.
- Caché de heurística → el cálculo de la distancia Haversine involucra operaciones trigonométricas costosas que, en grafos grandes, el algoritmo puede consultar la heurística del mismo nodo múltiples veces. Hemos incluido un mecanismo de caché en la clase AlgoritmoAEstrella. Antes de calcular h(n) verificamos si ya ha sido computada, si es así devolvemos el valor almacenado instantáneamente, evitando recalcular la fórmula de Haversine. Esto reduce significativamente la sobrecarga de CPU en búsqueda de larga distancia donde se expanden cientos de miles de nodos
- Hemos decidido mantener también la implementación de Dijkstra (y su aparición en la salida por terminal) ya que en algunos casos (que veremos en las pruebas) sí que tarda menos en encontrar la solución óptima, aunque el número de nodos expandidos nunca supere a los de A*.

PARTE 3

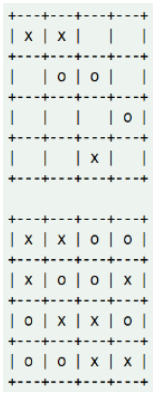
PRUEBAS PARTE 1

PRUEBA 1 – 4X4

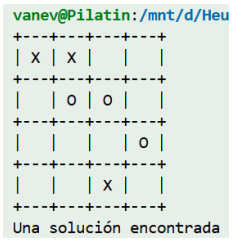
Fichero .in

```
xx..
.oo.
...o
..x.
```

Escritura en .out



Salida por terminal



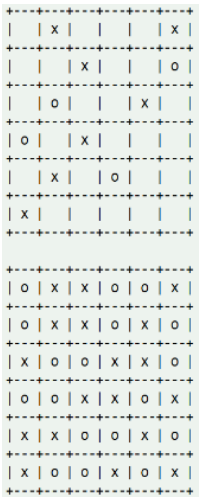
En esta prueba inicial, el sistema validó correctamente el modelo resolviendo las restricciones estructurales básicas sobre una instancia preasignada. El algoritmo encontró la solución única de forma inmediata

PRUEBA 2 – 6X6

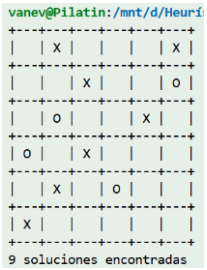
Fichero .in

```
.x...x
..x..o
.o..x.
o.x...
.x.o..
x.....
```

Escritura en .out



Salida por terminal



Al aumentar la complejidad, el solver demostró su capacidad para explorar exhaustivamente el espacio de búsqueda, identificando correctamente las 9 soluciones válidas existentes

PRUEBA 3 – CONSECUTIVOS

Fichero .in

```
XXX...
......
......
......
......
......
```

Escritura en .out

Salida por terminal

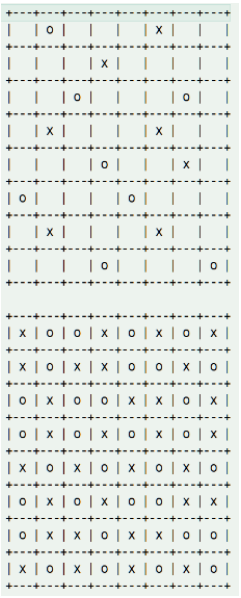
Al introducir un tablero con una violación explícita de las reglas el sistema detectó inmediatamente la inconsistencia, devolviendo 0 soluciones.

PRUEBA 4 – 8X8

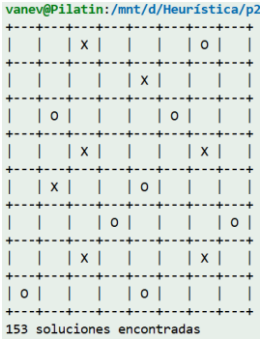
Fichero .in

```
..X...O.
....X...
.O...O..
..X...X.
.X..O...
...O...O
..X...X.
O...O...
```

Escritura en .out



Salida por terminal

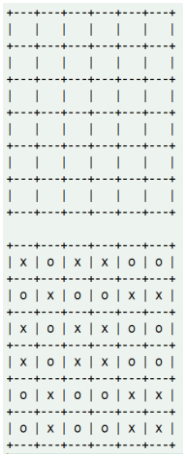


Para la instancia de 8x8, el solver manejó un espacio de estados mucho mayor.

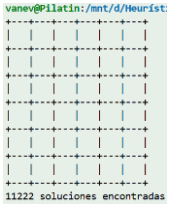
PRUEBA 5 – VACÍO

Fichero .in

Escritura en .out



Salida por terminal



En esta prueba de estrés sin preasignaciones, el algoritmo fue capaz de explorar y enumerar la totalidad del espacio de soluciones, identificando las 11.222 configuraciones válidas

ANÁLISIS DE LA COMPLEJIDAD DEL PROBLEMA

Hemos definido N^2 variables, correspondiendo una a cada celda del tablero. Cada variable se identifica por sus coordenadas (i,j) donde $i, j \in [0, N-1]$. El dominio de cada variable es binario $D_{ij} = \{0,1\}$ menos las celdas que ya tienen un valor asignado (su dominio sería $D_{ij} = \{0\}$ o $D_{ij} = \{1\}$).

También hemos definido dos tipos de restricciones explícitas.

- N restricciones para asegurar que cada fila suma exactamente N/2 (y lo mismo para las columnas). → 2N restricciones en total
- Además, para cada fila hay (N-2) tríos posibles (en las columnas igual) → N(N-2) → en el tablero total →2 N(N-2).

El número total de restricciones es 2N + 2N(N-2).

PRUEBAS PARTE 2

USA-ROAD-D.BAY

Este mapa se caracteriza por una geografía compleja con una gran bahía central que obliga a realizar rodeos significativos, lo que pone a prueba la capacidad de la heurística para guiar la búsqueda en presencia de obstáculos físicos grandes.

PRUEBA 1 – DISTANCIA CORTA (50.000 – 52.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	159757	10886	0,05	198691.87
Dijkstra	159757	14848	0,07	211352.87

En trayectos locales, la heurística es extremadamente efectiva, reduciendo las expansiones un 26,68% (10.886 - 14.848). Al no haber grandes obstáculos geográficos entre origen y destino, A* va directo al objetivo, siendo más rápido que Dijkstra (0,05s - 0,07s).

PRUEBA 2 – DISTANCIA MEDIA (5000 - 55.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	150421	14090	0,06	223953.49
Dijkstra	150421	18059	0,07	251369.55

La eficiencia se mantiene alta con un ahorro del 21,98% en nodos expandidos. El algoritmo sigue aprovechando la directividad de la distancia Haversine, logrando tiempos de ejecución competitivos (0,06s) frente a la fuerza bruta.

PRUEBA 3 – DISTANCIA LARGA (1 – 320.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	1853034	308442	1,92	160418.07
Dijkstra	1853034	312417	1,74	179826.93

Aquí se observa el impacto de la topología "en herradura" de BAY. La reducción de nodos cae drásticamente al 1,27%. La heurística subestima el coste real (al medir por encima del agua), haciendo que A* explore casi tanto como Dijkstra. De hecho, el coste computacional extra de calcular la heurística hace que A* sea ligeramente más lento (1,92s - 1,74s).

Las pruebas de USA-road-d.NY y las de USA-road-d.COL son bastante parecidas a las de USA-road-d.BAY ya que el número de vértices difiere por 100.000 aproximadamente. Por ello hemos decidido mostrar los resultados de USA-road-d.FLA a continuación, porque hay un cambio significativo en el número de vértices

USA-ROAD-D.FLA

Florida presenta una geometría alargada y peninsular, lo que limita las opciones de ruta y puede generar cuellos de botella en la exploración.

PRUEBA 4 – DISTANCIA CORTA (300.000 – 303.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	478710	16952	0,07	234092.47
Dijkstra	478710	20234	0,07	293349.74

El comportamiento es excelente en distancias cortas, con una reducción del 16,22% en el árbol de búsqueda. La heurística guía eficazmente al algoritmo, resultando en tiempos de ejecución idénticos a Dijkstra, pero con menor uso de memoria implícito.

PRUEBA 5 – DISTANCIA MEDIA (500.000 – 550.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	2215415	249901	1.91	130655.19
Dijkstra	2215415	277886	2.06	134970.34

A medida que aumenta la distancia, la eficiencia disminuye moderadamente a un 10,07% de ahorro. Aunque se expanden menos nodos (249.901 - 277.886), la ventaja temporal se diluye, siendo A* prácticamente igual de rápido que Dijkstra.

PRUEBA 6 – DISTANCIA LARGA (500.000 – 50.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	4070453	670825	5.86	114567.12
Dijkstra	4070453	716384	4.54	157864.72

Se confirma la tendencia de degradación en mapas costeros. El ahorro de nodos se reduce al 6,36% y el tiempo de ejecución de A* (5,86s) supera al de Dijkstra (4,54s). Esto sugiere que en grafos donde la línea recta difiere mucho de la ruta real por carretera, la sobrecarga de cálculo de la heurística no se amortiza suficientemente.

USA-ROAD-D.LKS

Similar a BAY, la presencia de grandes lagos crea obstáculos insalvables que la heurística Haversine no puede prever.

PRUEBA 10 – DISTANCIA CORTA (1.000.000 – 1.005.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	1894516	285581	2.86	99772.78
Dijkstra	1894516	339752	2.99	113780.41

Comienza con un buen rendimiento, ahorrando un 15,94% de expansiones y siendo ligeramente más rápido en tiempo (2,86s - 2,99s).

PRUEBA 11 – DISTANCIA MEDIA (1.500.000 - 1.700.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	9151722	2137065	23.65	90367.68
Dijkstra	9151722	2383403	23.15	102954.65

Se aprecia una inversión en el rendimiento temporal. Aunque A* expande un 10,34% menos de nodos, tarda más tiempo (23,65s) que Dijkstra (23,15s). Esto indica que el coste de gestión de la cola de prioridad y los cálculos geométricos pesa más que el ahorro obtenido.

PRUEBA 12 – DISTANCIA LARGA (1.000.000 – 1.500.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	7141897	1814965	20.98	86499.92
Dijkstra	7141897	1875599	18.88	99323.11

El rendimiento cae a un ahorro mínimo del 3,23%. Al tener que rodear los Grandes Lagos, la heurística "miente" constantemente sobre la dirección óptima, provocando que A* se comporte casi como una búsqueda ciega, pero con mayor overhead computacional.

Podemos ver que los mapas costeros (BAY, FLA, LKS) Tienen formas cóncavas (bahías, penínsulas, lagos). La heurística "línea recta" intenta cruzar el agua, pero como no hay carretera, A* se ve obligado a expandir muchos nodos laterales para rodear el obstáculo. Cuanto más lejos vas, más "trampas" de agua encuentras, degradando el rendimiento.

USA-ROAD-D.NE

Una región densa con una red de carreteras muy interconectada, lo que suele favorecer a las búsquedas guiadas al existir múltiples opciones de ruta.

PRUEBA 7 – DISTANCIA CORTA (700.000 - 703.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	164167	34084	0.33	103452.00
Dijkstra	164167	40051	0.32	125721.04

A* demuestra su valía con un ahorro del 14,90% en expansiones. La alta densidad de nodos en esta región hace que podar el 15% del grafo signifique evitar procesar miles de estados inútiles.

PRUEBA 8 – DISTANCIA MEDIA (500.000 - 550.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	1331031	401807	3.67	109549.80
Dijkstra	1331031	463055	3.52	131449.09

La eficiencia se mantiene muy estable, con una reducción del 13,23%. A diferencia de los mapas costeros, aquí no hay una caída brusca de rendimiento, lo que indica una buena correlación entre la distancia física y la distancia por carretera en la región.

PRUEBA 9 – DISTANCIA LARGA (1 - 1.500.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	2890784	1366766	15.63	87433.27
Dijkstra	2890784	1457309	16.02	90974.97

A pesar de la enorme distancia, A* logra mantener un ahorro del 6,21%. Lo más destacable es que, en este caso, A es más rápido* (15,63s) que Dijkstra (16,02s), demostrando que, en grafos masivos y densos, incluso una poda pequeña justifica el uso de la heurística para mejorar el tiempo total.

USA-ROAD-D.W

Un mapa vasto y de interior. A diferencia de los anteriores, aquí la topología juega a favor de la heurística, manteniendo un rendimiento robusto.

PRUEBA 13 – DISTANCIA CORTA (2.500.000 – 2.505.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	4459597	515795	4.61	56843.12
Dijkstra	4459597	590263	4.76	67577.98

Se obtiene un ahorro sólido del 12,62% en expansiones, con A* siendo más rápido (4,61s - 4,76s).

PRUEBA 14 – DISTANCIA MEDIA (2.500.000 - 2.600.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	5473952	681239	13.10	52003.08
Dijkstra	5473952	799129	14.06	56823.85

La eficiencia mejora con la distancia, alcanzando un 14,75% de reducción de nodos. Esto valida la hipótesis de que, en mapas continentales abiertos, las carreteras interestatales permiten trayectos muy directos que se alinean bien con la heurística Haversine.

PRUEBA 15 – DISTANCIA LARGA (2.800.000 – 1.000.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	6766427	1278777	19.08	67008.48
Dijkstra	6766427	1422844	21.19	67146.09

El algoritmo muestra su mejor cara en gran escala. A* mantiene un ahorro superior al 10% y es notablemente más rápido (19,08s) que Dijkstra (21,19s). Ahorrar 2 segundos de cómputo en una sola consulta evidencia la escalabilidad de la solución A* en este tipo de grafos.

USA-ROAD-D.CTR

Este mapa cubre las grandes llanuras centrales de USA, una región caracterizada por una orografía muy plana y una red de carreteras con fuerte tendencia a la ortogonalidad (rejilla), lo que a priori favorece enormemente a las heurísticas geométricas.

PRUEBA 16 – DISTANCIA CORTA (5000000 5005000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	924810	49138	0.57	86557.97
Dijkstra	924810	64231	0.58	109834.85

Los resultados confirman la idoneidad de la topología: A* logra una reducción del 23,50% en el número de nodos expandidos (49.138 - 64.231). Al no existir grandes barreras geográficas (como bahías o lagos) y disponer de una red viaria muy directa, la heurística Haversine guía la búsqueda con gran precisión, permitiendo una poda muy superior a la observada en mapas más irregulares como los costeros.

PRUEBA 17 – DISTANCIA MEDIA (4.000.000 - 4.200.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	4926766	2071255	37.97	54550.46
Dijkstra	4926766	2701257	45.57	59278.10

Los resultados validan la robustez de la heurística en esta región, manteniendo una reducción de nodos del 23,32% (2.071.255 - 2.701.257). A diferencia de los mapas costeros donde el rendimiento cae con la distancia, aquí la estructura de rejilla de las carreteras permite que A* aproveche esa poda masiva para reducir el tiempo de ejecución en casi 8 segundos (37,97s - 45,57s), demostrando una eficiencia computacional muy superior en escenarios de alta carga.

PRUEBA 18 – DISTANCIA LARGA ()

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	4230916	1073552	18.15	59160.25
Dijkstra	4230916	1389951	22.49	61796.29

La tendencia se mantiene estable incluso en trayectos largos, con un ahorro del 22,76% en expansiones. La consistencia de estos resultados (siempre por encima del 20% de ahorro) confirma que la topología plana y sin grandes accidentes geográficos del centro de USA es el escenario ideal para la distancia Haversine. El algoritmo A* logra ser más de 4 segundos más rápido (18,15s - 22,49s) que Dijkstra, justificando plenamente el uso de información heurística en este tipo.

PRUEBA 19 – USA-LOAD-D.USA (2.500.000 - 4.500.000)

Algoritmo	Coste óptimo	Expansiones	Tiempo (s)	Nodos/segundo
A*	16367298	4183203	38,41	108916,68
Dijkstra	16367298	4615407	39,16	117867,95

Esta última prueba verifica que nuestra heurística es efectiva en mapas y distancias grandes, ya que las expansiones en este caso son un 9,36% menores en A* y el tiempo (38,41 – 39,16) también es menor que Dijkstra.

Los Mapas de interior (W, NE, CTR) son masas de tierra continuas. Las carreteras suelen ser más directas. Aquí la heurística funciona bien tanto en corto como en largo, manteniendo el rendimiento estable.

ANÁLISIS DE LA COMPLEJIDAD DEL PROBLEMA

En relación con la complejidad espacial, el consumo de memoria de nuestra implementación escala de manera lineal respecto al tamaño del grafo y el espacio de búsqueda. La estructura estática del mapa se almacena mediante listas de adyacencia, ocupando un espacio de orden O(V+E), lo cual resulta óptimo para grafos dispersos como las redes de carreteras. Por otro lado, las estructuras dinámicas necesarias para la búsqueda crecen proporcionalmente al número de nodos visitados. Por último, hemos optimizado la Lista Abierta usando diccionarios dinámicos, en lugar de reservar un espacio enorme para todos los costes posibles desde el principio (lo que ocuparía mucha memoria inútilmente), nuestra estructura Dial's Bucket solo gasta memoria para los nodos que estamos procesando en cada momento, permitiendo resolver mapas grandes sin saturar el ordenador.

Respecto a la eficiencia temporal y la reducción del espacio de búsqueda, el impacto de la heurística de distancia geodésica (Haversine) varía considerablemente según la topología del terreno. En escenarios favorables de topografía plana, como los mapas del centro y oeste de USA (CTR, W), hemos observado reducciones consistentes de entre el 15% y el 23% en el número de nodos expandidos. Esta poda masiva se traduce directamente en una disminución significativa del tiempo de ejecución (hasta 8 segundos en consultas largas), amortizando el coste computacional del cálculo heurístico. Sin embargo, en mapas con accidentes geográficos complejos (BAY, LKS), la reducción es menor (entre el 1% y el 5%), ya que la distancia en línea recta subestima demasiado el coste real de rodear obstáculos, lo que obliga al algoritmo a explorar más nodos laterales.

Basándonos en estos resultados, hemos seleccionado el algoritmo A* con optimización de Dial's Bucket como la estrategia más adecuada para resolver este problema. Esta elección garantiza la optimalidad matemática de la solución gracias a la admisibilidad de la heurística Haversine, a la vez que mejora la eficiencia operativa frente a Dijkstra al dirigir la exploración hacia el objetivo. Además, la implementación específica de Dial's Bucket con conjuntos (set) para los cubos permite realizar inserciones y eliminaciones en tiempo amortizado, eliminando el coste lineal de actualizar nodos en la frontera, una optimización crítica para mantener el rendimiento en rutas de larga distancia sobre grafos masivos con pesos enteros.