Universidad Carlos III

Curso Heurística y Optimización 2025-26

Práctica 2

Segunda Práctica
Satisfacción de Restricciones y Búsqueda Heurística.

Grupo: **80**

Alumnos:

Álvaro García González

100522223@alumnos.uc3m.es

Samuel Gómez Fernández

100522224@alumnos.uc3m.es

Índice

1. Introducción

Este documento describe el proceso de modelado y resolución de dos tipos de problemas: **Satisfacción de Restricciones (CSP)** y **búsqueda óptima en grafos mediante heurísticas**. El propósito es doble: por un lado, implementar soluciones correctas para ambos enfoques; y por otro, analizar su comportamiento y escalabilidad mediante pruebas experimentales.

La práctica se estructura en tres partes: un **modelo CSP para resolver el pasatiempo BINAIRO** con *python-constraint* en tableros N x N; un sistema para obtener rutas mínimas en mapas reales de carreteras (DIMACS); y un **análisis de resultados** centrado en eficiencia, complejidad y robustez. En esta memoria se detallan los modelos adoptados, las decisiones de implementación y el análisis empírico obtenido.

2. Descripción de los modelos

2.1. Parte 1: Satisfacción de Restricciones

El problema de BINAIRO se ha modelado como un problema de **satisfacción de restricciones (CSP)**, siguiendo las condiciones del enunciado: (i) no puede quedar ninguna casilla vacía, (ii) en cada fila y en cada columna debe haber el mismo número de discos blancos y negros, y (iii) no puede haber más de dos discos consecutivos del mismo color ni en filas ni en columnas.

Variables y dominio

Sea un tablero de tamaño $n \times n$. Definimos una variable para cada casilla de la rejilla:

$$x_{i,j} \qquad para\ i,j \in \{0,\dots,n-1\}$$

Cada variable representa el contenido de la casilla $(i,\ j)$. Para facilitar la formulación de las restricciones de equilibrio por filas y columnas, se codifican los colores como valores enteros:

- $x_{i,j} = 0$ si la casilla contiene un disco blanco (“O”).
- $x_{i,j} = 1$ si la casilla contiene un disco negro(“X”).

El **dominio** inicial de todas las variables es: $D_{i,j} = \{0, 1\}$

Posteriormente, las casillas fijadas en la instancia se restringen mediante restricciones unarias $x_{i,j} = 0$ o $x_{i,j} = 1$. De este modo, la condición “no debe quedar ninguna posición vacía” se cumple de forma implícita, ya que en el modelo no existe la posibilidad de asignar el símbolo “.”; todas las casillas deben tomar necesariamente uno de los dos valores del dominio.

Restricciones Implementadas:

1. **Incorporación de la instancia inicial**

La instancia inicial se lee de un fichero de texto con n líneas y n caracteres por línea, donde cada símbolo es ".", "X" u "O".

- Si en la entrada aparece "O" en la posición $(i,\ j)$, se impone la restricción: $x_{i,j} = 0$
- Si en la entrada aparece "X" en la posición $(i,\ j)$, se impone la restricción: $x_{i,j} = 1$
- Si aparece ".", la casilla queda libre de restricciones adicionales y su valor se determinará durante la búsqueda, respetando únicamente las restricciones globales del problema.

En la implementación, estas restricciones se modelan como restricciones **unarias** sobre la variable correspondiente, fijando su valor al indicado por la instancia.

2. **Restricciones de suma exacta en filas y columnas**

El enunciado exige que, en cada fila y en cada columna, el número de discos blancos y negros sea el mismo. Dado que $x_{i,j} \in \{0, 1\}$ y hemos asociado el valor 1 a los discos negros, el número de discos negros en una fila i es simplemente la suma de las variables de esa fila. Para que haya el mismo número de blancos que de negros, en un tablero de tamaño n (siendo n par) debe cumplirse que en cada fila hay exactamente $n/2$ celdas con valor 1.

Por tanto, para cada fila i imponemos: $\sum_{j=0}^{n-1} x_{i,j} = \frac{n}{2}, \qquad \forall i = 0,\ldots, n-1$

De forma análoga, para cada columna j: $\sum_{i=0}^{n-1} x_{i,j} = \frac{n}{2}, \qquad \forall j = 0,\ldots, n-1$

En el código estas restricciones se implementan mediante restricciones de **suma exacta** (*ExactSumConstraint*) sobre las variables de cada fila y cada columna, fijando el valor objetivo de la suma a $n/2$.

3. **Restricción de no más de dos consecutivos**

La tercera condición del juego establece que no pueden existir tres discos consecutivos del mismo color, ni en horizontal ni en vertical. En términos del modelo, esto implica que, para cualquier trío de casillas consecutivas en una fila o columna, no es posible que las tres variables tengan el mismo valor.

Formalmente, para cada fila i y cada posición $j \in \{0,\ldots, n-3\}$, se impone:

$$\neg(x_{i,j} = x_{i,j+1} = x_{i,j+2})$$

y, de manera equivalente, para las columnas, para cada columna j y cada $i \in \{0,\ldots, n-3\}$:

$$\neg(x_{i,j} = x_{i+1,j} = x_{i+2,j})$$

Dado que las variables toman valores en $\{0, 1\}$, esta condición también puede interpretarse como exigir que la suma de cada trío consecutivo no sea ni 0 ni 3, es decir: $1 \leq x_a + x_b + x_c \leq 2$

En la implementación se define una restricción ternaria que, dados tres valores (a,b,c), devuelve falso únicamente cuando los tres son iguales (0,0,0 o 1,1,1), y verdadero en cualquier otro caso. Esta restricción se aplica a todos los tríos consecutivos de cada fila y cada columna.

2.2. Parte 2: Algoritmos de Búsqueda

En la segunda parte de la práctica se modela el problema de encontrar la mejor ruta entre dos vértices de una red viaria (mapas de la 9th DIMACS Shortest-Path Challenge), como un problema de **búsqueda de camino mínimo sobre un grafo dirigido y ponderado**, resuelto mediante el algoritmo **A***. El objetivo es, dado un vértice de inicio y otro de destino, obtener un camino cuya suma de costes (distancias de los arcos) sea mínima.

Modelo formal del grafo

Modelamos cada mapa de carreteras como un **grafo dirigido y ponderado**: $G = (V, E, w)$
Donde:

- $V = \{1, 2, \ldots, N\}$ es el conjunto de **vértices**; cada vértice $i \in V$ representa un punto geográfico de la red viaria (intersección).
- $E \subseteq V \times V$ es el conjunto de **arcos dirigidos**. Un arco $(u, v) \in E$ existe si en el fichero *.gr* aparece una línea de la forma *a u v c,* indicando que es posible desplazarse desde u hasta v.
- $w : E \to \mathbb{R}^+$ es la **función de coste** asociada a los arcos. Para cada arco (u, v), el valor $w(u, v)$ es la distancia, en metros, del tramo de carretera que une ambos vértices y se lee directamente del fichero *.gr*.

Adicionalmente, disponemos de las coordenadas geográficas de cada vértice. A cada $i \in V$ se le asocia un par (λ_i, φ_i), donde: λ_i es la longitud; Y φ_i es la latitud. Ambas expresadas como enteros multiplicados por 10^6 en el fichero *.co.*

Modelo de búsqueda (Espacio de estados)

Sobre este grafo, el problema de camino más corto se modela como un problema de búsqueda:

- **Estados**: cada estado del espacio de búsqueda se corresponde con un vértice $n \in V$.
- **Estado inicial**: es el vértice $s \in V$ indicado en el primer argumento del script.
- **Estado meta**: es el vértice $t \in V$ indicado en el segundo argumento del script.

Desde un estado n las **acciones** posibles consisten en recorrer cualquiera de los arcos salientes $(n, n') \in E$, pasando al estado sucesor n'. La **función de transición** se escribe como:

$$\text{Sucesores}(n) = \{n' \in V \mid (n, n') \in E\}$$

El **coste de transición** entre dos estados adyacentes es el coste del arco correspondiente:

$$c(n, n') = w(n, n') \quad \text{para } (n, n') \in E$$

Así, cualquier ruta válida de s a t queda representada por una secuencia de vértices:

$$P = (n_0, n_1, \ldots, n_k)$$

Tal que $n_0 = s$, $n_k = t$ y $(n_i, n_{i+1}) \in E$ para todo i. El coste total de la ruta se define como la suma de los costes de los arcos:

$$C(P) = \sum_{i=0}^{k-1} w(n_i, n_{i+1})$$

El objetivo del modelo es encontrar una ruta P^* que minimice $C(P)$ entre todas las rutas posibles de s a t.

Algoritmo de búsqueda

Para resolver el problema de camino mínimo se ha elegido el algoritmo **A*** como algoritmo principal. Para cada nodo n:

- $g(n)$: Coste acumulado del mejor camino conocido desde el origen s hasta el vértice n.
- $h(n)$: Estimación heurística del coste mínimo restante desde n hasta el destino t.
- $f(n) = g(n) + h(n)$: Función de evaluación que guía la expansión de nodos.

Adicionalmente, se ha implementado el algoritmo de **Dijkstra** (equivalente funcionalmente a A* con $h(n) = 0$). Este algoritmo se utiliza únicamente en el análisis experimental como método de fuerza bruta para validar dos aspectos: garantizar que la solución devuelta por A* es efectivamente óptima (los costes deben ser idénticos) y cuantificar la ganancia de rendimiento (reducción de nodos expandidos) aportada por la heurística.

Modelo de la heurística

Para guiar la búsqueda se define una heurística basada en la **distancia geodésica** (gran círculo) entre las coordenadas del vértice n y las del destino t, calculada mediante la **fórmula de Haversine** sobre una esfera de radio $R \approx 6\,371\,000$ metros.

1. Reescala coordenadas en grados:
   $$\lambda_i^\circ = \frac{\lambda_i}{10^6}, \quad \varphi_i^\circ = \frac{\varphi_i}{10^6}$$
2. Se convierten coordenadas en radianes:
   $$\varphi_n = \text{rad}(\varphi_n^\circ), \quad \lambda_n = \text{rad}(\lambda_n^\circ)$$

$\varphi_t = \mathrm{rad}(\varphi_t^\circ), \quad \lambda_t = \mathrm{rad}(\lambda_t^\circ)$

3. Se calculan las diferencias: $\Delta\varphi = \varphi_t - \varphi_n, \quad \Delta\lambda = \lambda_t - \lambda_n$
4. La fórmula de Haversine define:

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_n)\cos(\varphi_t)\sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \arctan 2(\sqrt{a}, \sqrt{1-a})$$

5. Finalmente, la heurística se toma como: $h(n) = R \cdot c$

Este valor corresponde a la distancia mínima posible sobre la superficie de la Tierra entre n y t. Dado que los costes de los arcos representan distancias de carretera, y las carreteras nunca pueden ser más cortas que la línea geodésica entre dos puntos, se cumple que:

$$h(n) \leq \text{distancia real mas corta por carretera desde } n \text{ hasta } t$$

Por tanto, la heurística es **admisible** (no sobrestima el coste real) y, en la práctica, también **consistente**, ya que para cada arco $(u, v) \in E$ se cumple aproximadamente: $h(u) \leq c(u, v) + h(v)$. Bajo estas condiciones, el modelo garantiza que el algoritmo A* encuentra siempre una **solución óptima**.

Estructuras de datos asociadas al modelo

Aunque esta sección no detalla código, sí es relevante especificar las **estructuras de datos** que materializan el modelo anterior:

1. **El grafo G se almacena mediante:**
   a. Un vector de **coordenadas** de tamaño $N+1$, donde *coordenadas[i]*$= (\lambda_i, \varphi_i)$ permite acceder en tiempo constante a las coordenadas del vértice i.
   b. Una lista de **adyacencia** de tamaño $N+1$, donde cada posición *adyacencia[u]* es un diccionario que asocia cada vecino v con el coste $c(u, v)$. Esto implementa directamente la función de sucesores $\mathrm{Sucesores}(n)$ y la función de coste $c(n, n')$, con acceso promedio $O(1)$.
2. **La lista abierta** del algoritmo de búsqueda se modela mediante una **lista simple** (no ordenada) que almacena pares $(f(n), n)$ y permite extraer en cada paso el nodo con menor valor de f utilizando una búsqueda lineal.
3. **La lista cerrada** se modela como un conjunto de vértices ya expandidos. Esta estructura representa el subconjunto de estados para los cuales ya se ha encontrado el mejor $g(n)$ posible y evita reexpansiones innecesarias.
4. **Además, se mantiene**: una estructura asociativa para $g(n)$ (diccionario *g_score*), donde se almacena el mejor coste acumulado conocido desde s hasta cada vértice; y una estructura (diccionario *came_from*) que modela la relación *padre* $\rightarrow$ *hijo* en el árbol de búsqueda, permitiendo reconstruir el camino P^* una vez alcanzado el destino t.

3. Análisis de Resultados

3.1. Parte 1

3.1.1. Validación de la solución obtenida

Para cada instancia utilizada en las pruebas, el script ***parte-1.py*** imprime la instancia original y posteriormente una solución factible. La solución obtenida siempre respeta las tres restricciones del enunciado:

1. **Ninguna casilla queda vacía:** en la solución todos los valores son "O" o "X".
2. **Cada fila y cada columna contienen el mismo número de blancos y negros:** en todas las soluciones obtenidas se verifica que, en un tablero de tamaño n, cada fila y cada columna contienen exactamente $n/2$ discos blancos y $n/2$ discos negros.
3. **No existen tres discos consecutivos del mismo color:** se ha comprobado visualmente y mediante las restricciones ternarias que ninguna terna consecutiva (horizontal o vertical) toma los valores (0,0,0) o (1,1,1), cumpliéndo la regla de no consecutividad.

3.1.2. Metodología de pruebas y crecimiento del tiempo de ejecución

Para estudiar el comportamiento temporal del algoritmo y el efecto de los parámetros de entrada, se ha diseñado una batería de pruebas. Los casos de prueba se guardaron en la carpeta del proyecto ***parte-1/pruebas***, donde se almacenan los ficheros *.in* (***parte-1/pruebas/entradas***) y los *.out* (***parte-1/pruebas/salidas***) generados por el script. Además, dentro de ***parte-1/pruebas*** se ha creado un script (*lanzar_pruebas.py*), que recorre automáticamente todos los ficheros *.in* de */entradas*, ejecuta el resolvedor sobre cada uno, mide el tiempo de ejecución, recoge el número de soluciones, y lo guarda en ***parte-1/pruebas/tiempos*** para cada caso de prueba.

1) **Caso 1: Tableros pequeños 4 x 4 (***caso1_4x4_muy/medio/poco_informado.in***)**

En este primer caso se han utilizado tres instancias de tamaño 4×4 con distinto grado de información inicial: un **tablero muy informado** (con la mayoría de casillas fijadas), uno de **información intermedia** y otro claramente **poco informado**, con muchas posiciones marcadas como desconocidas.
En estas pruebas, el espacio de búsqueda es muy reducido y el tiempo de ejecución es prácticamente despreciable (milisegundos), y muy parecido para cada prueba. El número de soluciones suele ser bajo, a veces incluso único.
Tiempos obtenidos:

- Muy informado: ≈ 0.036010 segundos.
- Medio informado: ≈ 0.037270 segundos.
- Poco informado: ≈ 0.044598 segundos.

2) **Caso 2: Tableros medianos 6 x 6 (***caso2_6x6_muy/medio/poco_informado/vacio.in***)**

En este segundo caso, se han evaluado cuatro tableros de tamaño 6x6 con niveles de información inicial distribuidos de la misma manera que en el caso 1, además de incluir un tablero completamente vacío.
A diferencia del caso 4×4, aquí el tamaño del problema (36 casillas) hace que el espacio de búsqueda crezca de forma notable y las diferencias entre los tres niveles de información sean más visibles. El tablero **muy informado** se resuelve todavía en tiempos muy reducidos (< 100 ms), mientras que el tablero medio informado tarda algo más, y el tablero poco informado es claramente el más costoso (> 100 ms), al requerir explorar muchas más combinaciones antes de encontrar soluciones válidas. Si el tablero está completamente **vacío**, el tiempo de ejecución llega a ser superior al segundo.

Tiempos obtenidos:

- Muy informado: ≈ 0.038714 segundos.
- Medio informado: ≈ 0.049942 segundos.
- Poco informado: ≈ 0.346108 segundos.
- Vacío: ≈ 1.931550 segundos.

3) Caso 3: Tableros grandes 8 x 8, 10 x 10 (*caso3_8x8/10x10_poco/intermedio_informado.in***)**

En este caso se han analizado tableros 8×8 y 10×10 con distintos niveles de información inicial, observando que, a mayor tamaño y menor número de casillas fijadas, más crece el número de soluciones posibles y el espacio de búsqueda que el solver debe explorar. En los tableros más informados la propagación de restricciones poda muy rápido las configuraciones inválidas, mientras que en los tableros poco informados aparecen muchas más combinaciones factibles y ramas parciales, lo que se traduce en búsquedas significativamente más largas.

Tiempos obtenidos:

- 8x8 medio informado: ≈ 1.318217 segundos.
- 10x10 medio informado: ≈ 6.404589 segundos.
- 8x8 poco informado: ≈ 90.797347 segundos.

En estos resultados se resalta como un tablero 8×8 poco informado llega a tardar en torno a 90 segundos, y en el caso de 10×10, al reducir todavía más el número de casillas fijadas inicialmente, el tiempo de ejecución puede dispararse muy por encima de esos valores[1].

De esta forma, tras analizar los casos y tiempos de ejecución, podemos determinar que el **crecimiento del tiempo de ejecución** se debe principalmente por:

- **El crecimiento del número de variables** (n^2): Cada variable tiene 2 valores posibles (0 u 1), por lo que el espacio de búsqueda sin restricciones sería: 2^{n^2}. Obviamente el solver no explora todo este espacio gracias a las restricciones, pero el crecimiento sigue siendo exponencial.

[1] Nota: esta afirmación se ha comprobado al ejecutar un caso de prueba de un tablero 10x10 poco informado, cuya ejecución tras varios minutos seguía realizándose. Por ello no se ha decidido incorporar a la batería de pruebas.

- **La disminución de la información inicial** (cantidad de casillas fijadas inicialmente): **Sin pistas**, el solver debe explorar asignaciones casi completas (mucho más lento), mientras que **con muchas pistas**, el solver descarta gran parte del espacio muy rápido (mucho más rápido).

En el entorno de pruebas utilizado, y siempre que el tablero disponga de un nivel moderado o alto de información inicial, el algoritmo es capaz de resolver instancias de hasta $n = 10$ en tiempos razonables (< 30 segundos). No obstante, cuando el tamaño del tablero crece y, especialmente, cuando el número de casillas desconocidas es muy elevado, el tiempo de ejecución aumenta de forma muy rápida, lo que en la práctica impone un límite al tamaño y al grado de "poca información" que resulta viable abordar con este enfoque. De hecho, los resultados muestran que incluso con tamaños relativamente modestos (como 8×8 o 10×10), si las pistas iniciales son escasas, los tiempos pueden llegar a ser muy elevados, confirmando el carácter combinatorio y altamente explosivo del problema.

3.1.3. Número de variables y restricciones (Complejidad del modelo)

Desde el punto de vista de la modelización, para un tablero de tamaño $n \times n$:
El número de **variables** es exactamente: n^2, ya que se define una variable $x_{i,j}$ por cada casilla (i, j).
El número de **restricciones** se puede desglosar como sigue:

1. **Restricciones unarias (casillas fijadas)**: Si la instancia inicial contiene k casillas ya determinadas ("X" u "O"), entonces se añaden k restricciones unarias. Por lo que teóricamente habrá como máximo hasta n^2 restricciones (aunque realmente habrá muchos menos).
2. **Restricciones de suma exacta**: n restricciones por fila, n restricciones por columna.
   Total: $2n$
3. **Restricciones ternarias (no tres consecutivos)**:
   - Filas: $n \cdot (n-2)$
   - Columnas: $n \cdot (n-2)$

   Total: $2n(n-2)$

Total restricciones (sin contar unarias): $2n + 2n(n-2) = 2n(1+n-2) = 2n(n-1)$

En resumen, para un tamaño n, la magnitud de la instancia crece como:

- **Variables:** $O(n^2)$.
- **Restricciones:** $O(n^2)$ (dominadas por las $2n(n-2)$ restricciones ternarias), además de las unarias asociadas a las casillas fijas.

Aunque el número de restricciones crece cuadráticamente, la complejidad de la búsqueda sigue siendo exponencial en el peor caso, ya que el número de combinaciones posibles de un tablero BINAIRO crece de forma exponencial con n^2. Esto concuerda con los resultados empíricos obtenidos: al aumentar el tamaño del tablero o disminuir la cantidad de información inicial, el tiempo de

ejecución crece rápidamente y acaba por limitar el tamaño máximo de las instancias que se pueden resolver de forma práctica con este enfoque.

3.2. Parte 2

3.2.1. Contexto experimental

Para evaluar el comportamiento del algoritmo **A*** frente a un algoritmo de fuerza bruta (Dijkstra) se ejecutó una batería de **12 pruebas** sobre 9 mapas de la familia *USA-road-d* (DIMACS). En cada prueba se comparan: **coste óptimo** del camino, **nodos expandidos** y **tiempo de ejecución**. Los casos de prueba se guardaron en la carpeta del proyecto ***parte-2/pruebas***, donde se almacenan los ficheros *de resultados* (***parte-2/pruebas/resultados***) y los *de salida* (***parte-2/pruebas/salidas***) generados por el script *analisis.py*.
Aunque el script final *parte-2.py* recibe IDs de vértices, para las pruebas experimentales se seleccionaron orígenes/destinos mediante coordenadas de ciudades y se mapeó cada coordenada al nodo más cercano del grafo (búsqueda lineal sobre los vértices cargados). Esto permite construir casos realistas (rutas “entre ciudades”) sin conocer a priori IDs significativos.

3.2.2. Análisis de Resultados de Búsqueda

1. **Trayecto muy corto y nodo inexistente**
   (*Test_1,2(NY)_trayecto_muy_corto/_nodo_inexistente***)**

Descripción: tests iniciales para verificar el comportamiento en un caso **trivial** (origen y destino muy próximos) y la robustez ante **entradas inválidas** (se fuerza un destino con coordenadas fuera de rango). El resultado de la tabla muestra el caso trivial.

Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
NY	**794**	**1**	**0 s**	**1**	**0 s**	**0%**

Análisis: en el caso **trivial**, ambos algoritmos alcanzan la meta inmediatamente. La heurística no aporta ventaja porque no hay espacio real de exploración que recortar. En el caso **inválido**, se comprueba que la implementación finaliza con **error**.

2. **Caso base del enunciado (***Test_3(BAY)_caso_normal***)**

Descripción: caso pequeño y estándar del enunciado.

Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
BAY	**10.216**	**4**	**0 s**	**4**	**0 s**	**0,00%**

Análisis: en rutas muy cortas con pocas alternativas, A* y Dijkstra tienden a comportarse igual.

3. **Travesía montañosa e inversa (***Test_4,5(COL)_travesia_montanosa/_inversa***)**

Descripción: ruta larga con topología compleja (muchas ramificaciones por red vial y relieve).

Test	Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
Ida	COL	3.902.845	373.770	8.2494 s	384.504	7.7028 s	2,79%
Vuelta	COL	3.902.845	181.796	3.0073 s	195.246	2.8595 s	6,89%

Análisis: la heurística guía la búsqueda y reduce expansiones, pero la mejora depende del sentido: la red vial no es simétrica y, en un sentido, aparecen más ramificaciones de bajo coste que dispersan la exploración; en el inverso, la búsqueda queda más encauzada y se expande mucho menos.

4. **Rodeo por obstáculo natural (***Test_6(LKS)_rodeo_lago***)**

Descripción: el camino en línea recta geográfica cruzaría el Lago Michigan, pero la red obliga a rodear.

Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
LKS	4.076.213	1.009.276	49.6183 s	1.123.183	55.5661 s	10,14%

Análisis: este es el ejemplo más claro de por qué A* funciona: Dijkstra expande nodos siguiendo frentes de coste acumulado $g(n)$ y tiende a invertir expansiones en zonas que apuntan hacia el lago, donde no existe cruce posible. A*, al priorizar $f(n) = g(n) + h(n)$, penaliza nodos que no se acercan geométricamente al objetivo y encuentra antes el rodeo correcto.

5. **Corredor peninsular (***Test_7(FLA)_corredor_peninsular***)**

Descripción: ruta norte-sur relativamente lineal por la península.

Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
FLA	5.504.886	792.661	17.6481 s	801.341	15.6345 s	1,08%

Análisis: la red principal ya fuerza un corredor bastante directo. Cuando casi cualquier avance reduce la distancia al destino, la heurística discrimina poco: hay pocas alternativas “falsas” que se alejen manteniendo bajo coste.

6. **Larga distancia (***Test_8(CAL)_larga_distancia_resultado***)**

Descripción: trayecto largo (de unos 760 km) con varias autopistas principales y múltiples combinaciones.

Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
CAL	8.052.821	1.357.609	43.5894 s	1.386.520	40.5997 s	2,09%

Análisis: mejora moderada. Hay estructura jerárquica (arterias principales) que ayuda a A*, pero también hay muchas alternativas que siguen siendo prometedoras según la heurística, así que la poda no es enorme.

7. **Trayecto urbano y rural (***Test_9,10(NE/NW)_trayecto_urbano/_rural***)**

Descripción: comparación entre un entorno **urbano** (alta densidad de nodos y conexiones) y uno **rural** (red menos densa y más "encauzada" por carreteras principales).

Test	Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
Urbano	**NE**	**1.408.205**	**470.811**	**18.8744 s**	**486.164**	**18.1738 s**	**3,16%.**
Rural	**NW**	**4.430.296**	**682.606**	**20.6559s**	**734.036**	**20.8040 s**	**7,01%**

Análisis: en zona urbana hay muchas alternativas competitivas y la frontera crece más, por lo que A* solo recorta una parte moderada. En zona rural, al existir menos ramificación y caminos equivalentes, la heurística discrimina mejor y evita más desvíos no competitivos, logrando una mejora mayor.

8. **Grafo disconexo(***Test_11(ISLAS)_islas_desconectadas***)**

Descripción: se intenta conectar dos componentes sin conexión (sin puentes/ferries). Para ello se creó un mapa específico denominado *ISLAS-road-d*, ubicado en ***parte-2/mapas***, compuesto por dos islas no conectadas entre sí.

Resultados: **sin solución**; expansiones mínimas: **1**.

Análisis: el valor de este test es validar **completitud/robustez**: al agotarse la lista abierta sin alcanzar el destino, el algoritmo concluye correctamente que no existe camino.

9. **Extremo Norte–Sur (***Test_12(CAL)_extremo_norte_sur***)**

Descripción: conexión de extremos del estado, caso masivo (~millones de expansiones).

Mapa	Coste (m)	Nodos (A*)	Tiempo (A*)	Nodos (Dijkstra)	Tiempo (Dijkstra)	Mejora (%)
CAL	**13.321.188**	**1.802.903**	**61.1856 s**	**1.809.872**	**53.2406 s**	**0,39%**

Análisis: aquí la heurística "se satura": la ruta es predominantemente norte–sur y coincide bastante con la dirección geográfica al objetivo. Eso hace que la heurística apenas discrimine entre alternativas porque **casi cualquier progreso vial también reduce distancia Haversine**. Además, este test evidencia un punto clave: cuando la poda es mínima, el coste extra de calcular $h(n)$ puede hacer que A* sea **más lento** que Dijkstra aunque expanda ligeramente menos nodos.

3.2.3. Análisis del impacto de la heurística

El objetivo principal de utilizar A* frente a algoritmos ciegos como Dijkstra es la reducción del espacio de búsqueda. Los resultados muestran una variabilidad notable en esta reducción,

dependiente de la topología del mapa y de la "directividad" del trayecto:

Escenarios de alto impacto (obstáculos geográficos): El Test 6 (Great Lakes) demuestra el mayor beneficio de la heurística, con una reducción del 10,14% en nodos expandidos. El trayecto cruza el Lago Michigan. Dijkstra expande nodos siguiendo frentes de coste acumulado $g(n)$, explorando innecesariamente cientos de miles de nodos hacia la orilla del lago, donde no hay cruce posible. En contraste, A* incorpora el término heurístico $h(n)$ (distancia Haversine al destino) para penalizar la exploración de nodos que no se acercan geométricamente al objetivo, favoreciendo rápidamente la ruta que rodea el obstáculo por el sur.

Escenarios de bajo impacto (rutas lineales): El Test 12 (California Norte–Sur) representa el "peor caso" para la heurística. La ruta sigue la costa del Pacífico en una trayectoria casi rectilínea norte–sur. En este escenario, los nodos con menor coste total $f(n) = g(n) + h(n)$ resultan muy similares a los priorizados solo por $g(n)$, ya que avanzar por la carretera también reduce, en general, la distancia geodésica (Haversine) al destino. La poda es mínima (0,39%) porque apenas existen "caminos falsos" que se alejen del destino y mantengan un coste acumulado bajo.

3.2.4. Crecimiento de las estructuras de datos y complejidad

Memoria: El tamaño de las estructuras de datos propias de A* (lista abierta, conjunto cerrado y diccionarios auxiliares como *g_score* y *came_from*) crece aproximadamente de **forma lineal con el número de vértices descubiertos/expandidos** (en el peor caso, $O(|V|)$). En nuestra prueba de carga máxima (Test 12), el algoritmo gestionó 1.802.903 nodos sin agotar la memoria disponible, lo que sugiere que la implementación es viable para grafos de escala real (millones de vértices), donde el **consumo total** está dominado por la propia representación del grafo ($O(|V| + |E|)$).

Tiempo de ejecución: El tiempo crece con el número de expansiones y con el coste de seleccionar en cada iteración el nodo con menor $f(n)$ en la lista abierta. En nuestra implementación, esa extracción se realiza mediante **búsqueda lineal** del mínimo, por lo que cada *pop()* cuesta $O(|V|)$ en el peor caso; acumulado a lo largo de la búsqueda, el **coste global** queda acotado por $O(|V|^2 + |E|)$. Empíricamente el tiempo está influido por el número de expansiones, pero no depende solo de él: en A* también influye el coste de calcular la heurística Haversine en cada expansión. Por ello, una reducción moderada de nodos no siempre se traduce en una mejora temporal.

Overhead de la heurística (impacto en tiempo): Aunque A* expande menos nodos que Dijkstra en todos los tests con solución, en varias instancias el **tiempo total resulta ligeramente mayor** (habitualmente por unos segundos, y en el caso más extremo por varios segundos). La causa es que A* paga un coste adicional en cada expansión: el cálculo de la heurística Haversine, que puede compensar o incluso superar la pequeña reducción de nodos cuando la poda es baja (p. ej., sólo

0,39% en el Test 12). En cambio, cuando la heurística sí recorta una fracción apreciable del espacio de búsqueda (p. ej., Test 6 Great Lakes, 10,14%), A* reduce también el tiempo (49,618 s vs 55,566 s).

3.2.5. Validación de la solución óptima

En todos los tests con solución, el coste de A* coincide con el de Dijkstra, lo que valida empíricamente la optimalidad del resultado. Esto confirma la admisibilidad de la heurística utilizada (distancia geodésica Haversine): la distancia en línea recta sobre la superficie terrestre entre dos coordenadas es siempre menor o igual que la distancia real recorrida por carretera. Por tanto, $h(n)$ no sobreestima el coste restante y A* garantiza que no descarta el camino óptimo.

4. Conclusión

La realización de esta práctica ha permitido cumplir los objetivos formativos planteados en la introducción, ofreciendo una visión completa de dos estrategias fundamentales de resolución de problemas: el **modelado mediante Satisfacción de Restricciones (CSP)** y la **búsqueda óptima en grafos mediante algoritmos heurísticos**.

El desarrollo de la **Parte 1** ha demostrado que **BINAIRO** puede modelarse de forma natural como un CSP con variables binarias y restricciones globales, obteniendo soluciones válidas que satisfacen todas las condiciones del enunciado. El análisis experimental confirma, además, el comportamiento esperado: el tiempo de ejecución crece rápidamente cuando aumenta el tamaño del tablero y, sobre todo, cuando disminuye la información inicial, reflejando la naturaleza combinatoria del problema y la importancia de la propagación de restricciones para podar el espacio de búsqueda.

En la **Parte 2**, se ha abordado el cálculo de rutas en mapas viarios reales (DIMACS) comparando el algoritmo heurístico (**A* con heurística Haversine**) con un método de referencia de fuerza bruta (**Dijkstra**). Los resultados verifican la **optimalidad** del coste en todos los casos con solución y muestran que la ganancia práctica del algoritmo heurístico suele ser **moderada**: la reducción de nodos expandidos depende fuertemente de la topología (más notable con obstáculos y mínima en rutas casi lineales) y, además, en varias instancias el menor número de expansiones no implica menor tiempo debido al coste adicional de calcular la heurística.

En resumen, la práctica muestra que el uso de heurística en A* mantiene la corrección y puede reducir el espacio de búsqueda, aunque su beneficio depende de la topología del problema y del coste de calcular la propia heurística. Del mismo modo, en CSP se confirma que la calidad del modelado y la información inicial condicionan la viabilidad práctica. En conjunto, se ha observado cómo decisiones de diseño (restricciones, heurística y estructuras de datos) afectan directamente a la escalabilidad y al rendimiento.