

## 2. Búsqueda en Espacio de Estados

Copyright (c) 2025 Adrián Quiroga Linares Lectura y referencia permitidas;  
reutilización y plagio prohibidos

### 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)

#### Info

**Tipos de grafos generados en los procesos de búsqueda:**

- **Explícito:** Es un grafo que **ya está construido y guardado en la memoria** del ordenador antes de empezar a buscar nada.
- **Implícito:** Es un grafo que **no existe en memoria**, sino que se va **generando a medida que avanzas**. Se define mediante reglas.

### 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.

### Características de una 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*	Si cuando $l > p$	$O(r * l)$	$O(r * l)$
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 azar.

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

## Concepto Clave: Irrevocable vs. Tentativa

Tipo de Estrategia	Definición Sencilla	Ejemplo Real
<b>Irrevocable</b>	<b>"Quemar las naves"</b> . 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.
<b>Tentativa</b>	<b>"Dejar migas de pan"</b> . 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.

### Nota

Si  $f(n) = g(n)$  es una búsqueda no informada

## 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).
- **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).

## 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).

- **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).

### Info

Una heurística admisible u optimista: siempre suponer casos en los que la realidad será peor que la estimación.

**Si eres Pesimista ( $h >$  coste real):** Imagina que hay un atajo secreto que tarda 15 minutos. Pero tu heurística es pesimista y le dice al algoritmo: "Por ese callejón vas a tardar 1 hora".

- **Consecuencia:** El algoritmo se asusta, ve un coste altísimo y **no explora ese camino**.
- **Resultado:** Pierdes la oportunidad de encontrar el mejor camino (la solución óptima) porque tu estimación te engañó diciendo que era malo.

**Si eres Optimista ( $h \leq$  coste real):** Tu heurística le dice al algoritmo: "Por ese callejón parece que tardas solo 5 minutos".

- **Consecuencia:** Como parece un camino barato, el algoritmo **lo explora**.

- **Resultado:** Al entrar, se da cuenta de que en realidad son 15 minutos (la realidad es peor que la estimación, como dice tu nota), pero **al menos lo exploró**.
- A\* prefiere equivocarse pensando que un camino es "demasiado bueno" (porque así lo comprueba) que equivocarse pensando que es "demasiado malo" (porque entonces lo ignora y podría ser la solución).

En el caso del 8-puzzle se acostumbra a usar estas **heurísticas**:

**Misplaced Tiles (Piezas Descolocadas):** Es la más simple. Simplemente **cuentas cuántas fichas no están en su posición final**.

- **Cálculo:** Si la ficha '1' está en la casilla del '2', suma 1. Si la '2' está bien colocada, suma 0.
- **Ejemplo:** Si 5 fichas están mal puestas,  $h(n) = 5$

**Distancia de Hamming:** En el contexto del 8-puzzle, es **sinónimo de "Misplaced Tiles"**.

- Proviene de la teoría de la información (número de posiciones en las que dos cadenas de símbolos difieren). Aquí compara el "estado actual" con el "estado meta" y cuenta las diferencias.

**Distancia de Manhattan (City Block):** Es más precisa (y más informada) que la anterior.

- **Cálculo:** Para cada ficha, calculas cuántas casillas debe moverse (horizontal + vertical) para llegar a su destino. Luego **sumas todas esas distancias**.
- **Por qué se llama así:** Porque simula moverse por las calles de Manhattan (en cuadrícula), no puedes ir en diagonal.
- **Ejemplo:** Si la ficha '1' está a 2 casillas a la derecha y 1 abajo de su meta, su distancia es 3.

**Heurística de Gaschnig:** es una **relajación** del problema. Asume que puedes mover cualquier ficha a la posición del hueco (teletransportándola), no solo las adyacentes.

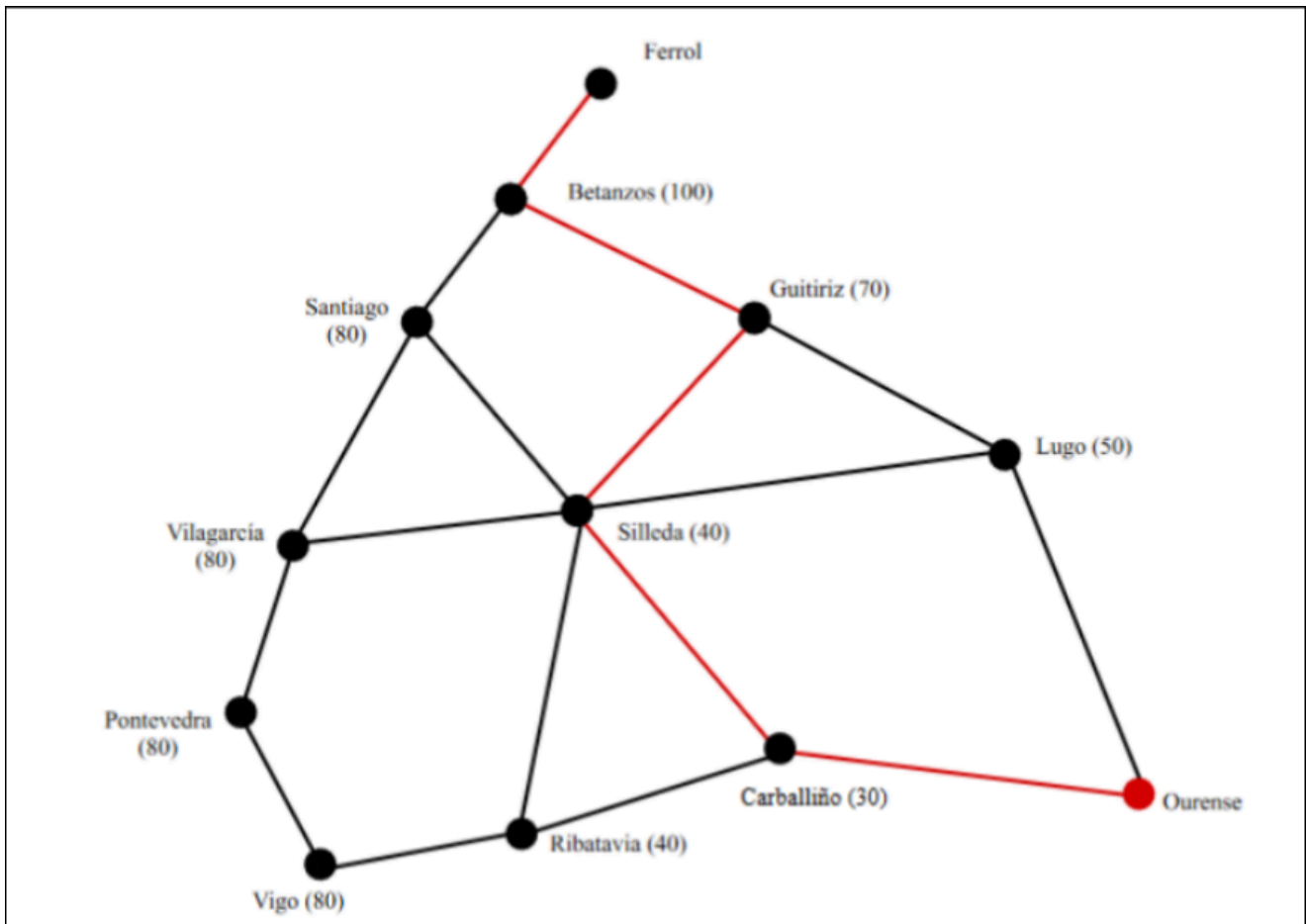
- **Cálculo:** Cuenta el número de intercambios necesarios para resolver el puzzle si pudieras intercambiar el hueco con **cualquier** ficha del tablero para ponerla en su sitio de un solo movimiento.
- Suele dar un valor entre la de Hamming y la real.

## 2.3 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\* ( $f = g + h$ )

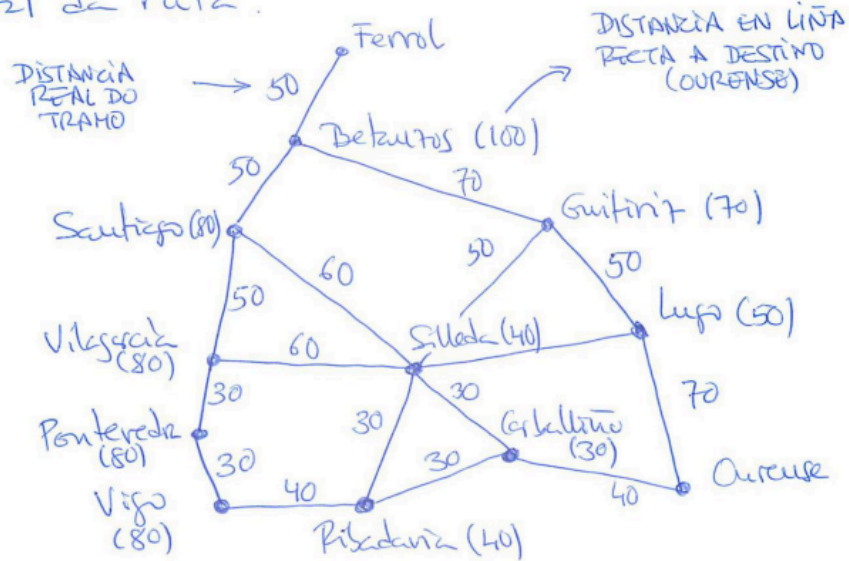
Mira el pasado ( $g$ ) y el futuro ( $h$ ).

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 la distancia total de la ruta.



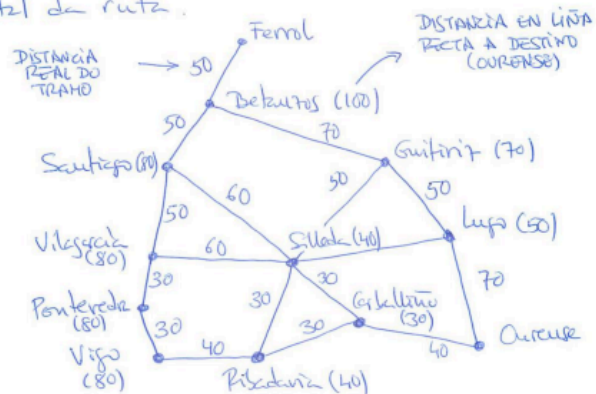
$$f(n) = g(n) + h(n)$$

↓  
Distancia real desde origen (Ferrol)

↓  
Distancia en línea recta al destino (Ourense)

Problema:

Ir de Ferrol a Ourense minimizando la distancia total de la ruta.

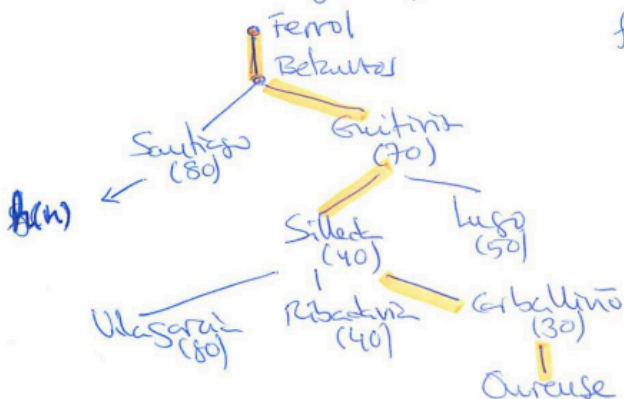


$$f(n) = g(n) + h(n)$$

↓  
Distancia real desde origen (Ferrol)

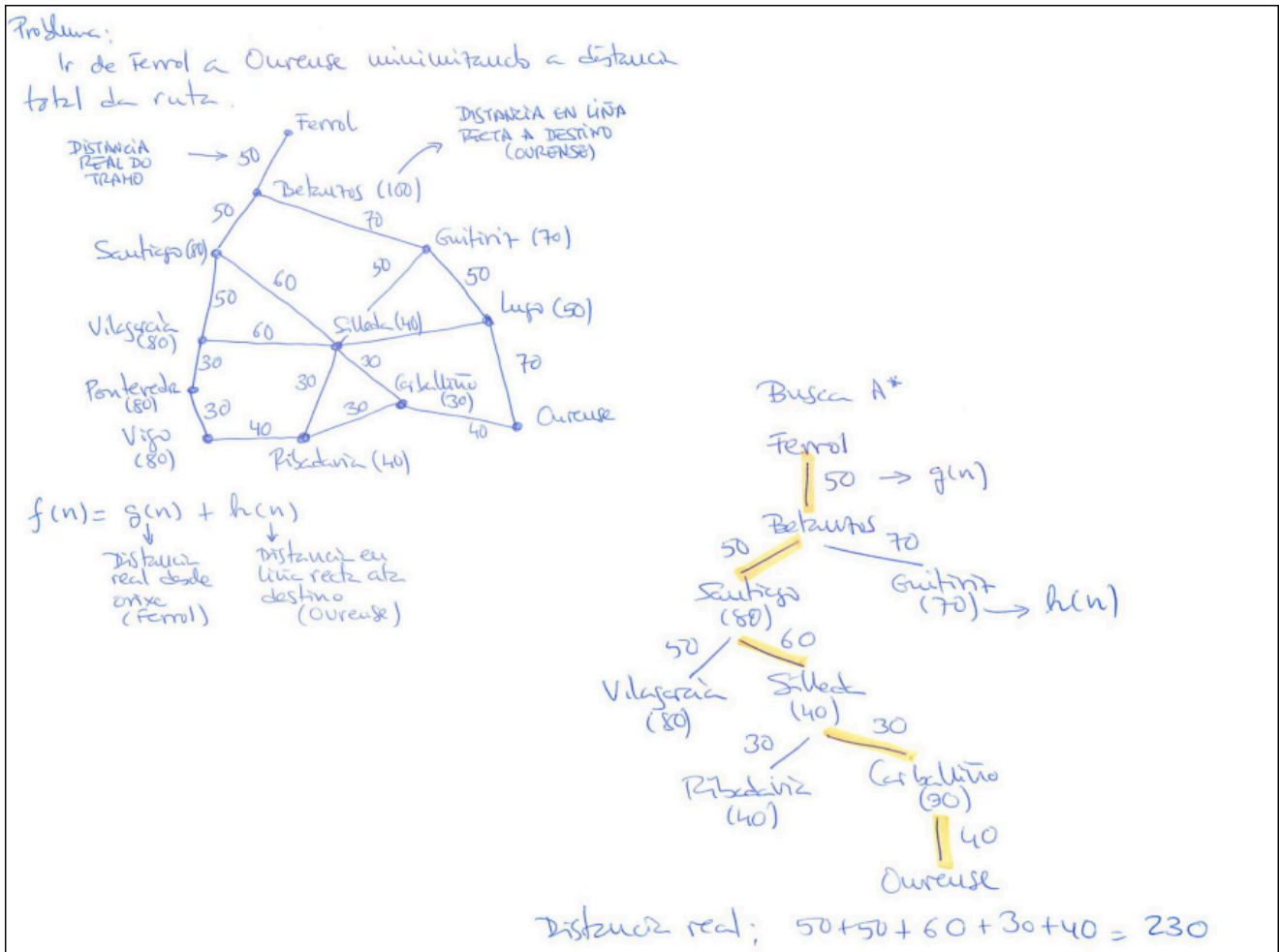
↓  
Distancia en línea recta al destino (Ourense)

Busca avanza  $g(n) = 0$



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





## 2.4 Destellos de calidad del Senén en clase

### 2.4.1 ¿Qué es el Problema del Viajante de Comercio (TSP)?

Imagina que eres un repartidor de Amazon.

- Tienes una lista de **100 ciudades** que visitar.
- Sales de la central (digamos, la ciudad 0).
- **Reglas:** Tienes que visitar todas las ciudades **una sola vez** y volver al punto de partida.
- **Objetivo:** Encontrar la ruta más corta posible para ahorrar gasolina.

Esto es el **TSP (Traveling Salesperson Problem)**. En informática es famoso porque es muy difícil de resolver perfectamente cuando hay muchas ciudades, ya que el número de combinaciones posibles es gigantesco.

### 2.4.2 ¿Qué es el "Tamaño del Espacio de Búsqueda"?

Es una forma técnica de preguntar: "**¿Cuántas rutas distintas posibles existen en total en el universo?**". Para calcularlo en este problema (100 ciudades, empezando en la 0), usamos matemáticas de combinatoria (permutaciones):

1. La **Ciudad 0** está fija al principio y al final. No la puedes mover.

- Te quedan **99** ciudades libres para barajar.
- El número de formas de ordenar esas 99 ciudades es:  
 $99 \times 98 \times 97 \times \dots \times 1 = 99!$

Este número es inmensamente grande (mayor que el número de átomos en el universo). Por eso es imposible probarlas todas una por una.

### 2.4.3 ¿Qué es el Entorno (Contorna) y su Tamaño?

Aquí es donde entra la confusión habitual. Si el "Espacio de Búsqueda" es toda la biblioteca, el **Entorno** son solo **los libros que puedes alcanzar con la mano sin moverte**. Los algoritmos de búsqueda local (como Escalada o Tabú) no ven todo el espacio a la vez. Funcionan paso a paso modificando la ruta actual.

- Definición:** El entorno es el conjunto de **todas las rutas "vecinas"** que puedes generar aplicando **un solo operador** a tu ruta actual.
- El Operador:** Normalmente usamos el "intercambio de dos ciudades" (swap).

**Cálculo del Tamaño del Entorno (N):** La pregunta es: *"¿Cuántas parejas de ciudades puedo elegir para intercambiar?"*.

- Tenemos **99 ciudades** móviles (recuerda: la ciudad 0 es fija, esa no se toca).
- El operador elige **2** de ellas para cambiarlas de sitio.
- Matemáticamente es una **combinación sin repetición**:  

$$N = 299 \times 98$$

#### Diferencia Clave:

- Espacio de Búsqueda (99!):** Todas las soluciones que existen.
- Entorno (≈4851):** Las opciones que tiene el algoritmo *en cada paso* para decidir hacia dónde moverse.

### 2.4.4 ¿Qué es la Búsqueda Tabú?

Es una **estrategia** inteligente para moverse por ese entorno sin atascarse.

- El algoritmo genera el entorno (las 4851 opciones vecinas).
- Elige la mejor opción para moverse.
- El problema:** Podría quedarse atrapado haciendo y deshaciendo el mismo cambio (ej: cambiar Madrid-Barna y luego Barna-Madrid).
- La solución (Tabú):** Tiene una "memoria" de los últimos cambios.
- La regla:** "Si acabo de probar este cambio, está **prohibido (tabú)** volver a hacerlo o deshacerlo durante un número X de turnos".

**Nota para el examen:** Usar Búsqueda Tabú **NO cambia el tamaño del espacio de búsqueda** (99! sigue siendo 99!). Lo que hace es guiar *qué partes* de ese espacio

exploramos para no perder tiempo en círculos, por lo que el contorno inicial es más grande que el resto.