

Sistemas Inteligentes



Félix Gómez Mármol

3º Ingeniería Informática

Índice general

1. Elementos de un Problema	5
1.1. Problemas y Representación de Problemas	5
1.1.1. Representación mediante Estados	5
1.1.2. Representación por Reducción	5
1.2. Características de los Problemas	6
1.2.1. Problemas Descomponibles	6
1.2.2. Problemas Recuperables e Ignorables	6
1.2.3. Problemas de cualquier camino y de mejor camino	6
1.2.4. Consistencia y papel del conocimiento	6
1.3. Características de los Procesos de Búsqueda	7
1.3.1. Búsqueda de la Solución. Árboles y Grafos	7
1.3.2. Razonamiento hacia delante y hacia atrás	8
1.3.3. Representación de Conocimiento y el Problema de la Estructura	8
1.3.4. Operadores	8
1.3.5. Heurísticas	8
1.4. Introducción a la Organización de Control	9
2. Métodos Básicos para la Resolución de Problemas	11
2.1. Introducción	11
2.1.1. Catalogación de los Problemas	12
2.1.2. Elementos para describir los Algoritmos	13
2.1.3. Evaluación de los Algoritmos	13
2.2. Búsqueda a Ciegas sobre una representación mediante Estados	14
2.2.1. Búsqueda Primero en Anchura	14
2.2.2. Búsqueda de Costo Uniforme	14
2.2.3. Búsqueda Primero en Profundidad	15
2.2.4. Búsqueda Primero en Profundidad Limitada	15
2.2.5. Búsqueda Primero en Profundidad Iterativa	15
2.2.6. Búsqueda Bidireccional	16
2.3. Búsqueda en Grafos	17
2.3.1. Problemas sin Sensores	18
2.4. Búsqueda a Ciegas sobre una representación mediante Reducción	20
2.4.1. Algoritmo de Búsqueda en Árboles YO	21
2.4.2. Búsqueda Primero en Anchura sobre Árboles YO	21
2.4.3. Búsqueda Primero en Profundidad sobre Árboles YO	22
2.4.4. Búsqueda Primero en Profundidad Limitada sobre Árboles YO	22
3. Introducción a los Sistemas Inteligentes Artificiales	23
3.1. Definición de Sistema Inteligente Artificial	23
3.2. Estructura de un Sistema Inteligente Artificial	24
3.2.1. Fases de los Sistemas	24

3.3. Objetivos Actuales	24
3.4. Aplicaciones	25

Capítulo 1

Elementos de un Problema

1.1. Problemas y Representación de Problemas

Los sistemas de resolución de problemas están compuestos por tres elementos principales:

- **Datos o Representación:** Describe la situación del problema, el objetivo que queremos alcanzar y la situación inicial.
 - Representación mediante estados
 - Representación por reducción
- **Operadores:** Permiten manejar y modificar la representación.
- **Estrategia de Control:** Define la secuencia de operadores que debemos aplicar para conseguir el objetivo.

1.1.1. Representación mediante Estados

Representa el problema como una colección de elementos (instantes del problema), entre los que hay un estado inicial, un estado objetivo y estados intermedios.

Para transitar de un estado a otro aplicamos los operadores siguiendo las indicaciones que dicte la estrategia de control.

Ejemplo 1.1 *El problema del 8-puzzle es un problema típico de representación mediante estados.*

1	2	3
4	5	
7	8	6

 \implies

1	2	3
4	5	6
7	8	

Operadores: $\leftarrow, \rightarrow, \uparrow, \downarrow$

1.1.2. Representación por Reducción

Partimos de un problema que descomponemos en subproblemas más sencillos hasta llegar a un subproblema de solución inmediata.

A continuación deberemos recomponer la solución final, a partir de las soluciones inmediatas halladas.

Ejemplo 1.2 *Resolver una integral.*

$$\int (x + x^2) dx \left\{ \begin{array}{l} \int x dx \\ \int x^2 dx \end{array} \right.$$

1.2. Características de los Problemas

1.2.1. Problemas Descomponibles

Un problema es descomponible si se puede descomponer en un conjunto de subproblemas independientes más sencillos.

Ejemplo 1.3 *El problema de resolver la integral*

$$\int (x^2 + 3x + \sin^2 x \cdot \cos^2 x) dx$$

es un problema descomponible ya que

$$\int (x^2 + 3x + \sin^2 x \cdot \cos^2 x) dx = \int x^2 dx + \int 3x dx + \int \sin^2 x \cdot \cos^2 x dx$$

Ejemplo 1.4 *El problema de apilar y desapilar bloques no es descomponible porque los subproblemas no son independientes entre sí (no se puede, por ejemplo, desapilar un bloque que no esté en la cima).*

1.2.2. Problemas Recuperables e Ignorables

Un problema es ignorable cuando parte del camino (desde el punto de partida hasta el objetivo) es ignorable porque no entorpece para encontrar la solución (aunque sí ha consumido recursos -tiempo y esfuerzo-).

Ejemplo 1.5 *Demostración de un teorema en la que se demuestran proposiciones que luego no se utilizan para demostrar realmente el teorema.*

Un problema es recuperable si los pasos hacia la solución se pueden deshacer.

Ejemplo 1.6 *Si en el problema del 8-puzzle hacemos movimientos “inútiles”, podemos deshacerlos más tarde.*

No ocurre así, por ejemplo, en el juego del ajedrez.

1.2.3. Problemas de cualquier camino y de mejor camino

Los problemas de mejor camino, requieren explorar todos los caminos hasta la solución, mientras que los de cualquier camino son mucho menos exhaustivos.

Ejemplo 1.7 *Dado un conjunto de axiomas, demostrar otro es un problema de cualquier camino.*

El problema del viajante de comercio es un problema de mejor camino.

1.2.4. Consistencia y papel del conocimiento

El conocimiento del problema debe ser consistente, es decir, el punto de partida no puede ser contradictorio, así como tampoco puede serlo la solución encontrada.

Ejemplo 1.8 *¿Qué debemos hacer para acertar en una diana a 50 metros de distancia con una bala a 500 km/h?*

Para resolver el problema suponemos que la trayectoria de la bala es una línea recta. ¡Inconsistencia!, pues la bala describirá una parábola.

En resumen, no podemos suponer conocimiento que haga inconsistente el problema.

En cuanto al papel del conocimiento, existen problemas que requieren poco conocimiento para ser resueltos, mientras que en otros ocurre todo lo contrario.

Ejemplo 1.9 *El juego del ajedrez, en el que sólo es preciso conocer la reglas del juego, es un ejemplo de problema que requiere poco conocimiento.*¹

Ejemplo 1.10 *El problema de analizar una noticia en el periódico y que el ordenador la explique, requiere de mucho conocimiento: diccionarios, reglas gramaticales, dobles sentidos, etc.*

1.3. Características de los Procesos de Búsqueda

1.3.1. Búsqueda de la Solución. Árboles y Grafos

Las estructuras de datos subyacentes que van a aparecer respecto a los dos tipos de representaciones que hemos visto (estados y reducción) son los árboles y los grafos.

La diferencia entre ambos es que en los grafos existen ciclos (y por lo tanto suelen ser más difíciles de tratar), mientras que en los árboles no.

Así, emplearemos un grafo o un árbol dependiendo de las características del problema.

Representación mediante Estados

Emplearemos árboles y grafos ordinarios en los que a cada nodo se le pueden aplicar n operadores para obtener n nodos hijo nuevos.

Si en nuestro problema aparecerán pocos estados repetidos, usaremos un árbol, en caso contrario, trataremos con un grafo.

Llamaremos **solución** a la secuencia de operadores empleados para llegar desde el estado inicial hasta el estado final. Es decir, el camino seguido (en el caso de un árbol) desde el nodo raíz hasta el nodo hoja que es solución del problema.

Puede suceder que lleguemos al mismo estado solución por distintos caminos. La elección de uno u otro depende del enunciado del problema (camino más corto, menor coste, etc).

Algunas situaciones que podemos encontrar que dependen del tipo de solución son:

- Que el estado solución no esté en la estructura, por lo tanto, no existe solución.
- Que el estado solución sea único.
- Que el estado solución sea múltiple (debe existir algún criterio para seleccionar uno de ellos).

Representación por Reducción

Emplearemos árboles YO en los que cada nodo representa un subproblema simple (nodo O) o un conjunto de subproblemas a resolver (nodo Y).

Un nodo que no se descompone o simplifica se llama nodo terminal. Un nodo terminal con solución se corresponde con un problema primitivo y se llama Primitiva.

Si al aplicar un operador se produce un conjunto de subproblemas solución alternativos, entonces se genera un nodo O. Si por el contrario se produce un conjunto de subproblemas que deben ser resueltos necesariamente, entonces se produce un nodo Y.

Un nodo de un árbol YO tiene solución (es resoluble) si se cumple alguna de las siguientes condiciones:

¹El resto de conocimiento “sólo” sirve para limitar la búsqueda

- Es un nodo primitiva
- Es un nodo no terminal de tipo Y y sus sucesores son todos resolubles.
- Es un nodo no terminal de tipo O y alguno de sus sucesores es resoluble.

1.3.2. Razonamiento hacia delante y hacia atrás

En un razonamiento hacia delante partimos de lo que conocemos para intentar alcanzar nuestro objetivo, mientras que en un razonamiento hacia atrás ocurre lo contrario: partimos de la solución e intentamos llegar a la situación inicial.

Para determinar qué tipo de razonamiento es mejor en cada caso debemos considerar lo siguiente:

- ¿Existen más estados inicio posibles o estados meta? Deberíamos movernos hacia el conjunto de estados más grande (así es más fácil encontrarlo).
- ¿En qué dirección es el factor de ramificación² más grande? Deberíamos proceder en la dirección en la que el factor de ramificación sea más pequeño.

Otra posibilidad es emplear ambos razonamientos, dando lugar a una búsqueda bidireccional, la cual no sabemos cuándo acabará (probablemente cuando ambas búsquedas se corten).

1.3.3. Representación de Conocimiento y el Problema de la Estructura

El problema de la representación del conocimiento nos plantea si debemos representar de forma global el estado o subproblema o de forma mínima uniendo varias representaciones.

Ejemplo 1.11 *En el problema del 8-puzzle, podríamos representar cada estado como un vector que indica qué operador aplicamos sobre qué pieza (representación mínima), o bien representar el tablero entero (representación global).*

El problema de la estructura está relacionado con la unidad más pequeña a la que se aplica un operador, pues éste debe determinar qué transformaciones tienen lugar sobre el estado al que se aplica y qué cosas quedan inalteradas en dicho estado.

1.3.4. Operadores

Se deben definir operadores genéricos (no específicos de un estado concreto), que cumplan unas condiciones.

$$\text{Operador} \left\{ \begin{array}{l} \text{Precondiciones} \\ \text{Adición} \\ \text{Supresión} \end{array} \right.$$

El proceso de emparejamiento consiste en que si existe un unificador que haga ciertas las precondiciones de un operador a partir del estado actual, podremos aplicar dicho operador.

1.3.5. Heurísticas

Información adicional adquirida a través de la experiencia, que nos permite resolver los problemas de manera más eficiente.

Una función heurística nos mide cuánto de bueno es un estado para seguir expandiéndolo o elegir otro para avanzar.

²Nº medio de nodos hijo que tiene un nodo

1.4. Introducción a la Organización de Control

Existen dos modalidades para implementar todo lo anteriormente dicho:

1. Programación orientada a los procedimientos.
El objeto está disgregado por todos los procedimientos.
2. Programación orientada a los objetos.

Otra forma de actuar sería dividir el problema y repartir el trabajo entre distintos agentes que se comunican entre sí para dar lugar a la solución final.

Capítulo 2

Métodos Básicos para la Resolución de Problemas

2.1. Introducción

Veamos un agente encargado de resolver un problema, para centrarnos en el procedimiento de búsqueda sin ningún tipo de información (búsqueda a ciegas):

Función agente_resolvedor_problema(Percepción) devuelve una acción

```
Estado := actualizar_estado(Estado,Percepción);
Si secuencia está vacía entonces hacer
  Objetivo := Formular_objetivo(Estado);
  Problema := Formular_problema(Estado,Objetivo);
  Secuencia := Búsqueda(Problema);
Acción := Primero(Secuencia);
Secuencia := Resto(Secuencia);
Devolver Acción;
```

1

Formular_objetivo nos dice dónde queremos llegar, mientras que Formular_problema calcula el camino más corto desde el estado inicial hasta el objetivo y Búsqueda devuelve la secuencia de acciones.

Observar que el agente no devuelve la secuencia, sino una acción, por lo tanto es capaz de manipular su entorno.

Cuando aplicamos un agente como éste suponemos que:

- El entorno es estático, ya que suponemos que la búsqueda de la solución y la aplicación de la misma son independientes.
- Se conoce el estado inicial (ya que existen problemas en los que esto no se puede asegurar).
- El entorno es discreto, esto es, existe un número finito de acciones a aplicar (lo que se traduce en un factor de ramificación finito).
- El entorno es determinista, no aparece incertidumbre en la acción.

Para el caso de emplear la representación mediante estados tendremos en cuenta los siguientes elementos:

¹Se comprueba que la secuencia esté vacía porque en caso contrario el agente ya ha resuelto el problema

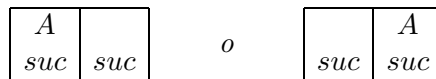
- Estado inicial.
- Descripción de los operadores aplicables.
- Test objetivo: Dice si un estado intermedio pertenece al conjunto de estados finales.
- Función costo: Mide el costo asociado a la aplicación de cierto operador.

2.1.1. Catalogación de los Problemas

1. Problemas de juguete: Problemas preparados para aplicar en el laboratorio. Sirven para probar la eficiencia de los algoritmos.
2. Problemas del mundo real: Son problemas obtenidos del ambiente o entorno. Muchas veces no podemos probar la eficiencia del algoritmo.

Ejemplo 2.1 *El problema de la aspiradora es un problema de juguete.*

- *Estados iniciales:*



- *Estados posibles:* $2 \times 2^2 = 8$ (en general $n \cdot 2^n$).
- *Test objetivo:* Función que comprueba si el estado actual pertenece al conjunto de estados finales.



- *Función sucesor:* Operadores que puedo aplicar en cada estado
 - Mover a la izquierda
 - Mover a la derecha
 - Aspirar
- *Función Costo:* En este problema el costo de pasar de un estado a otro es siempre el mismo.

Ejemplo 2.2 *El problema del 8-puzzle es otro problema de juguete.*

- *Estados:* Disponemos de 8 piezas y un blanco.
- *Estado inicial:* Configuración elegida.
- *Sucesor:* Mover blanco (arriba, izquierda, derecha, abajo)
- *Test objetivo:* Comprobar que le estado actual es el estado objetivo.
- *Función Costo:* El mismo para todos los operadores.
- *Configuraciones alcanzables:*
 - 8-puzzle: $\frac{9!}{2} = 181440$ estados
 - 15-puzzle: 1'3 millones de estados
 - 8-puzzle: 10^{23} estados

Ejemplo 2.3 *Problemas del mundo real:*

- *Búsqueda de una ruta aérea.*
- *Problemas turísticos.*
- *El viajante de comercio.*
- *Distribución de circuitos (VSI)*
- *Búsqueda en internet*

Obsérvese que el espacio de estados está formado por todos los estados en los que se puede estar según el problema, mientras que el espacio de búsqueda lo constituyen todos los posibles caminos dentro del espacio de estados.

2.1.2. Elementos para describir los Algoritmos

Cada nodo (estado) del espacio de búsqueda contiene:

- Una descripción del estado al que se refiere.
- Una descripción del nodo padre.
- Una mención a la acción aplicada.
- Costo del camino hasta el estado.
- Profundidad del estado.

Además, en todo momento el algoritmo cuenta con lo que se conoce como frontera.

La frontera es el conjunto de nodos que se han generado pero que todavía no han sido estudiados (no han sido expandidos).²

De ese conjunto, que implementaremos como una cola, el procedimiento de búsqueda va cogiendo los nodos a estudiar.

2.1.3. Evaluación de los Algoritmos

Los criterios de evaluación de estos algoritmos son:

1. Completitud: Un algoritmo es completo si encuentra una solución cuando ésta exista.
2. Optimalidad: Un algoritmo es óptimo si encuentra la solución, si existe, y es óptima.
3. Complejidad en tiempo: Tiempo que tarda en encontrar la solución.
4. Complejidad en espacio: Cantidad de memoria que necesita el algoritmo.

Para medir la complejidad en tiempo y memoria, haremos uso de los siguientes parámetros:

- b : factor de ramificación.
- d : profundidad del nodo objetivo más superficial, es decir, el más cercano al nodo inicial.
- m : longitud máxima de cualquier camino en el espacio de búsqueda.

El tiempo lo mediremos en términos del número de nodos generados en la búsqueda, mientras que el espacio lo mediremos en términos del número de nodos que podemos almacenar en memoria.

La complejidad del algoritmo se puede medir como la complejidad de tiempo y espacio, o bien la complejidad en tiempo y espacio más la complejidad de hallar y aplicar la solución.

²También se conoce como conjunto de abiertos

2.2. Búsqueda a Ciegas sobre una representación mediante Estados

2.2.1. Búsqueda Primero en Anchura

En este algoritmo la frontera es una cola FIFO (primero en entrar, primero en salir) en la que se insertan los nodos al final sin importar el orden de inserción (pues todos los nodos pertenecen al mismo nivel). Los nodos se recorren por niveles.

Evaluación de la Eficiencia

1. Es completo. Si existe solución la encuentra, pues hace un barrido del espacio de búsqueda.
2. Es óptimo dependiendo del problema, pues encuentra el camino más corto, pero no necesariamente el de menor costo.
3. Complejidad en tiempo.
El número de nodos generados es $b + b^2 + b^3 + \dots + b^d + (b^{d+1} - 1) \in O(b^{d+1})$
4. Complejidad en espacio. $O(b^{d+1})$ ya que necesitamos que todos los nodos estén en memoria para poder reconstruir la solución.

Ejemplo 2.4 Suponiendo un factor de ramificación $b = 10$, que se generan 10000 nodos/s y que ocupa en memoria 1000 bytes/nodo, se tiene la siguiente tabla:

Profundidad	Nodos	Tiempo	Memoria
2	1100	11 seg	1 Mega
4	11100	5 min	106 Megas
6	10^7	19 min	10 Gigas
8	10^9	31 horas	1 Terabyte
10	10^{11}	129 días	101 Terabytes
12	10^{13}	35 años	10 Petabytes
14	10^{15}	3523 años	1 Exabyte

2.2.2. Búsqueda de Costo Uniforme

Para cada nodo j se tiene su costo asociado desde el nodo inicial hasta dicho nodo j , calculado como

$$g(j) = g(i) + C[i, j]$$

donde $C[i, j]$ es el costo de aplicar un operador para pasar del nodo i al j .

Ahora la frontera es una cola ordenada según el coste, de menor a mayor. Por lo tanto, antes de llegar a la solución se estudiarán todos los nodos con menor coste que los nodos del camino óptimo.

Evaluación de la Eficiencia

1. Es completo siempre que los costes sean positivos (en caso contrario el algoritmo podría quedar en un bucle infinito, pues el costo del camino sería cada vez menor).
2. Complejidad en tiempo y espacio.
Sea c^* el costo de la solución óptima y sean todos los costos $c_i \geq \varepsilon > 0$. Así la complejidad en tiempo y espacio $\in O\left(b^{\frac{c^*}{\varepsilon}}\right)$.

2.2.3. Búsqueda Primero en Profundidad

La frontera se implementa ahora como una cola LIFO (último en entrar, primero en salir).

Ahora no hay que almacenar en memoria todos los nodos, sino sólo aquellos que formarán parte de la solución.

Evaluación de la Eficiencia

1. No es completo, pues si entra en una rama infinita, el algoritmo no termina nunca.
2. No es óptimo, pues devuelve la primera solución que encuentra, que no tiene por qué ser la óptima.
3. Complejidad en tiempo: $O(b^m)$
4. Complejidad en espacio: $O(bm)$ si para cada nodo expandimos todos sus descendientes o $O(m)$ si expandimos los nodos uno a uno.

2.2.4. Búsqueda Primero en Profundidad Limitada

El algoritmo es igual que el anterior con una cota máxima de profundidad, denotada por l , de modo que cuando se llega a dicha profundidad no se expanden los nodos, aunque tengan hijos.

Evaluación de la Eficiencia

1. Es completo dependiendo de l .
Si $l \geq d$ es completo.
Si $l < d$ no es completo.
2. Es óptimo dependiendo de l
Si $l < d$ no es óptimo.
Si $l > d$ la solución no tiene por qué ser la óptima.
Si $l = d$ encontramos la solución de camino más corto, pero no tiene por qué ser la de menor costo.
3. Complejidad en tiempo: $O(b^l)$
4. Complejidad en espacio: $O(bl) \rightarrow O(l)$

2.2.5. Búsqueda Primero en Profundidad Iterativa

Consiste en aplicar búsqueda primero en profundidad limitada con $l = 1, 2, 3, 4, \dots$. Es decir, incrementando l hasta encontrar la solución.

Hace un barrido como la búsqueda en anchura aunque es mejor que ésta, ya que sólo almacena en memoria los nodos que forman parte de la solución. Sin embargo tiene el inconveniente de generar varias veces los mismos nodos.

Evaluación de la Eficiencia

1. Es completo, ya que realiza un barrido del espacio de búsqueda.
2. No es óptimo, ya que encuentra la solución de camino más corto, no necesariamente la de menor costo.
3. Complejidad en tiempo y espacio: $O(b^d)$

Ejemplo 2.5 Sean $b = 10$ y $d = 5$, los nodos generados en una búsqueda primero en anchura e iterativa son:

Iterativa: $5 \cdot 10 + 4 \cdot 10^2 + 3 \cdot 10^3 + 2 \cdot 10^4 + 10^6$

Anchura: $10 + 10^2 + 10^3 + 10^4 + 10^5 + (10^6 - 10)$

2.2.6. Búsqueda Bidireccional

Es necesario poder definir operadores inversos a los que tenemos así como el nodo objetivo. Por ejemplo, esto no es posible en el ajedrez, pero sí en el 8-puzzle.

El costo de la búsqueda bidireccional es $O\left(b^{\frac{d}{2}} + b^{\frac{d}{2}}\right) = O\left(2b^{\frac{d}{2}}\right) = O\left(b^{\frac{d}{2}}\right)$, por lo tanto menor que el de la búsqueda unidireccional.

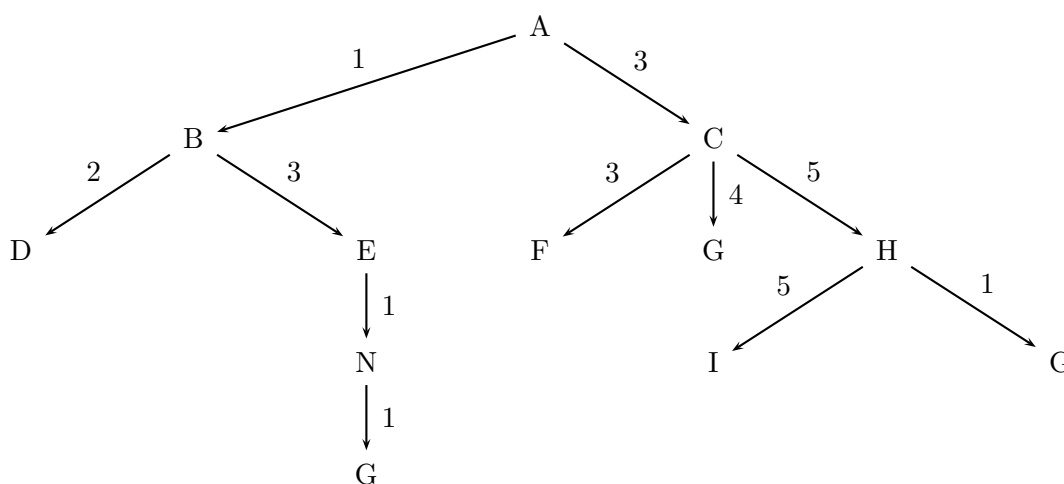


Figura 2.1: Árbol del espacio de búsqueda

Ejemplo 2.6 Muestra el contenido de la frontera durante todo el proceso de búsqueda (el nodo objetivo es el G) para el árbol de la figura 2.1 siguiendo todos los algoritmos vistos anteriormente.

<u>Anchura</u>	
A	
B, C	
C, D, E	Costo 7
D, E, F, G, H	Nodos 8
E, F, G, H	Solución: A-C-G
F, G, H, N	
G, H, N	
<u>Costo Uniforme</u>	
A	
B = 1, C = 3	
C = 3, D = 3, E = 4	Costo 6
D = 3, E = 4, F = 6, G = 7, H = 8	Nodos 10
E = 4, F = 6, G = 7, H = 8	Solución: A-B-E-N-G
N = 5, F = 6, G = 7, H = 8	
G = 6, F = 6, G = 7, H = 8	

Primero Profundidad

A
 B, C Costo 6
 D, E, C Nodos 7
 E, C Solución: A-B-E-N-G
 N, C
 G, C

Profundidad Limitada

$l=1$	$l=2$	$l=3$	$l=4$	
A	A	A	A	
B, C	B, C	B, C	B, C	Para $l = 2$:
	D, E, C	D, E, C	D, E, C	Costo 7
	E, C	E, C	E, C	Nodos 11
	C	N, C	N, C	Solución: A-C-G
	F, G, H	C	G, C	
	G, H	F, G, H		
		G, H		

Profundidad Iterativa

$l=1$	$l=2$	
A	A	
B, C	B, C	Costo 7
	D, E, C	Nodos 11
	E, C	Solución: A-C-G
	C	
	F, G, H	
	G, H	

Resumen Estrategias

	Primero Anchura	Costo Uniforme	Primero Profundidad
Completo	$b \neq \infty \Rightarrow SI$	$b \neq \infty, c > 0 \Rightarrow SI$	NO
Óptimo	$c_i = K \forall i \Rightarrow SI$	SI	NO
Tiempo	$O(b^{d+1})$	$O\left(b \frac{c^*}{\epsilon}\right)$	$O(b^m)$
Espacio	$O(b^{d+1})$	$O\left(b \frac{c^*}{\epsilon}\right)$	$O(bm)$

	Profundidad Limitada	Profundidad Iterativa	Bidireccional (si es posible)
Completo	NO	$b \neq \infty, c > 0 \Rightarrow SI$	Anchura $\Rightarrow SI$
Óptimo	NO	$c_i = K \forall i \Rightarrow SI$	Anchura $\Rightarrow SI$
Tiempo	$O(b^l)$	$O(b^d)$	$O\left(b^{\frac{d}{2}}\right)$
Espacio	$O(bl)$	$O(bd)$	$O\left(b^{\frac{d}{2}}\right)$

2.3. Búsqueda en Grafos

Para evitar el inconveniente de los estados repetidos que aparecían en los anteriores apartados, vemos ahora la búsqueda en grafos.

Definición 2.1 El conjunto de cerrados es el conjunto de los nodos que ya han sido expandidos.

1	0	0	0
0	0	1	0
0	1	0	0
1	0	0	0

Cuadro 2.1: Tablero de juego

Mediante conjunto de *cerrados* se realiza el control de los nodos repetidos.

La modificación que se hace consiste en comprobar que el nodo que vayamos a introducir en la frontera no se encuentre ya en ella.

- Si el nodo SI está en Cerrados → Lo eliminamos.
- Si el nodo NO está en Cerrados → Lo insertamos.
- Si el nodo SI está en la Frontera → Nos quedamos con el óptimo.
- Si el nodo NO está en la Frontera → Lo insertamos.

Ejercicio 2.1 Dado el tablero 2.1, se tiene un peón que se puede colocar en cualquier casilla de la primera fila.

Desde una casilla con un 0, el peón puede moverse hacia abajo, si el cuadrado de abajo es un 0; si el cuadrado de abajo es un 1, el peón sólo puede moverse horizontalmente.

Desde una casilla con un 1, no puede realizar ningún movimiento.

El objetivo es alcanzar una casilla con un 0, en la cuarta fila.

¿Sería mejor resolverlo con un árbol o con un grafo?

¿Con qué estrategia lo resolverías?

2.3.1. Problemas sin Sensores

Hasta ahora sólo hemos visto problemas completos, en los que el espacio de estados es observable y determinista y en los que se conoce perfectamente los cambios producidos por la aplicación de cada operador.

En los problemas sin sensores, no hay ningún sensor que nos diga en qué estado nos encontramos. Por lo tanto, no conocemos el estado inicial (supondremos que sí conocemos lo que produce cada operador o acción).

Ejemplo 2.7 El mundo de la aspiradora.

1.

ASP	
SUC	SUC

5.

ASP	
	SUC

2.

	ASP
SUC	SUC

6.

	ASP
	SUC

3.

ASP	
SUC	

7.

ASP	

4.

	ASP
SUC	

8.

	ASP

Tenemos ahora estados de creencia, es decir, estados en los que creo estar.

El estado inicial sería la configuración de estados donde creo empezar, en nuestro caso: {1, 2, 3, 4, 5, 6, 7, 8}

Si movemos la aspiradora a la derecha, llegamos al estado {2, 4, 6, 8}.

Si ahora aspiramos llegamos al estado {4, 8}.

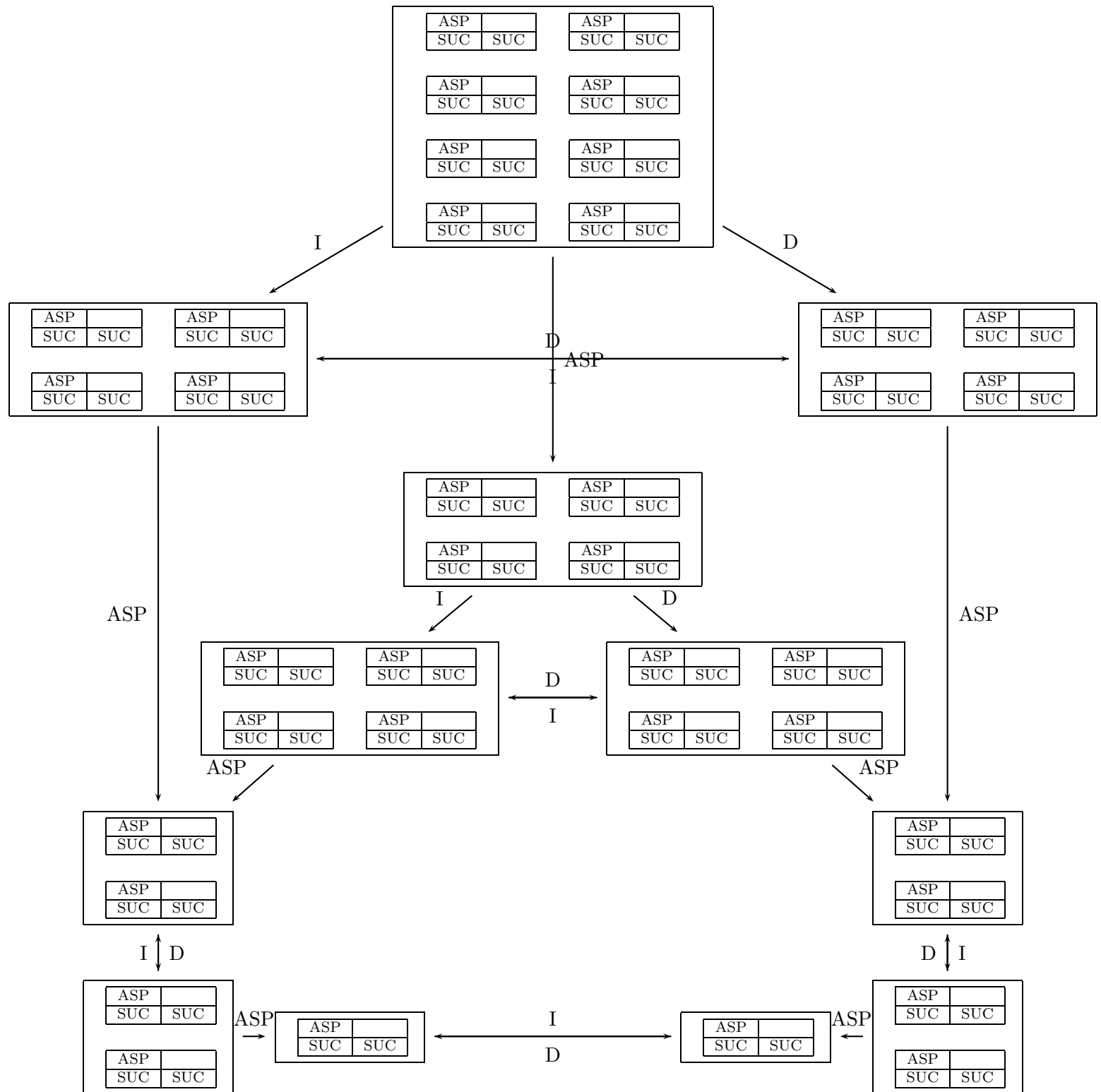


Figura 2.2: Grafo completo del problema de la aspiradora

*Y si aplicamos izquierda y aspiramos llegamos al estado final {7}.
El grafo completo es el mostrado en la figura 2.2.*

2.4. Búsqueda a Ciegas sobre una representación mediante Reducción

En la representación mediante estados partíamos del estado inicial. Ahora partimos del problema que queremos resolver y aplicamos operadores hasta llegar a subproblemas primitivos.

Este proceso es factible si los subproblemas son más sencillos que el problema del que se descomponen.

La solución ahora es la secuencia de operadores que nos produce u subárbol de búsqueda que conecta el problema inicial con los problemas primitivos. La búsqueda de dicha solución es una búsqueda hacia atrás, ya que partimos del problema final.

Ejemplo 2.8 Dado el árbol YO de la figura 2.3, en el que los nodos enmarcados identifican a los problemas primitivos, las posibles soluciones son:

- 1,2,4,8,9
- 1,3,5,6,7,10
- 1,3,5,6,7,11

Es decir, los subárboles mostrados en la figura 2.4.

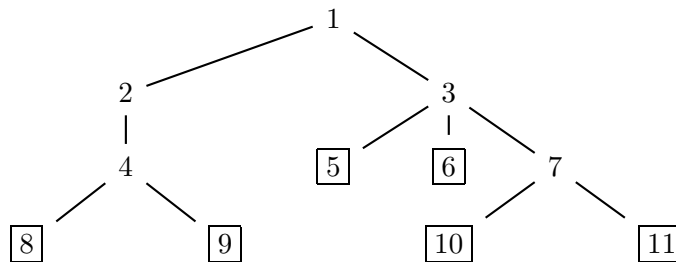


Figura 2.3: Árbol YO del ejemplo 2.8

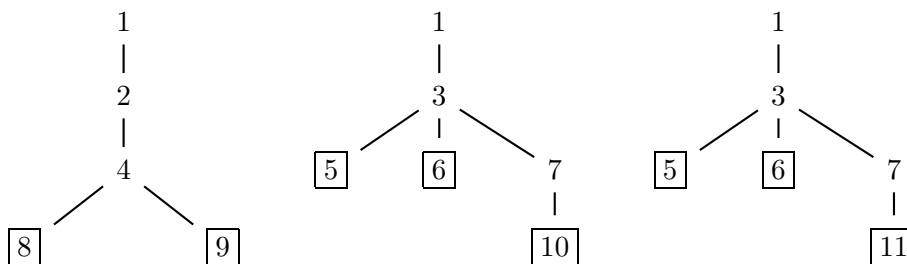


Figura 2.4: Soluciones del Árbol YO del ejemplo 2.8

Para simplificar el problema nosotros supondremos que:

1. Sólo trabajaremos con árboles (no con grafos).
2. Cada subproblema se resolverá independientemente de los demás y en el orden que queramos.

2.4.1. Algoritmo de Búsqueda en Árboles YO

Definición 2.2 La frontera es el conjunto de subproblemas que han sido generados pero que todavía no se han estudiado (expandido).

Definición 2.3 Función **EXPANDIR-YO(nodo)**

Para cada sucesor, m , de nodo

Si m es un conjunto de subproblemas (e.d. es un nodo Y) entonces

Generar los sucesores de m e insertarlos en SUC

Sino (m es un nodo O) entonces

Insertamos m en SUC.

Devolver SUC

Ejemplo 2.9 El resultado de expandir el nodo 1, en el árbol de la figura 2.3, sería $\{2, 5, 6, 7\}$

Definición 2.4 Algoritmo de Búsqueda en Árboles YO

En cada paso del bucle del algoritmo, hacemos:

1. Tomamos el primer nodo de la frontera, llamémosle n_1 .
2. Insertamos en la frontera todos los sucesores del nodo n_1 según la función EXPANDIR-YO.
3. Si n_1 no tiene sucesores, lo marcamos como irresoluble

2.4.2. Búsqueda Primero en Anchura sobre Árboles YO

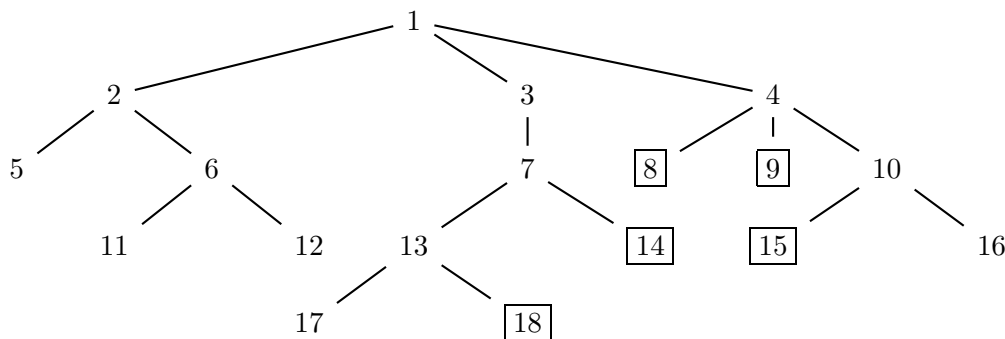


Figura 2.5: Árbol YO de ejemplo

- La Frontera es una cola FIFO.
- Añadimos los sucesores al final de la frontera.
- El primer nivel de anchura para el árbol de la figura 2.5 es $\{5, 6, 3, 8, 9, 10\}$.

Ejemplo 2.10 A continuación se muestra el contenido de la frontera en cada pasada del algoritmo, para el árbol de la figura 2.5:

1ª pasada	1					
Frontera:	5	6	3	8	9	10
	5	6	3	10		

2ª pasada

Frontera: 6 3 10
 Marcamos 5 como irresoluble
 \Rightarrow 2 irresoluble
 \Rightarrow quitamos hijos de 2
 3 10

3ª pasada

Frontera: 10
 10 13 14
 10 13

4ª pasada

Frontera: 13
 13 15 16
 13 16

Como se puede observar en la figura 2.6, la solución obtenida es la óptima.

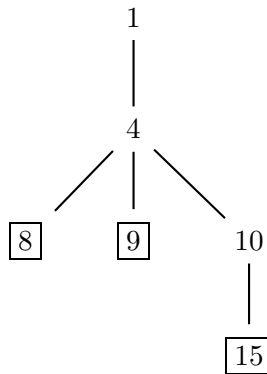


Figura 2.6: Solución del ejemplo 2.10

2.4.3. Búsqueda Primero en Profundidad sobre Árboles YO

- La Frontera es una cola LIFO.
- Añadimos los sucesores al final de la frontera.
- Si una rama es infinita, el algoritmo no para, aunque exista solución.

2.4.4. Búsqueda Primero en Profundidad Limitada sobre Árboles YO

Igual que el anterior, pero con una cota máxima de profundidad.

Capítulo 3

Introducción a los Sistemas Inteligentes Artificiales

3.1. Definición de Sistema Inteligente Artificial

A continuación se presentan las definiciones que la Real Academia Española de la Lengua da a los términos “sistema”, “inteligencia” e “inteligencia artificial”.

Definición 3.1 Sistema.-

Conjunto de elementos que ordenadamente relacionados entre sí contribuyen a un determinado objetivo.

Denotamos como entorno a lo que rodea al sistema.

Definición 3.2 Inteligencia

Capacidad de entender o comprender. || Capacidad de resolver problemas. || Conocimiento, comprensión, acto de entender. || Sentido en que se puede tomar una sentencia, un dicho o una expresión. || Habilidad, destreza y experiencia.

Definición 3.3 Inteligencia Artificial

Desarrollo y utilización de ordenadores con los que se intenta reproducir los procesos de la inteligencia humana. || Conjunto de técnicas que empleando la informática permite la realización de operaciones hasta ahora exclusivas de la inteligencia humana.

Veamos ahora nuestras propias definiciones:

1. **Definición Cualificada.-¹**

La inteligencia artificial es la parte de la ciencia de los computadores concerniente con el diseño de computadores inteligentes, es decir, sistemas que exhiben las características que asociamos con la inteligencia en la conducta humana.

2. **Definición Operacional.-²**

Aplicamos una prueba a un sistema inteligente y a un humano para saber si una máquina es tan inteligente como una persona.

PC	Humano
Realizador de la Prueba	

¹Definición dada por los expertos en el tema

²Definición dada por Alan Turing

3. Definición por Extensión.-

Inteligencia es una capacidad o conjunto de ellas para desarrollar un conjunto de propiedades presentes en los seres biológicos en distintos grados, según su escala evolutiva.

De entre esas propiedades, destacamos las siguientes:

- Interpretar en términos semánticos un entorno.
- Tomar decisiones acerca de las acciones a desarrollar.
- Deducir a partir de un conjunto de hechos conocidos.
- Aprender de una realidad dada de hechos.
- Abstractar.
- Planificar o simular la evolución de los modelos.

Definición 3.4 Artificial

Lo creado por el ser humano.

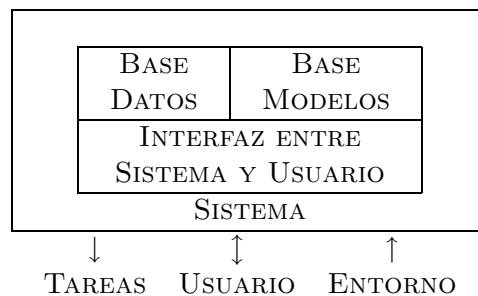
Definición 3.5 Inteligencia Artificial

Máquina con comportamiento humano con las anteriores propiedades.

Definición 3.6 Sistema Inteligente Artificial

Sistema que aprende durante su existencia y que actúa continuamente de forma interna o externa de manera que alcanza su objetivo cada vez mejor.

3.2. Estructura de un Sistema Inteligente Artificial



3.2.1. Fases de los Sistemas

- Introducimos la información en el sistema sobre el entorno.
- Los datos son tratados en el preproceso, eliminando lo que no necesitamos.
- Aprendizaje de la máquina.
- Damos información de cómo hemos obtenido la información y/o realimentamos de nuevo la máquina.

3.3. Objetivos Actuales

1. Desarrollo de entornos de programación que ayuden a construir sistemas inteligentes o sistemas basados en el conocimiento.
2. Desarrollo de aplicaciones.
3. Desarrollo de sistemas de comunicación en lenguaje natural.
4. Desarrollo de entornos de automatización.

3.4. Aplicaciones

1. Construcción de sistemas expertos (es un caso particular de sistema inteligente en el que la información ha sido aportada por expertos en una determinada área).
2. Conocer la demostración de teoremas matemáticos.
3. Robots que modelan algo.

