Apuntes de Sistemas Inteligentes

Miguel Mejía Jiménez

13 de marzo de 2018

ETSI Informática, UMA

Página intencionalmente en blanco

Índice

Por favor, no imprimas estos apuntes si no es estrictamente necesario.
Este documento es gratuito; si has pagado por él, has sido engañado.
Así como no deberías pagar por él, tampoco deberías cobrar por distribuirlo.

1. Introducción

La inteligencia es aquello que nos permite seleccionar las acciones adecuadas. La IA es una ciencia que utiliza la ingeniería como método, puesto que ni siquiera comprendemos la inteligencia natural.

1.1. Historia

1950 - Alan Turing: Computer machinery and intelligence

El pensamiento no es observable, pero el comportamiento sí. Propone que se pueden construir máquinas cuyo comportamiento es indistinguible del humano. Sugiere el juego de la imitación, que conocemos hoy como Test de Turing. Años después, en 1990 nace el premio Loebner.

1956 - John McCarthey: Dortmouth Summer Conference on AI

Aparece por primera vez el término de *Artificial Inteligence*. Reúne a un conjunto de investigadores interesados en la conjetura de que "todos los aspectos del aprendizaje o cualquier otra característica de la inteligencia pueden describirse en principio con tanta precisión como para hacer que una máquina los simule": Herbert Simon (Nobel en economía, investigador de psicología cognitiva, fundador del ACM), Allen Newell (fundador del ACM), Claude Shanen (teoría de la información), Marvin Minsky ...

1976 - Newell y Simon: Computer Science as empirical inquiry: symbols and search

Presentan las dos hipótesis de la Inteligencia Artificial Simbólica:

1. *Hipótesis del sistema de símbolos físico* (SSF): Un SSF posee los medios suficientes y necesarios para acciones inteligentes generales
2. *Hipótesis de la búsqueda heurística*: Las soluciones a los problemas pueden representarse mediante estructuras simbólicas. Un SSF ejerce su inteligencia en la resolución de problemas mediante una búsqueda; es decir, generando y modificando progresivamente estructuras simbólicas hasta producir una estructura solución.

Definición. Sistema de Símbolos Físico: Un SSF posee un conjunto de símbolos, está relacionado con un contexto real y es capaz de crear, copiar, modificar y destruir expresiones simbólicas.

Se describen a dos niveles:
- Físico
- Simbólico
  - Expresiones simbólicas
  - Procesos que las manipulan

Las expresiones y los procesos son lo que estudia la *Ingeniería del conocimiento*.

Otras hipótesis

- 1943 - McCullough y Pitts: Hipótesis conexionista (Redes neuronales)
- 1950 - Rodney Brooks: Intelligence without representation (robótica evolutiva)

1.2. IA débil e IA fuerte

No tiene sentido hablar de *pensar* sin hablar de *conciencia*

Definición. IA débil o estrecha: Una inteligencia artificial capaz de realiar tareas específicas. Se puede simular el comportamiento humano, que es observable, a diferencia de la conciencia. (Turing)

Definición. IA fuerte o general: Una inteligencia artificial capaz de realizar todas las actividades intelectuales de las que el ser humano es capaz. La mente humana es un sistema informático y por tanto puede simularse. (Hofstadter y Kurzweil)

Searle propone el *Experimento de la habitación china*. La defensa de la IA fuerte llega a contradecir la hipótesis de SSF.

1.3. Agente y Entorno

Agente

El concepto de *agente* permite organizar el estudio y desarrollo de sistemas inteligentes. Está definido por las siguientes características:

- Se desenvuelve en un *entorno*.
- Puede percibir el *entorno*.
- Puede actuar en el *entorno*.
- Tiene un objetivo.
- Es racional (racionalidad limitada).
- Posee cierto conocimiento (según la HSSF).

Entorno

Se puede ver, pues, que la arquitectura de un agente depende en gran medida del entorno. Este puede clasificarse de varias maneras:

- *Discreto* o *continuo*.
- *Total* o *parcialmente* observable.
- *Determinista* o *estocástico*.
- *Uno* o *múltiples* agentes.

Definición. Plan: Solución a un problema en un entorno discreto, totalmente observable, conocido, determinista y de un agente.

Ejemplo
Un problema de pathfinding se resuelve con un plan.

Definición. Juego: Problema situado en un entorno discreto, totalmente observable, conocido, determinista y con un adversario.

Ejemplo
El ajedrez es un juego

2. Búsqueda

Según la hipótesis heurística[1] los problemas pueden solucionarse mediante la búsqueda de una estructura simbólica solución.

2.1. Formalismo de representación. Espacio de Estados

Un Espacio de Estados (EE) está formado por dos componentes:

$$\text{EE} \begin{cases} \text{Conjunto de estados} \\ \text{Conjunto de reglas} \end{cases}$$

Definición. Conjunto de estados (E): Todas las situaciones posibles del mundo del entorno.

Definición. Conjunto de reglas: Determina como puede cambiar el mundo. Suelen tener una *precondición* $E \to (true, \text{false})$ y una *postcondición* $E \to E$

Hay casos en los que las acciones tienen costes. Suelen ser costes aditivos;es decir, se van sumando con cada acción.

Las acciones y los estados pueden definirse de muchas maneras en cada EE. Nos interesa definirlos de la manera completa más sencilla posible. Debemos tener en cuenta que el objetivo final de darle representación al problema es implementarlo en un ordenador.

Definición. Instancia de problema: Un problema se instancia con un *estado inicial* y un *estado* o *condición objetivo*.

Definición. Solución: Consideramos dos tipos de soluciones:

1. Si conocemos el estado objetivo
   Secuencia válida de acciones que permiten pasar del estado inicial al objetivo. Por ejemplo, un problema de pathfinding.

2. Si conocemos la condición objetivo
   La solución es el propio el estado objetivo. Por ejemplo, en un sudoku.

Ejemplo
El *puzzle 8* es problema en un entorno discreto, totalmente observable, conocido, determinista y de un solo agente. Existe una matriz de 3×3 en la que hay 8 números y 1 hueco, organizados aleatoriamente. Los números adyacentes al hueco se van intercambiando con éste para llegar a una configuración dada.

	6	5
8	1	4
7	3	2

$\to$

1	2	3
8		4
7	6	5

Vamos a definir el espacio de estados.

- *Conjunto de estados*. En este caso, es el conjunto de todas las posibles permutaciones de la matriz. El tamaño de este conjunto de estados sería de $9! = 362880$ casos.

[1] Heurística proviene de *eureka*, que significa "lo encontré"; el proceso de búsqueda no es ciego; se realiza la prueba y error aprovechando las regularidades del entorno para guiarse.

- *Conjunto de reglas.*

  Dos enfoques
  - Puedo mover cada ficha en cuatro direcciones: 32 reglas
  - Puedo mover el hueco en cuatro direcciones: 4 reglas

  Nos interesa más el segundo enfoque. El primero requeriría más especificaciones para cada regla, que además son muchas más. El segundo, en cambio, es mucho más simple y expresa exactamente las mismas conexiones entre estados.
  Vamos a definir estas cuatro reglas.

	Precondición	Efecto
mover arriba	$F > 0$	permutar (F,C) con (F-1,C)
mover abajo	$F < 2$	permutar (F,C) con (F+1,C)
mover derecha	$C < 2$	permutar (F,C) con (F,C+1)
mover izquierda	$F > 0$	permutar (F,C) con (F,C-1)

Así queda definido el espacio de estados del problema del puzzle 8. Como el propósito es plantear el EE y no resolverlo, dejaremos el ejemplo aquí.

2.2. Búsqueda en grafos abstractos

Definición. Algoritmo de búsqueda completo: Un algoritmo de búsqueda es completo si, habiendo solución, la encuentra.

Definición. Algoritmo de búsqueda admisible: Un algoritmo de búsqueda es admisible si, habiendo solución, encuentra la óptima.

Un EE define implícitamente un grafo. Los nodos representan los estados y los arcos las transiciones entre ellos. Resolver un problema es pasar de un estado inicial a otro final; por tanto, resolver un problema equivale a encontrar un camino en un grafo.

Hay tres tipos de algoritmos o representaciones:

1. *Búsqueda en grafos abstractos* (representación atómica)
   Es la que estudiaremos en este apartado. Entran en esta clasificación los algoritmos que no acceden a la representación interna de nodos y arcos. Solo usan operaciones básicas (comprobar si un nodo es final, ver nodos sucesores...).

2. *Representaciones vectoriales* (Representación factorizada)
   Los estados se representan como vectores de variables.

3. *Representación basada en lenguajes lógicos*

Aquí estudiaremos la *Búsqueda en grafos abstractos*. Hay dos maneras de clasificarlas:

- -Estrategia de búsquedas (Clasificación de Nilson)
  - *Irrevocables* Estudian un solo camino
  - *Tentativas* Exploran sistematicamente los caminos que parten del estado inicial
    - *Con retroceso* *Depth first* Sólo recuerdan el camino actual
    - *Con árbol* *Best first*Recuerdan todos los caminos que nos interesan
- -Estrategia de búsquedas Otras clasificaciones
  - *A ciegas*
  - *Heurística*

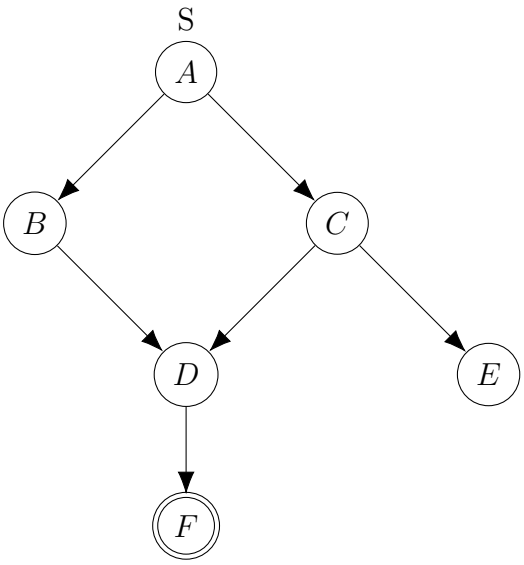
2.2.1. Búsqueda con árbol

Las búsquedas con árbol usan dos estructuras de datos:

1. *Árbol de búsqueda* que tiene la raíz en el nodo inicial S[1].
2. *Lista de nodos abiertos*. Son los nodos por los que se puede seguir buscando.

   La operación básica en la búsqueda es la *expansión de nodos*. Expandir un nodo consiste en calcular sus sucesores y añadirlos al árbol. Si procede (no se permiten ciclos en el árbol), se añaden a la lista de abiertos.

Ejemplo

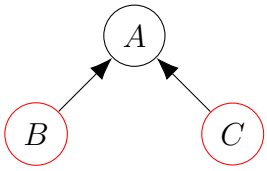


Tenemos un grafo como el del dibujo, con un estado inicial A y un estado final F.
A continuación se muestra la evolución de las estructuras de datos de la búsqueda de árbol. Tanto el *árbol de búsqueda* como la *lista de abiertos* empieza almacenando solo el nodo raíz. De ahí iremos abriendo nodos hasta llegar al final.
Es importante notar que con cada expansión, los nuevos nodos que se añaden al árbol apuntan a su padre y no al revés, como es habitual. Esto se hace para facilitar la distinción del camino cuando acabe la búsqueda.
Para ahorrar espacio sólo se muestra el *árbol de búsqueda*. Los nodos abiertos están señalados en rojo.

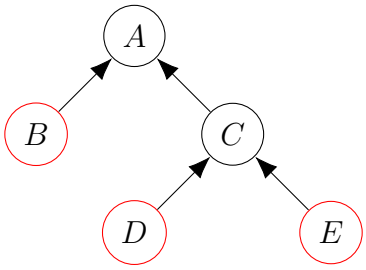
[1] De *Start* en inglés o *Salida* en español.

1. Empezamos con el nodo inicial.

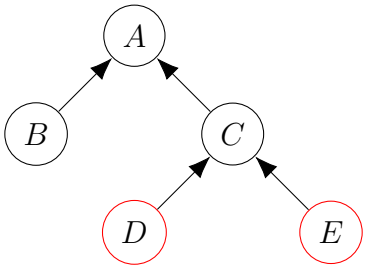
2. Expandimos A.



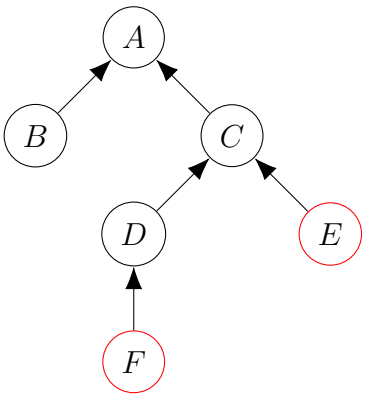
3. Expandimos C.



4. Expandimos B. Como el único sucesor de B ya está en el árbol, no lo añadimos. Hacerlo provocaría un ciclo y dejaríamos de tener un árbol.



5. Expandimos D.

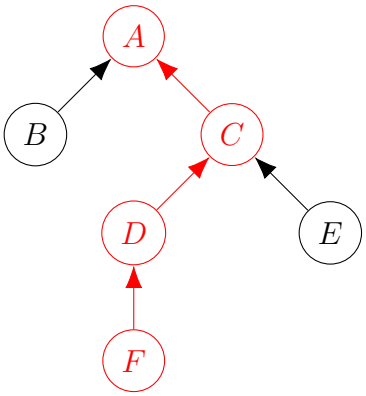


6. Llegados a este punto existen dos procedimientos posibles. El nodo final se considera encontrado si:

   a) Lo añadimos a la lista de abiertos.

   b) Lo seleccionamos para la expansión.

   En este caso, simplificaremos escogiendo la primera opción.
   Ahora podemos *trackear* el camino solución siguiendo los punteros hasta llegar al nodo inicial.



Los distintos algoritmos de búsqueda con árbol difieren principalmente en dos aspectos:

- Ordenación de la lista de abiertos.
- Política de elección de caminos a un mismo nodo.

2.2.2. Búsqueda en amplitud

Este algoritmo de la búsqueda en árboles analiza cada nodo del nivel de profundidad actual antes de pasar al siguiente.

- La lista de abiertos se comporta como una cola (política *FIFO*).
- Política de elección de caminos arbitraria.

Las características de este algoritmo son las siguientes:

- *Completo.*
- *Admisible.*
- *Consumo de tiempo* exponencial. Las iteraciones que realiza aumentan de forma $O(b^d)$, siendo b el factor de ramificación (cuántas ramas parten de cada nodo) y d la profundidad a la que está la solución.
- *Consumo de memoria* exponencial. La propia distribución de nodos de un árbol explica esto.

2.2.3. Algoritmo de Dijkstra

Es un algoritmo de optimización. Encuentra el camino menos costoso entre dos nodos de un grafo ponderado. Cada nodo tiene un coste positivo asociado y el coste de un camino es la suma de los arcos que lo componen.
Para cada nodo n se define la función $g(n)$, que es el coste del camino guardado en el árbol desde S hasta n.

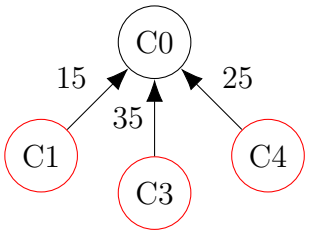
- Los nodos abiertos se ordenan según su $g(n)$ de forma creciente.
- Entre dos caminos a un nodo m, escogemos el que tiene menor $g(m)$

El algoritmo de Dijkstra es *admisible*
Ejemplo

	C0	C1	C2	C3	C4
C0		15		35	25
C1	15		15	28	
C2		15			4
C3	35	28			
C4	25		4		

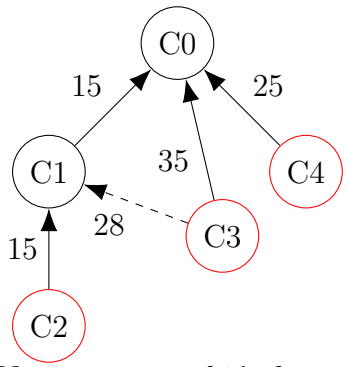
Queremos calcular la distancia menor entre las ciudades C0 y C3 . Los costes de viajar entre cada ciudad están en la tabla. Llevaremos el progreso del algoritmo con el árbol de búsqueda, y un a tabla donde analiamos los nodos que abrimos. El árbol lo construiremos igual que en el último ejercicio de búsqueda con árbol.

1. Empezamos con el nodo inicial y lo abrimos.



nodo sel	sucesores	$g(n)$
C0	C1	15
	C3	35
	C4	25

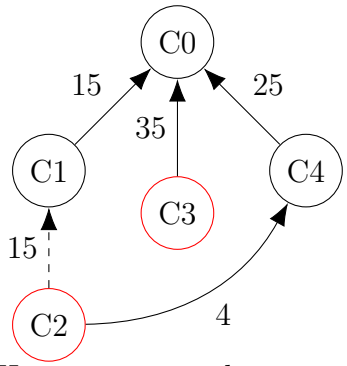
2. Expandimos el nodo abierto con menor $g(n)$: C1.



nodo sel	sucesores	$g(n)$	
C0	C1	15	Cerrado 1º
	C3	35	
	C4	25	
C1	C2	$15 + 15 = 30$	
	C3	$15 + 28 = 43$	Descartado

Vemos que también hay un camino que lleva a C3 pasando por C1. Según la política de elección de caminos, nos quedamos con el de menor $g(n)$, que en este caso es el antiguo. Descartamos el nuevo.

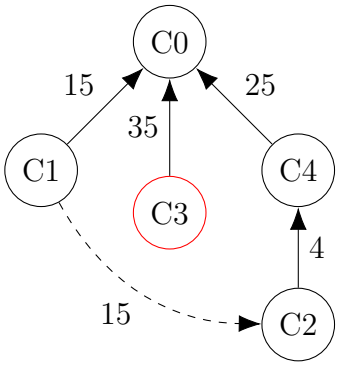
3. El siguiente nodo abierto con menor $g(n)$ es C4.



nodo sel	sucesores	$g(n)$	
C0	C1	15	Cerrado 1º
	C3	35	
	C4	25	Cerrado 2º
C1	C2	30	Descartado
	C3	43	Descartado
C4	C2	$25 + 4 = 29$	

Hemos encontrado un camino hacia C2 que pasa por C4 y que es mejor que el anterior. Descartamos el antiguo.

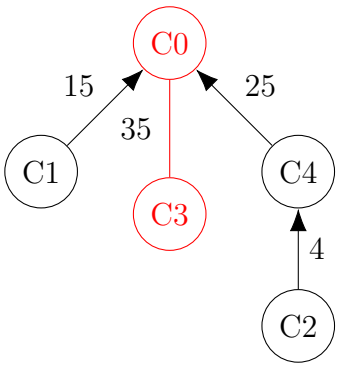
4. El siguiente nodo para ser abierto es C2.



nodo sel	sucesores	$g(n)$	
C0	C1	15	Cerrado 1º
	C3	35	
	C4	25	Cerrado 2º
C1	C2	30	Descartado
	C3	43	Descartado
C4	C2	29	Cerrado 3º
C2	C1	$29 + 15 = 44$	Descartado

El único sucesor de C2 que no provoca un ciclo es C1; en realidad, el camino alternativo que se nos presenta es el mismo que borramos en el paso anterior. Al principio lo habíamos considerado desde C1, pero ahora, como lo habíamos borrado, vuelve a aparecer desde C2.

5. Por último, nos queda el nodo C3.



nodo sel	sucesores	$g(n)$	
C0	C1	15	Cerrado 1º
	C3	35	Cerrado 3º
	C4	25	Cerrado 2º
C1	C2	30	Descartado
	C3	43	Descartado
C4	C2	29	Cerrado 3º
C2	C1	44	Descartado
C3	-	-	SOLUCIÓN

En el algoritmo de Dijkstra, consideramos que hemos encontrado la solución cuando seleccionamos el nodo objetivo para su expansión. Incluso si ya hemos encontrado un camino antes, no podemos asegurar que el camino hallado es óptimo hasta que lo seleccionamos para expandir.

Puesto que el nodo objetivo era C3 y acabamos de seleccionarlo para expansión, hemos encontrado el camino solución.

2.2.4. Algoritmo A*

A diferencia de Dijkstra y busqueda de amplitud, que son algoritmos de búsqueda a ciegas, A*es un algoritmo de *búsqueda heurística*.

Para la búsqueda heurística nos hace falta un *heurístico*, que notaremos $h(n)$.

Definición. Heurístico: Es una estimación rápida del coste de ir desde el nodo n hasta el nodo objetivo. Debe tener poco coste computacional, porque, a fin al cabo, el objetivo de la búsqueda heurística es llegar más rápido al resultado final.

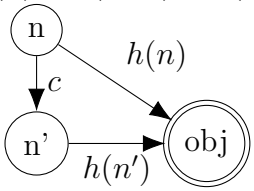
Definición. Heurísitco más informado: Dados dos heurísticos admisibles para un mismo problema h_1 y h_2 decimos que h_2 está más informado que h_1 si

$$h_2(n) > h_1(n)\ \forall n \notin O$$

donde consideramos O como el conjunto de nodos objetivo.

Definición. Heurístico monótono: Un heurístico h es monotono (o consistente) si para todo arco $(n.n')$ se cumple la desigualdadd siguiente:

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



y cuando esto se cumple, la secuencia de valores $f(n)$ a lo largo del camino es monótona no decreciente.

$$f(S) \leqslant f(n_1) \leqslant f(n_2)...$$

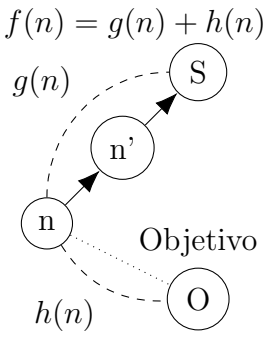
Definición. Distancia euclídea: Distancia "ordinaria" entre dos puntos. Tal y como se mediría con una regla o el teorema de pitágoras.

Definición. Distancia Manhattan: Suma de las distancias para cada coordenada por separado.

Para encontrar un heurístico para un problema, se usa la *técnica de relajación*, que consiste en eliminar restricciones del entorno con el fin estimar un coste apróximado para el camino.

Para un mismo problema puede haber distintos heurísticos. Conviene encontrar uno que, sin tener mucho coste computacional, tenga una precisión satisfactoria. Hay casos en los que mayor precisión conlleva mayor coste, y casos en los que no.

- En el algoritmo A*los nodos abiertos se ordenan según la función $f(n)$, definida como:



  donde $g(n)$ se define igual que en el algoritmo de Dijkstra y $h(n)$, como hemos visto, es el heurístico. El componente heurístico de este algoritmo reside entonces en tener en cuenta una estimación $h(n)$ del coste para llegar al objetivo como guía, además del coste del camino recorrido.

- Entre dos caminos que llevan a un mismo nodo, elegiremos el que presente menor $f(n)$. Cuando se encuentra un nuevo camino para un nodo que ya estaba cerrado, hay que volver a abrirlo. Si esto sucede, los sucesores de este nodo reabierto tendrán un $f(n)$ menor, así que habrá que estudiarlos de nuevo.

Propiedades de A*

1. A*es *admisible incluso en grafos infinitos* si se cumple que:

   a) El grafo explorado es localmente finito

   b) Los costes están acotados positivamente

   $$c(n, n') \geqslant \varepsilon > 0, \forall (n, n')$$

   c) La función heurística es optimista (admisible)

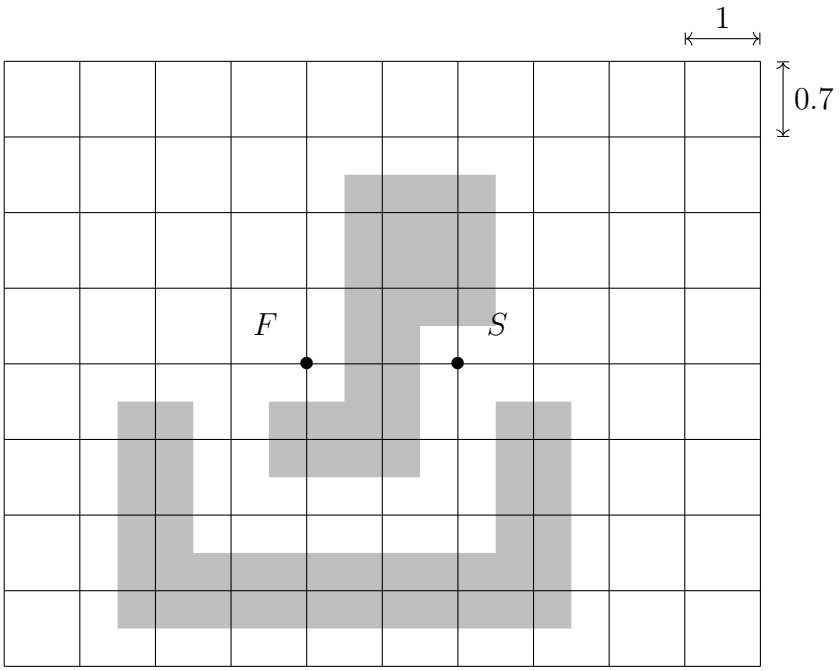
   $$h(n) \leqslant h^*(n)$$

   donde $h^*(n)$ es el coste óptimo real de un camino desde n hasta el objetivo.Sabemos esto si es monótono y nulo para todo nodo objetivo.

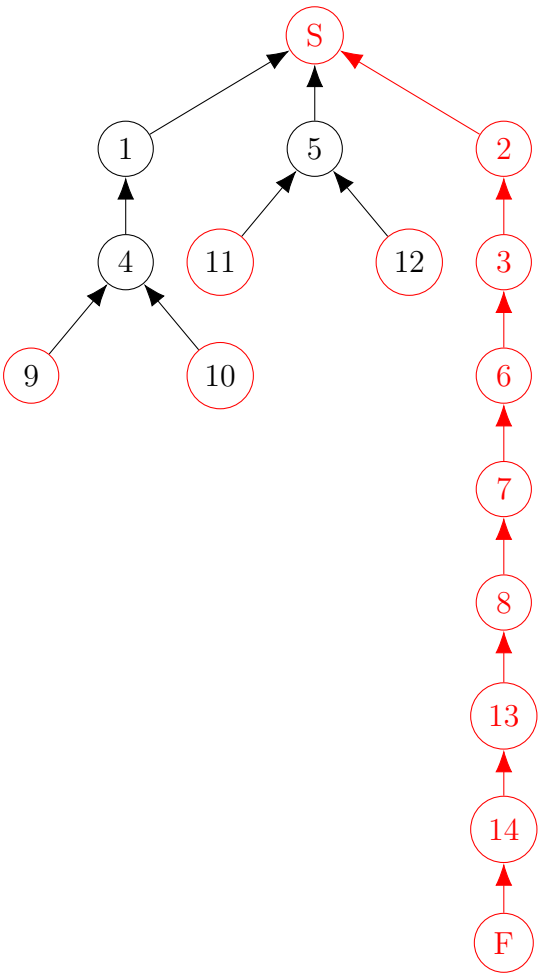
2. Sean dos algoritmos A*: A_1 con un heurístico h_1 y A_2 con h_2. Si h_2 está más informado que h_1, entonces A_1 expande *al menos* los mismos nodos que A_2.
   En otras palabras, un algoritmo con un heurístico mejor informado expande tantos nodos o menos que uno menos informado.

3. Si $h(n)$ es monótono, cuando A*selecciona un nodo, *ya ha encontrado un camino óptimo hasta él*. Esta propiedad nos asegura que A*abra cada nodo una y solo una vez.

Ejemplo
Un problema de pathfinding en una malla *no cuadrada* con obstáculos (configuraciones imposibles). En el ejemplo del algoritmo de Dijkstra mostré paso a paso como se expanden los nodos. Esta vez voy a mostrar el resultado final, porque creo que es suficientemente ilustrativo como para entender el proceso que se ha seguido.
El nombre de cada nodo se corresponde con el orden en que se han añadido a la lista de abiertos. Cada vez que abrimos un nodo, añadimos sus sucesores empezando por el de arriba y siguiendo en orden horario. Obviamente, añadimos solo los que sean una configuración permitida (no un obstáculo).
El heurístico usado es la distancia Manhattan.



nodo sel	sucesor	$g + h = f$	
S	n_1	$1 + 3 = 4$	Cerrado 2º
	n_2	$0'7 + 2'7 = 3'4$	Cerrado 1º
n_2	n_3	$1'4 + 3'4 = 4'8$	Cerrado 3º
n_1	n_4	$1'7 + 3'7 = 5'4$	Cerrado 6º
	n_5	$2 + 4 = 6$	Cerrado 7º
n_3	n_6	$2'4 + 2'4 = 4'8$	Cerrado 4º
n_6	n_7	$3'4 + 1'4 = 4'8$	Cerrado 5º
n_8	n_8	$4'4 + 2'4 = 6'8$	Cerrado 8º[1]
n_4	n_9	$2'4 + 4'4 = 6'8$	
	n_{10}	$2'7 + 4'7 = 7'4$	
n_5	n_{11}	$2 + 5 = 7$	
	n_{12}	$2'7 + 4'7 = 7'4$	
n_8	n_{13}	$5'1 + 1'7 = 6'8$	Cerrado 9º
n_{13}	n_{14}	$5'8 + 1 = 6'8$	Cerrado 10º
n_{14}	n_{15}	$6'5 + 1'7 = 8'2$	
	F	$6'8 + 0 = 6'8$	
F			SOLUCIÓN


S
1
5
2
4
11
12
3
9
10
6
7
8
13
14
F