2. Búsqueda en Espacio de Estados

2.1 Conceptos Principales

Para resolver problemas en IA, formalizamos el mundo como un conjunto de estados y transiciones.

1. **Espacio de Estados**: Conjunto de todos los estados alcanzables desde el estado inicial mediante una secuencia de operadores
2. **Estado Inicial**: Descripción de partida del sistema
3. **Operadores (Acciones)**: Reglas que transforman un estado en otro (ej. mover una pieza, mover el hueco en un puzle)
4. **Prueba de Meta (Goal Test)**: Condición que determina si un estado es la solución
5. **Costo de Ruta** ($g(n)$): Costo acumulado desde el inicio hasta el estado actual (ej. número de movimientos, distancia en km)

La Función de Evaluación ($f(n)$): Es una fórmula matemática fundamental para guiar la búsqueda, decidiendo qué nodo es el más "prometedor" para expandir a continuación. Asigna un valor numérico a cada nodo que representa la estimación de cuán bueno es ese camino para llegar a la solución. Permite ordenar los nodos en la "frontera" de búsqueda. El algoritmo siempre elegirá expandir el nodo con el mejor valor $f(n)$ (generalmente el menor valor si hablamos de costes).

Componentes típicos:

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

- $g(n)$: Lo que ya has recorrido (costo real desde el inicio hasta n)
- $h(n)$: Lo que te falta (estimación heurística desde n hasta la meta)

2.2 Estrategias de Búsqueda: Paso a Paso

Las estrategias se dividen en **Ciegas** (sin información del dominio) y **Heurísticas** (con información/estimaciones).

> **Heurística:** Es un criterio basado en conocimiento del problema que ayuda a seleccionar los operadores o acciones más prometedoras, descartando opciones poco útiles.
> **Una heurística es como un "atajo inteligente" que evita búsquedas exhaustivas.**

2.2.1 Búsqueda a Ciegas (No Informada)

Las **estrategias de búsqueda a ciegas** (no informadas) exploran posibles soluciones sin información adicional sobre cuál camino puede ser mejor. Se utilizan principalmente en problemas donde no se tiene una **heurística** clara para guiar la búsqueda.

- **Búsqueda completa:** garantiza encontrar una solución si existe.
- **Búsqueda óptima:** garantiza encontrar la mejor solución disponible.
- **Complejidad temporal:** número total de nodos explorados durante la búsqueda.
- **Complejidad espacial:** número máximo de nodos almacenados en memoria simultáneamente

Símbolos:

- **r:** factor de ramificación (promedio de sucesores por nodo)
- **p:** profundidad de la solución
- **m:** profundidad máxima
- **l:** límite de profundidad

Búsqueda en Amplitud (Breadth-First Search - BFS)

Funcionamiento: Explora nivel por nivel. Primero visita el nodo raíz, luego todos sus hijos, luego los nietos, etc.

Estructura de datos: Usa una lista ABIERTA como cola FIFO (Primero en entrar, primero en salir)

Propiedades:

- **Completa**: Sí, encuentra la solución si existe (en espacios finitos)
- **Óptima**: Sí, pero solo si el costo de los operadores es uniforme
- **Problema**: Consume muchísima memoria ($O(r^p)$) porque guarda todos los nodos del nivel actual

Búsqueda en Profundidad (Depth-First Search - DFS)

Funcionamiento: Explora una rama hasta el final antes de retroceder. Si llega a un punto muerto, vuelve atrás.

Estructura de datos: Usa una lista ABIERTA como pila LIFO (Último en entrar, primero en salir)

Propiedades:

- **Completa**: No (puede caer en bucles infinitos o ramas infinitas)
- **Óptima**: No (puede encontrar una solución muy profunda antes que una corta)

- **Ventaja**: Muy eficiente en memoria ($O(r \cdot m)$), solo guarda la rama actual

Otras Búsquedas

- **En profundidad limitada.** Se establece un límite máximo de profundidad para la exploración. Evita ciclos infinitos, pero puede perder soluciones si el límite es bajo.
- **En profundidad iterativa.** Aplica búsqueda en profundidad con límites crecientes, combinando ventajas de amplitud y profundidad. Es completa y óptima (en espacios finitos).
- **Bidireccional.** Realiza la búsqueda simultáneamente desde el estado inicial y desde el objetivo, esperando que ambas se encuentren. Reduce la complejidad temporal y espacial.

Aquí está la tabla transcrita a formato Markdown:

Estrategia	Completa	Óptima	Tiempo	Espacio
Amplitud	Sí	Sí	O(r^p)	O(r^p)
Profundidad	No*	No	O(r^m)	O(r*m)
Prof. limitada	No*	No	O(r*l)	O(r*l)
Iterativa	Sí	Sí	O(r^p)	O(r^p)
Bidireccional	Sí	Sí	O(r^(p/2))	O(r^(p/2))

* Solo completa en espacios finitos y sin bucles.

2.2.2 Búsqueda Heurística (Informada)

A diferencia de la búsqueda a ciegas (que explora sin rumbo), la búsqueda informada utiliza una "pista" o estimación llamada **Heurística ($h(n)$)**.

- **¿Qué es una Búsqueda Informada?** Es aquella que tiene conocimiento sobre el problema (como un mapa o una brújula) para estimar cuánto falta para llegar a la meta. Esto le permite **ordenar** las opciones y probar primero las que parecen más prometedoras, en lugar de probarlas al azar1111.

Antes de ver los algoritmos, debemos distinguir cómo toman decisiones:

Concepto Clave: Irrevocable vs. Tentativa

Tipo de Estrategia	Definición Sencilla	Ejemplo Real
Irrevocable	**"Quemar las naves"**. Tomas una decisión y avanzas sin guardar el camino de vuelta. No puedes rectificar. Si te equivocas, fallas. Ahorra mucha memoria pero es arriesgada.	Escalar una montaña en la niebla: solo subes por donde está más empinado. Si llegas a un pico falso, no puedes bajar y buscar otro pico.
Tentativa	**"Dejar migas de pan"**. Guardas información de las alternativas no exploradas. Si un camino no funciona, puedes volver atrás (rectificar) y probar otro.	Explorar un laberinto con un hilo atado a la entrada. Si hay un muro, regresas por el hilo y pruebas otro pasillo.

Búsqueda Retroactiva (Backtracking)

*Tipo: **Tentativa** (Permite rectificar)*

Es una estrategia inteligente que intenta no gastar memoria. Avanza en profundidad, pero si se equivoca, vuelve sobre sus pasos.

- **Funcionamiento**:
  - Elige el mejor hijo disponible (usando la heurística para ordenar) y avanza.
  - **Gestión de memoria estricta**: Solo recuerda el **camino actual** (la ruta activa desde el inicio), no todo el árbol.
  - **Vuelta atrás**: Si llega a un "callejón sin salida" (punto muerto), retrocede al padre y prueba el siguiente hijo prometedor 2.
- **Utilidad**: Ideal cuando tienes muy poca memoria pero necesitas encontrar una solución, aunque no sea la más corta.

Búsqueda Ávara (Greedy / Primero el Mejor)

*Tipo: **Tentativa** (En su versión general)*

Es la versión general de "guiarse por la intuición". A diferencia de la escalada pura, esta estrategia sí puede guardar una lista de alternativas pendientes (lista ABIERTA).

- **Estrategia**: Selecciona siempre el nodo que parece estar más cerca de la meta ($h(n)$ más bajo) de entre todos los disponibles en la frontera de búsqueda.
- Función de evaluación:

$$f(n) = h(n)$$

  _(Solo importa la estimación a la meta)_3.
- **Riesgo**: Aunque es más flexible que la escalada (porque guarda alternativas), es **miope**. Como ignora el coste del camino recorrido ($g(n)$), puede encontrar

soluciones, pero no garantiza que sean las óptimas (puede dar muchos rodeos innecesarios) y no es completa en espacios con bucles4.

Algoritmo de Escalada (Hill Climbing)

*Tipo: **Irrevocable** (Versión estricta del algoritmo ávaro)*

Es el ejemplo clásico de estrategia irrevocable. Busca la cima de la montaña dando siempre el paso que más sube, sin mirar atrás.

- **Estrategia**: Evalúa los vecinos y se mueve *inmediatamente* al que tiene mejor heurística (parece estar más cerca de la meta), **descartando y olvidando el resto**.
- **Función de evaluación**: $f(n) = h(n)$ (Solo importa lo que falta)5.
- **Riesgo**: Como no guarda la historia ni alternativas, puede quedarse atrapado en **máximos locales** (una pequeña colina que parece la cima pero no lo es) o **mesetas** (donde todos los pasos son iguales y no sabe a dónde ir)6666.

Algoritmo A* (A-Estrella)

*Tipo: **Tentativa** (La más completa)*

Es la estrategia estrella porque equilibra la precaución con la eficiencia. Guarda alternativas (es tentativa) y evalúa el coste total.

- **Estrategia**: No solo mira cuán cerca parece estar la meta (h), sino cuánto le ha costado llegar hasta ahí (g).
- Función de evaluación:

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

  _(Costo Total = Costo ya pagado + Costo estimado restante)_7.
- **Propiedad Clave**: Si la heurística es **admisible** (nunca es pesimista, es decir, $h(n) \leq$ coste real), A* garantiza encontrar la solución **óptima** (la más barata/corta) y es **completa** (siempre encuentra solución)8.

2.3 Fórmulas Clave y Resumen de Fórmulas

La función de evaluación cambia según el algoritmo:

Algoritmo	Fórmula de Evaluación $f(n)$	Significado
Costo Uniforme	$f(n) = g(n)$	Solo importa el costo acumulado (busca lo más barato hasta ahora). Equivalente a Amplitud si costos iguales.
Voraz (Greedy)	$f(n) = h(n)$	Solo importa la cercanía estimada a la meta (miope).

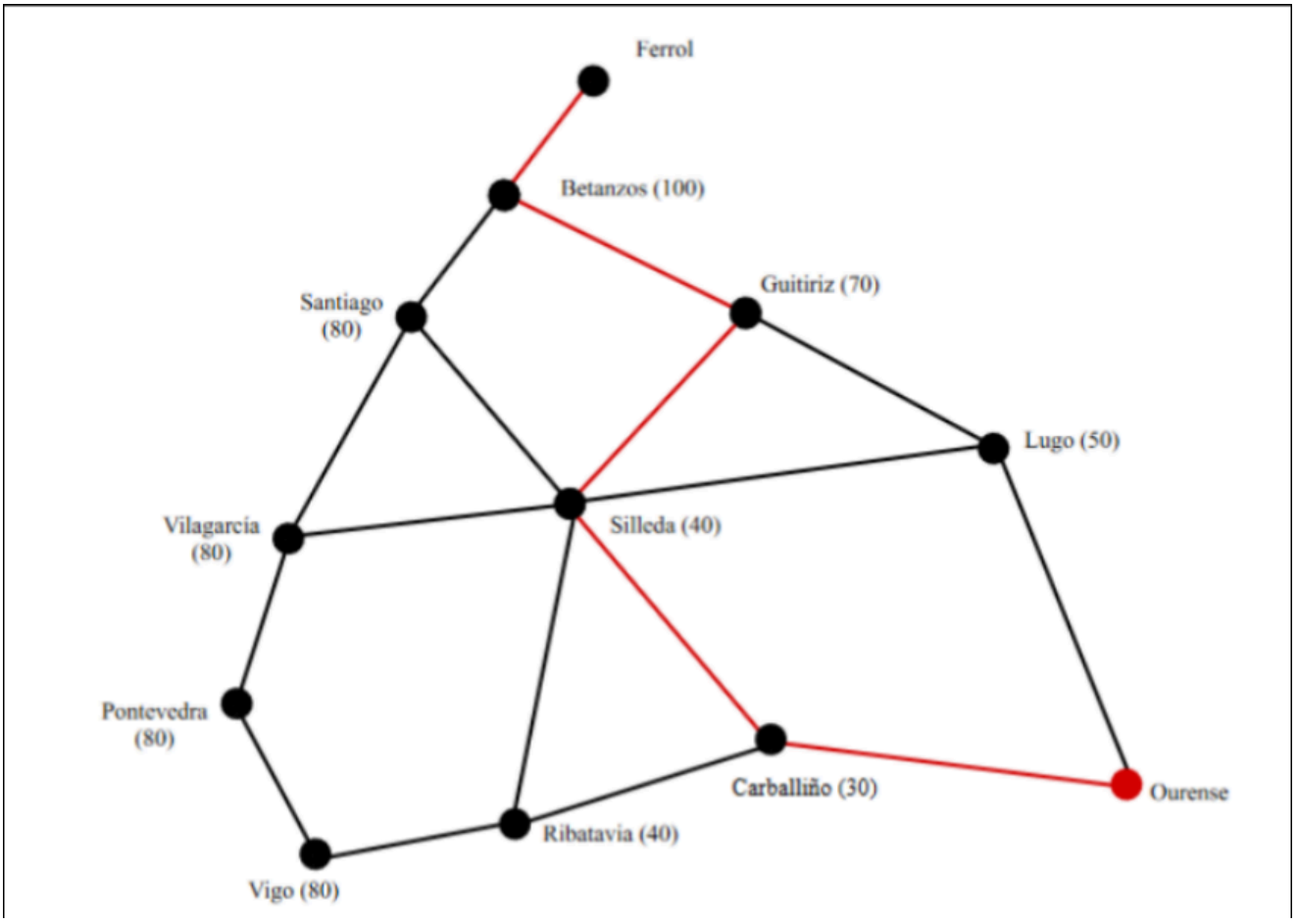
Algoritmo	Fórmula de Evaluación $f(n)$	Significado
A*	$f(n) = g(n) + h(n)$	Balance entre costo real y estimado. Busca la solución más barata globalmente.

2.4 Caso de Estudio Práctico: Ruta en Ciudades Gallegas

Este caso ilustra la diferencia entre ser "miope" (Voraz) y ser "inteligente" (A*).

El Problema: Ir de Ferrol a Ourense

- $g(n)$: Distancia real por carretera (tramos negros en el mapa)
- $h(n)$: Distancia en línea recta a Ourense (números rojos en paréntesis)



A. Ejecución Búsqueda Voraz ($f = h$)

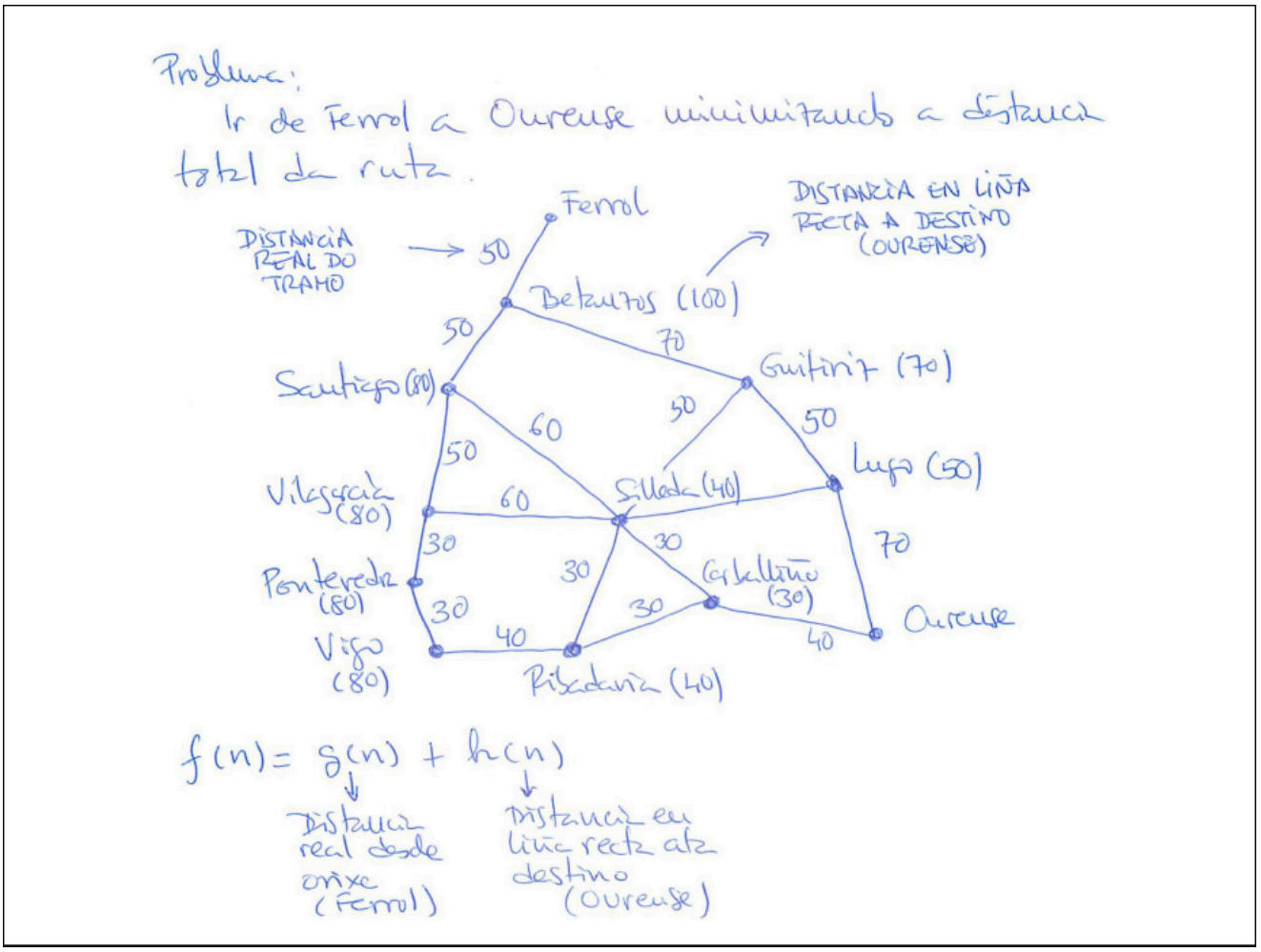
Solo mira la distancia recta a la meta.

1. Ferrol va a Betanzos
2. Betanzos tiene vecinos: Santiago ($h = 80$) y Guitiriz ($h = 70$)
3. **Decisión**: Elige Guitiriz porque $70 < 80$ (parece más cerca)
4. **Resultado**: Termina encontrando una ruta de 240 km. **No es la mejor**.

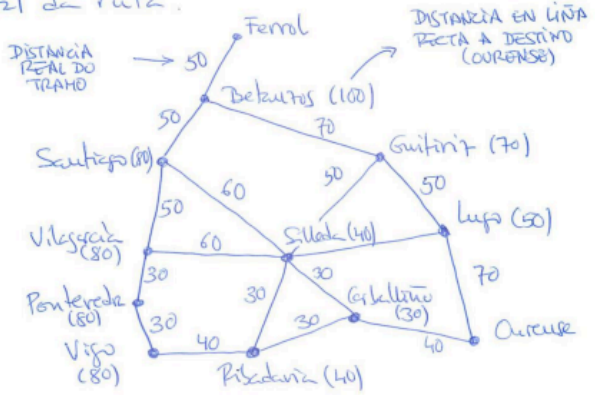
B. Ejecución Búsqueda A* ($f = g + h$)

Mira el pasado (g) y el futuro (h).

1. Ferrol va a Betanzos ($g = 50$)
2. En Betanzos, evalúa opciones:
   - Ruta por Guitiriz: $g(50 + 70) + h(70) = 120 + 70 = 190$
   - Ruta por Santiago: $g(50 + 50) + h(80) = 100 + 80 = 180$
3. **Decisión**: Elige Santiago porque $180 < 190$. Aunque Santiago parece más lejos en línea recta, el costo total estimado es menor.
4. **Resultado**: Encuentra la ruta óptima de 230 km pasando por Santiago y Silleda.



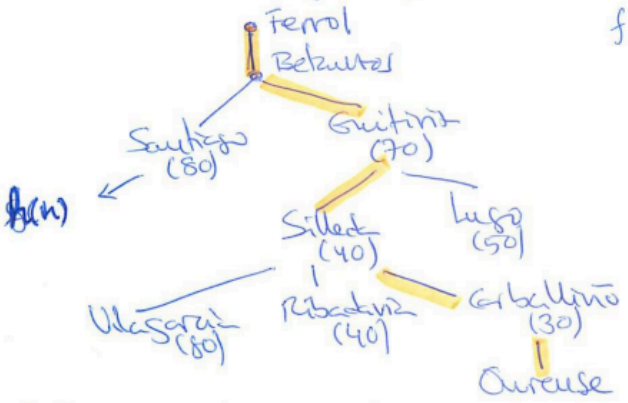
Problema:
Ir de Ferrol a Ourense minimizando a distancia total da ruta.



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

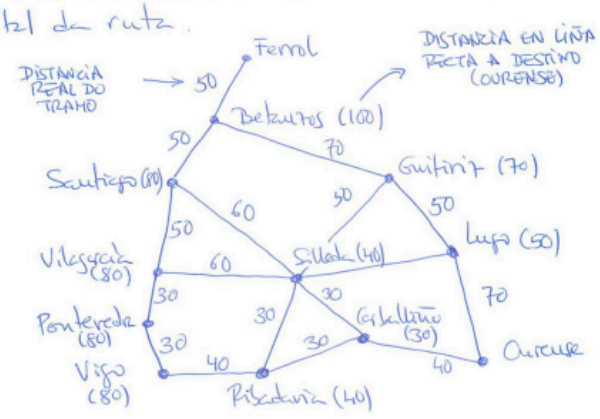
- $g(n)$: Distancia real desde orixe (Ferrol)
- $h(n)$: Distancia en liña recta ata destino (Ourense)

Busca avara $g(n) = \emptyset$



Distancia real: $50 + 70 + 50 + 30 + 40 = 240$

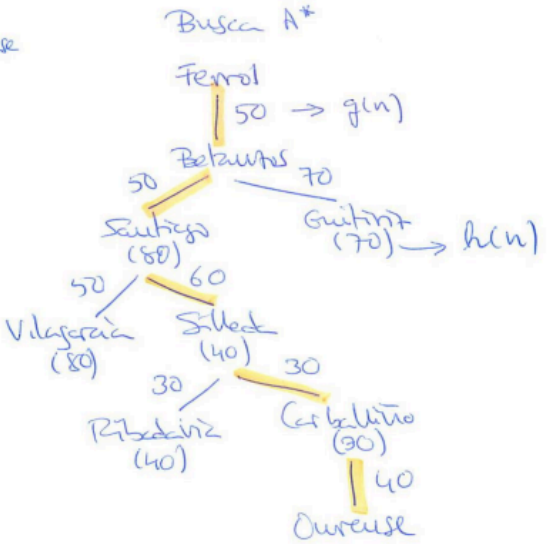
Problema:
Ir de Ferrol a Ourense minimizando a distancia total da ruta.



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

- $g(n)$: Distancia real desde orixe (Ferrol)
- $h(n)$: Distancia en liña recta ata destino (Ourense)

Busca A*



Distancia real: $50 + 50 + 60 + 30 + 40 = 230$