Facsímil 02

Inteligencia clásica

Capítulo 03

Greedy, A* y heurísticas: buscar con estimaciones

686f6c61 · 2026

Capítulo 03: Greedy, A* y heurísticas: buscar con estimaciones

Entrando en el tema

Hasta ahora has explorado a ciegas. BFS, DFS, UCS: algoritmos que no saben nada del problema salvo qué acciones existen y cuánto cuestan. Funcionan, pero son terriblemente ineficientes: exploran miles de estados que no llevan a ninguna parte.

Ahora imagina que tienes una estimación matemática barata. Una función $h(n)$ que, para cualquier estado n, estima cuánto falta para llegar a la meta. No es exacta, porque si lo fuera ya habrías resuelto el problema, pero es útil. Con esa estimación, puedes priorizar los estados que *parecen* más prometedores e ignorar los que se alejan. Puedes encontrar la solución explorando cientos de estados en vez de millones.

Esa función se llama **heurística**. Y los algoritmos que la usan, Greedy y A*, son piezas centrales de la búsqueda informada.[1] Sin heurísticas, muchos problemas de rutas, planificación y videojuegos tendrían que mirar demasiadas opciones antes de responder.

Greedy best-first: seguir solo la estimación

El algoritmo *greedy best-first search* es el más simple de los informados. Evalúa cada estado exclusivamente con la heurística $h(n)$, la estimación de lo que falta hasta la meta, e ignora completamente el coste acumulado $g(n)$.[2]

Es como ir de Madrid a Berlín mirando exclusivamente la distancia en línea recta: «Barcelona está más cerca de Berlín que Lisboa, voy a Barcelona. París está más cerca que Roma, voy a París». Nunca mira hacia atrás. Nunca considera si el camino acumulado es bueno o si la ruta aparente esconde un desvío enorme.

Formalmente, Greedy usa esta función de evaluación:

$$f_{\text{Greedy}}(n) = h(n)$$

Y en cada iteración extrae de la frontera el nodo que parece más cercano a la meta:

$$n_t = \arg\min_{n \in F_t} h(n)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$f_{\text{Greedy}}(n)$	Prioridad que Greedy asigna al nodo n.	Si $h(B) = 3$, la prioridad de B es 3.
$h(n)$	Estimación de coste desde n hasta la meta.	Distancia Manhattan hasta la salida.
F_t	Frontera en la iteración t.	$\{B, C, D\}$.
$\arg\min$	Operación que devuelve el elemento con menor valor.	Si $h(C) = 2$, Greedy elige C.

Mira el detalle peligroso: no aparece $g(n)$. Si llegar hasta C ya te ha costado 40 pasos y llegar hasta B solo 2, Greedy no lo ve. Solo pregunta: «¿cuál parece más cerca de la meta desde aquí?».

Ventaja: velocidad. Si $h(n)$ es razonablemente buena, Greedy encuentra una solución explorando muy pocos estados. En navegación con distancia euclídea, suele ir directo al destino.

Riesgo: no es ni completo ni óptimo. Puede quedarse atascado en un mínimo local, eligiendo repetidamente estados que *parecen* buenos según h pero que no llevan a ninguna parte. Y el camino que encuentra rara vez es el mejor: la heurística puede empujarte hacia una montaña, porque en línea recta parece más corta, cuando el camino óptimo da un rodeo por el valle.[3]

En agentes LLM modernos, Greedy equivale a elegir la *tool* que parece obvia sin verificar restricciones: «¿cuál es la respuesta? Voy a buscar en la base de datos». Pero quizás antes necesitabas comprobar permisos. La heurística te empujó en una dirección que parecía correcta pero no lo era.

La tabla de propiedades queda así:

PROPIEDAD	VALOR	POR QUÉ
Completitud	No en general	Puede entrar en ciclos o perseguir una rama infinita que parece prometedora.
Optimalidad	No	Ignora el coste acumulado $g(n)$.
Tiempo	$O(b^m)$ en el peor caso	Si la heurística engaña, puede explorar una rama profunda entera.
Espacio	$O(b^m)$ en el peor caso	Mantiene frontera y visitados como otros algoritmos de búsqueda en grafos.

A*: coste real + estimación

A* corrige el defecto fundamental de Greedy combinando dos piezas de información en una sola función de evaluación:[4]

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

SÍMBOLO	SIGNIFICADO	CÁLCULO
$f(n)$	Coste total estimado del camino óptimo que pasa por n	$g(n) + h(n)$
$g(n)$	Coste real acumulado desde el inicio hasta n	$\sum c(s_{i-1}, a_i)$
$h(n)$	Heurística: coste estimado desde n hasta la meta	Específica del problema
$h^*(n)$	Coste real óptimo desde n hasta la meta	Desconocido (es lo que buscamos)

$g(n)$ es lo que ya has pagado. $h(n)$ es lo que estimas que te queda. $f(n)$ es el total estimado. A* expande siempre el nodo con menor $f(n)$.[5]

La política de extracción de A* es:

$$n_t = \arg\min_{n \in F_t} (g(n) + h(n))$$

NODO	$G(N)$: COSTE YA PAGADO	$H(N)$: ESTIMACIÓN RESTANTE	$F(N) = G(N) + H(N)$	QUIÉN LO ELEGIRÍA
B	8	2	10	Greedy, porque $h(B)$ es menor.
C	3	4	7	A*, porque $f(C)$ es menor.

Este ejemplo pequeño captura toda la diferencia. Greedy ve $2 < 4$ y elige B. A* ve $10 > 7$ y elige C, porque entiende que lo que ya has pagado también cuenta.

Esto resuelve el problema de Greedy. Si la heurística te empuja hacia un camino que *parece* corto pero es caro, $g(n)$, el coste real acumulado, penaliza esa decisión. A* no solo mira lo prometedor: también penaliza los caminos que ya han costado demasiado.[6]

Propiedades formales de A*

Admisibilidad. Una heurística $h(n)$ es admisible si nunca sobreestima el coste real hasta la meta. Formalmente:

$$0 \leq h(n) \leq h^*(n)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$h(n)$	Estimación que calculas rápido.	Distancia en línea recta: 12 km.
$h^*(n)$	Coste real óptimo desde n hasta la meta.	Mejor carretera real: 15 km.
$0 \leq h(n)$	La heurística no puede ser negativa.	No tiene sentido decir “faltan -3 km”.
$h(n) \leq h^*(n)$	La heurística no promete más de lo que existe.	12 km no sobreestima 15 km.

La distancia en línea recta es admisible porque ningún camino por carreteras puede ser más corto que la línea recta. Una heurística que estime tiempos ignorando semáforos, cuestas o peajes puede dejar de ser admisible si promete rutas demasiado optimistas.[7]

Teorema de optimalidad. En búsqueda en árbol, si $h(n)$ es admisible, A* encuentra un camino de coste mínimo. En búsqueda en grafo, la garantía requiere además gestionar correctamente reaperturas de estados o trabajar con una heurística consistente. La demostración se basa en que A* no termina hasta haber descartado los nodos que podrían tener $f(n) < C^*$ (el coste óptimo) y, cuando encuentra la meta, su f es igual a g porque $h(\text{meta}) = 0$.

Consistencia (monotonicidad). Una propiedad más fuerte que la admisibilidad:

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

SÍMBOLO	SIGNIFICADO	EJEMPLO
$c(n, a, n')$	Coste de ir de n a n' aplicando a.	Moverte una casilla cuesta 1.
$h(n)$	Estimación antes de moverte.	Faltan 8 pasos estimados.
$h(n')$	Estimación después de moverte.	Faltan 7 pasos estimados.

La lectura es sencilla: la estimación desde n no puede ser mayor que «lo que cuesta dar un paso» más «lo que estimas desde el siguiente estado». Si h es consistente, entonces $f(n)$ nunca baja a lo largo de un camino:

$$f(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

Por eso A* con heurística consistente puede tratar los estados como cerrados la primera vez que los expande: no necesitará reabrirlos más tarde con un coste mejor.[8]

Un ejemplo trazado de A*

Veamos A* completo sobre un grafo pequeño. Cuatro estados, con estos costes y esta heurística admisible:

- A → B (coste 1), A → C (coste 4)
- B → D (coste 5), C → D (coste 1)
- $h(A) = 4$, $h(B) = 4$, $h(C) = 1$, $h(D) = 0$

El estado inicial es A y la meta es D. A* expande siempre el nodo de menor $f = g + h$:

PASO	SE EXTRAE (F)	FRONTERA RESULTANTE
1	A, $f = 0 + 4 = 4$	B ($f = 5$), C ($f = 5$)
2	B, $f = 1 + 4 = 5$	C ($f = 5$), D ($f = 6$)
3	C, $f = 4 + 1 = 5$	D ($f = 5$, actualizado desde 6)
4	D, $f = 5$	meta: camino A → C → D, coste 5

Fíjate en el paso 3. Al expandir B, A* descubre D con $f = 6$, por el camino A → B → D, que cuesta 6. Pero al expandir C encuentra un camino mejor a D, con $f = 5$, por A → C → D, que cuesta 5, y actualiza su coste. Cuando por fin saca D, lo hace con el coste óptimo. Greedy, que solo mira h, habría ido directo a C y luego a D sin dudar; A* da el mismo resultado aquí, pero porque ha tenido en cuenta g, no por suerte. Es la misma traza que puedes reproducir en el cuaderno del facsímil.

Heurísticas: el arte de saber qué ignorar

Una heurística es una función $h : S \to \mathbb{R}_0^+$ que, dado un estado, devuelve una estimación. Diseñar una buena heurística exige escoger información que aporte señal, sea barata de calcular y no rompa las garantías que quieres conservar.[9]

En una rejilla, dos heurísticas clásicas son:

$$h_{\text{Manhattan}}(n) = |x_n - x_{\text{meta}}| + |y_n - y_{\text{meta}}|$$

$$h_{\text{Euclidea}}(n) = \quad (x_n - x_{\text{meta}})^2 + (y_n - y_{\text{meta}})^2$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x_n, y_n	Coordenadas del estado actual.	$n = (2, 3)$.
$x_{\text{meta}}, y_{\text{meta}}$	Coordenadas de la meta.	$\text{meta} = (7, 6)$.
$	x_n-x_{\text{meta}}	$
$	y_n-y_{\text{meta}}	$

Con esos números:

$$h_{\text{Manhattan}}(n) = 5 + 3 = 8$$

$$h_{\text{Euclidea}}(n) = \quad 5^2 + 3^2 = \quad 34 \approx 5.83$$

Si solo puedes moverte en horizontal y vertical, Manhattan suele ser más informativa. Si puedes moverte en cualquier dirección, la euclídea encaja mejor.

TIPO	EJEMPLO	ADMISIBLE	INFORMATIVA
Buena	Distancia euclídea en navegación	Sí	Alta
Aceptable	Distancia Manhattan en rejilla	Sí	Media
Mala	«Dificultad del examen = páginas del temario»	No	Baja
Engañosa	«Número de paquetes que faltan»	No	Engaña

La diferencia entre una heurística buena y una mala puede ser la diferencia entre explorar 100 estados y 100 000. Una heurística perfecta, con $h(n) = h^*(n)$, haría que A* fuera directamente a la solución sin explorar nada más, pero calcular $h^*(n)$ es tan difícil como resolver el problema original. El arte está en aproximar $h^*(n)$ sin calcularla exactamente.

Una técnica clásica es **relajar el problema**: eliminar alguna restricción para crear una versión más fácil. La solución del problema relajado es una heurística admisible para el problema original.[10] Por ejemplo, la distancia en línea recta es la solución al problema de navegación relajado donde ignoras los obstáculos y las carreteras.

También comparamos heurísticas por **dominancia**:

$$h_1 \succeq h_2 \quad \Longleftrightarrow \quad \forall n,\ h_1(n) \geq h_2(n)$$

siempre que ambas sigan siendo admisibles. Si h_1 domina a h_2, A* con h_1 no expande más nodos que A* con h_2. Dicho en castellano: cuanto más cerca esté $h(n)$ de $h^*(n)$ sin pasarse, menos trabajo hace A*.

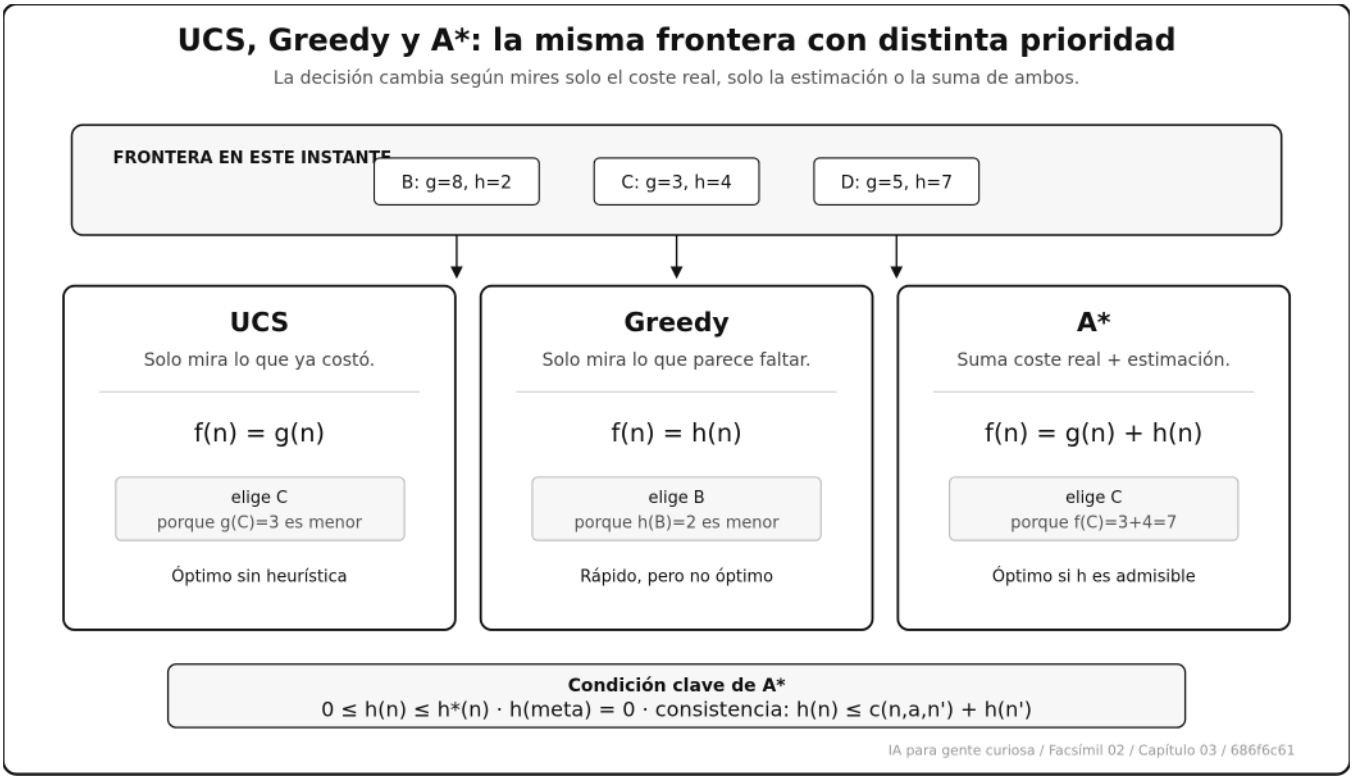
Auditar una heurística antes de usarla

Una heurística no se acepta porque “suena razonable”. Se audita. Para problemas pequeños, puedes calcular $h^*(n)$, el coste óptimo real desde cada estado hasta la meta, y comparar.

PRUEBA	FÓRMULA	QUÉ DETECTA
No negatividad	$h(n) \geq 0$	Estimaciones sin sentido.
Meta a cero	$h(\text{meta}) = 0$	Heurísticas que penalizan llegar.
Admisibilidad	$h(n) \leq h^*(n)$	Sobreestimaciones que rompen optimalidad de A*.
Consistencia	$h(n) \leq c(n, a, n') + h(n')$	Necesidad de reabrir nodos en grafos.
Dominancia	$h_1(n) \geq h_2(n)$ para todo n	Qué heurística informará más a A* sin perder garantías.

En producción no siempre puedes conocer $h^*(n)$; si pudieras, ya tendrías resuelto el problema. Pero en entornos pequeños, tests, mapas de juguete o fixtures, esta auditoría es oro: te permite validar que tu heurística no está metiendo una promesa falsa en el algoritmo.

También conviene medir el coste de calcular $h(n)$. Una heurística brillante pero carísima puede perder contra una heurística sencilla si cada evaluación tarda demasiado. A* no solo paga expansiones; también paga evaluaciones de heurística.



Variantes de A*: cuando la memoria o la velocidad aprietan

A* es óptimo, pero tiene un punto débil que comparte con BFS: la memoria. A* guarda en la frontera y en los visitados todos los nodos que descubre, así que en el peor caso necesita espacio exponencial, del orden de $O(b^d)$. En problemas grandes, A* se queda sin memoria mucho antes que sin tiempo. De ese problema nacen dos variantes habituales.

IDA* (A* con profundización iterativa). Aplica a A* la idea del IDS del capítulo anterior. En lugar de un límite de profundidad, usa un límite sobre f. Hace una búsqueda en profundidad podando toda rama cuyo $f(n)$ supere el umbral; si no encuentra la meta, sube el umbral al menor f que se pasó del límite y repite. Conserva la optimalidad de A* con heurística admisible, pero su memoria baja a $O(b \cdot d)$, la de DFS. El precio, como en IDS, es reexplorar niveles.

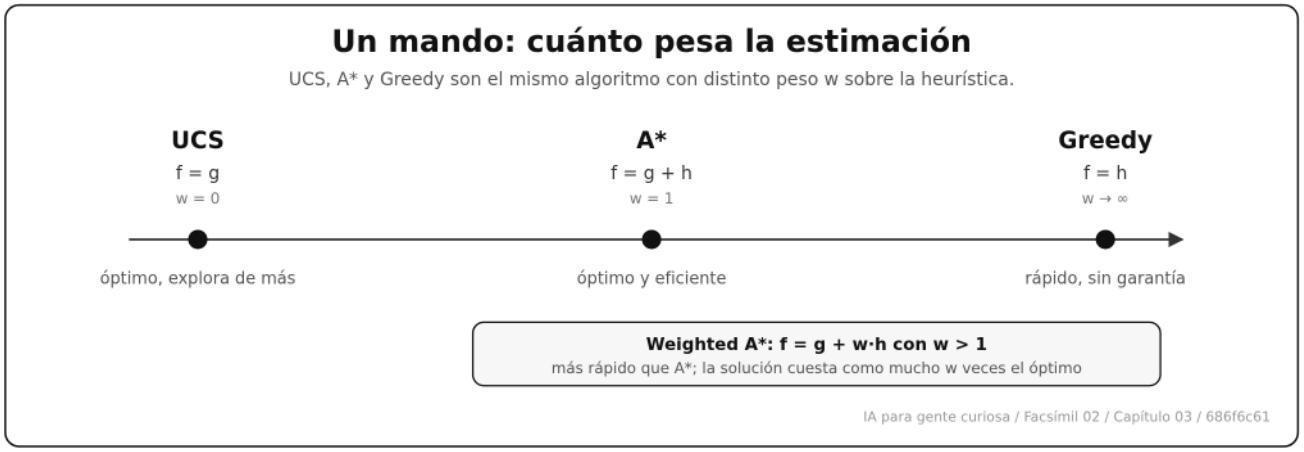
Weighted A*. Da más peso a la heurística para correr más, a cambio de renunciar a la optimalidad garantizada:

$$f(n) = g(n) + w \cdot h(n), \qquad w \geq 1$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
w	Peso que se da a la heurística.	$w = 1$ es A*; $w = 2$ confía el doble en la estimación.
$g(n)$	Coste real acumulado.	Lo que ya has pagado.
$h(n)$	Heurística admisible.	Lo que estimas que falta.
$w \cdot h(n)$	Estimación amplificada.	Empuja la búsqueda hacia la meta más agresivamente.

En palabras: cuanto mayor es w, más se parece A* a Greedy y menos nodos expande, pero la solución deja de ser óptima. La buena noticia es que el desvío está acotado: con una heurística admisible, Weighted A* devuelve un camino que cuesta como mucho w veces el óptimo. Es un mando con el que cambias calidad por velocidad de forma controlada: en $w = 1$ tienes A* y la solución óptima, y subiendo w ganas velocidad aceptando soluciones algo peores.

De hecho, los tres algoritmos del capítulo, junto a UCS, son el mismo esquema con distinto peso sobre la heurística:



En el día a día

En **GPS y navegación**, A* con $h(n)$ igual a la distancia euclídea es una simplificación útil para entender la idea. Los sistemas reales añaden datos de tráfico, restricciones de giro, jerarquías de carretera y cachés, pero el mecanismo permanece: una heurística buena reduce mucho el espacio que hay que mirar.

En **videojuegos**, el *pathfinding* de los personajes no jugadores usa A* con distancia Manhattan para moverse por la rejilla del escenario. Sin una búsqueda informada, los personajes se quedarían atascados contra las paredes o tomarían rutas absurdas que rompen

la sensación de inteligencia.

En **agentes LLM**, una decisión de herramienta puede modelarse como un paso de búsqueda. El modelo propone qué herramienta parece más prometedora, que es el papel de $h(n)$, pero el sistema debería añadir coste acumulado, riesgo, permisos y observación. Si penalizas los caminos que ya han consumido demasiados tokens o llamadas caras, estás metiendo una idea parecida a $g(n)$ en el diseño, justo lo que separa a A* de Greedy.

Por qué debería importarte

A* es uno de los algoritmos centrales de la IA clásica: combina coste real y estimación futura de una forma sorprendentemente clara. La ecuación $f(n) = g(n) + h(n)$ resume décadas de investigación. Pero la lección más profunda no es memorizar el algoritmo: es aprender a diseñar **heurísticas** que ahorran búsqueda sin romper las garantías que necesitas.

Y este concepto trasciende la IA: en optimización, en diseño de algoritmos, en toma de decisiones, la habilidad de encontrar atajos informados que no comprometan la calidad de la solución es universal. A* formaliza esa tensión entre coste observado y estimación restante.

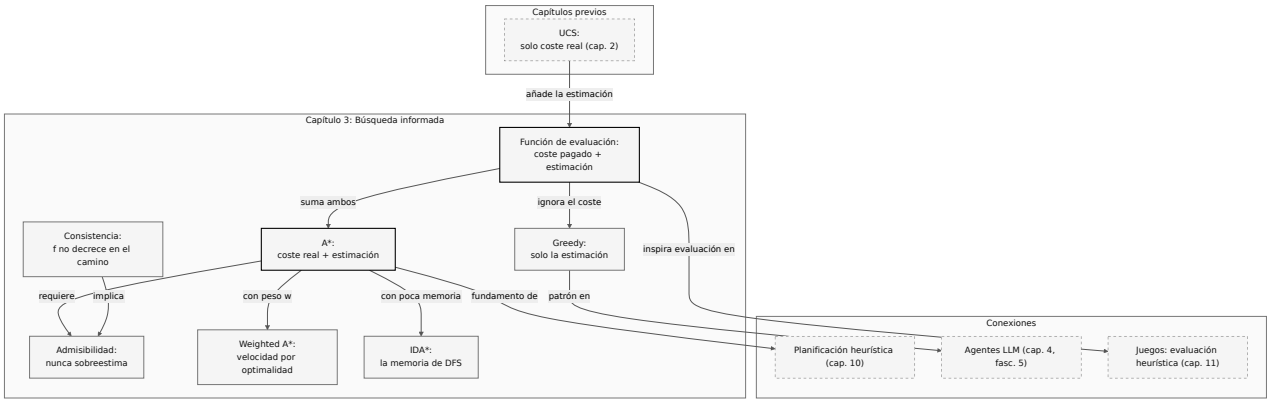
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Heurística no admisible con A*	Si $h(n)$ sobreestima el coste real, A* puede pasar de largo la solución óptima.	Verifica $0 \leq h(n) \leq h^*(n)$. La distancia euclídea y Manhattan son admisibles. Las heurísticas basadas en «coste medio» no lo son.
Confundir Greedy con A*	Greedy ignora $g(n)$: no garantiza optimalidad. A* usa $g(n) + h(n)$: sí la garantiza con h admisible.	Usa A* si necesitas optimalidad. Greedy solo si necesitas una solución rápida y la calidad no es crítica.
Heurística demasiado cara de calcular	Evaluar $h(n)$ cuesta tiempo. Si calcular h tarda más que expandir unos cientos de nodos extra con una heurística más simple, estás perdiendo eficiencia neta.	Mide el tiempo de $h(n)$. Si es más lento que expandir ~100 nodos, simplifica la heurística.

Cómo encaja todo

Este capítulo añade una pieza nueva a la frontera del capítulo 02: ya no ordenas solo por antigüedad o por coste real acumulado. Ahora introduces una estimación del futuro. Esa estimación puede ahorrar muchísimo trabajo, pero solo mantiene garantías si se comporta matemáticamente bien.

La idea seguirá viva en planificación, juegos y agentes: muchas decisiones inteligentes son una mezcla de coste pagado, coste esperado, riesgo y una función de evaluación.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
A*	$f(n) = g(n) + h(n)$. Óptimo si h es admisible.
Heurística admisible	Nunca sobreestima: $0 \leq h(n) \leq h^*(n)$.
Greedy best-first	Solo $h(n)$. Rápido, no óptimo.
Consistencia	$h(n) \leq c(n, a, n') + h(n')$. Más fuerte que admisibilidad.
Relajación	Simplificar el problema para obtener una heurística admisible.
Dominancia heurística	Una heurística admisible domina a otra si estima siempre igual o más sin sobreestimar. Suele reducir expansiones.
Weighted A*	Variante $f(n) = g(n) + wh(n)$. Con $w > 1$ puede ser más rápida, pero pierde optimalidad garantizada (coste hasta w veces el óptimo).
IDA*	A* con profundización iterativa sobre f. Memoria $O(b \cdot d)$ como DFS, manteniendo la optimalidad con h admisible.

Antes de pasar página

- [ ] ¿Puedo escribir $f(n) = g(n) + h(n)$ y explicar cada término? (Si no, vuelve a «A*: coste real + estimación».)
- [ ] ¿Entiendo qué significa que h sea admisible? (Si no, vuelve a «Propiedades formales de A*».)
- [ ] ¿Sé diferenciar Greedy de A*? (Si no, vuelve a «Greedy best-first» y «A*: coste real + estimación».)
- [ ] ¿Puedo auditar una heurística con admisibilidad, consistencia y dominancia? (Si no, vuelve a «Auditar una heurística antes de usarla».)
- [ ] ¿Entiendo por qué A* consume memoria exponencial y qué resuelve IDA*? (Si no, vuelve a «Variantes de A*».)
- [ ] ¿Sé qué hace Weighted A* con el peso w y qué garantía pierde y conserva? (Si no, vuelve a «Variantes de A*».)

- [ ] ¿Sé explicar por qué Weighted A* puede perder optimalidad? (Si no, vuelve a «Variantes de A*: cuando la memoria o la velocidad aprietan».)

En resumen

IDEA FUERZA	DETALLE
A* = UCS + heurística.	$g(n)$ garantiza optimalidad; $h(n)$ añade eficiencia.
Admisibilidad abre la puerta a la optimalidad.	$h(n) \leq h^*(n)$ es la condición suficiente que A* necesita en búsqueda en árbol; en grafos, la consistencia evita reexpansiones.
La heurística lo es todo.	De millones de estados a cientos. El arte está en el equilibrio entre precisión y coste.
Las heurísticas también se testean.	No basta con una estimación plausible: admisibilidad, consistencia, dominancia y coste de cálculo son parte del contrato.

Para saber más

Hart, P. E., Nilsson, N. J. y Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107. https://doi.org/10.1109/TSSC.1968.300136

Luger, G. F. (2008). *Artificial intelligence: structures and strategies for complex problem solving* (6.ª ed.). Pearson.

Nilsson, N. J. (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley.

Poole, D., Mackworth, A. y Goebel, R. (1998). *Computational intelligence: a logical approach*. Oxford University Press.

Rich, E., Knight, K. y Nair, S. B. (2009). *Artificial intelligence* (3.ª ed.). McGraw-Hill.

Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. https://aima.cs.berkeley.edu/

Notas

1. Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley. Pearl estableció las bases teóricas de la búsqueda heurística, formalizando conceptos como admisibilidad, consistencia y poder heurístico que permiten comparar rigurosamente distintas heurísticas. ↩

2. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. La sección 3.5 presenta *greedy best-first* como el primer algoritmo informado, destacando que su velocidad tiene como contrapartida la ausencia de garantías de optimalidad. ↩

3. Nilsson, N. J. (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann. La sección 8.2 analiza las limitaciones de Greedy y demuestra con ejemplos concretos por qué ignorar el coste del camino puede llevar a soluciones arbitrariamente malas. ↩

4. Hart, P. E., Nilsson, N. J. y Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107. https://doi.org/10.1109/TSSC.1968.300136. Este artículo presentó A* y demostró formalmente que, con heurística admisible, el algoritmo es óptimo y expande el mínimo número de nodos entre todos los algoritmos óptimos que usan la misma heurística. ↩

5. Poole, D., Mackworth, A. y Goebel, R. (1998). *Computational intelligence: a logical approach*. Oxford University Press. ↩

6. Luger, G. F. (2008). *Artificial intelligence: structures and strategies for complex problem solving* (6.ª ed.). Pearson. ↩

7. Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley. El capítulo 3 demuestra el teorema de optimalidad de A*: si h es admisible, A* es óptimo en búsqueda en árbol. ↩

8. Rich, E., Knight, K. y Nair, S. B. (2009). *Artificial intelligence* (3.ª ed.). McGraw-Hill. ↩

9. Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley. Pearl introdujo el concepto de poder heurístico: una heurística h_1 domina a h_2 si $h_1(n) \geq h_2(n)$ para todo n, y una heurística más informada produce una búsqueda más eficiente. ↩

10. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. La sección 3.6 explica cómo derivar heurísticas admisibles a partir de versiones relajadas del problema, siendo la distancia en línea recta el ejemplo canónico (ignorar obstáculos es la relajación). ↩