## TEMA 6: La lógica y la deducción, la representación

## del conocimiento, aprendizaje automático,

## Introducción

En general, podemos afirmar que un problema consiste en

• una descripción de la situación de la que se parte;

• una descripción de la situación a la que se quiere llegar;

• una descripción de los medios de que disponemos para alcanzar nuestro objetivo.

En el contexto de la Informática, a partir de un problema, se intenta construir un

sistema que lo resuelva. Resumidamente, las acciones para construir un sistema que resuelva un problema serían:

• Definir el problema con precisión (especificación del problema), habitualmente a partir de un enunciado del problema expresado en lenguaje natural:

•de qué se parte;

•cuál es el objetivo.

• Analizar el problema: información para elegir las técnicas.

• Aislar y representar el conocimiento necesario para resolver el problema.

•Elegir la mejor técnica que resuelva el problema y aplicarla al problema

particular.

Particularizar estas acciones al contexto de la Inteligencia Artificial consiste en

elegir las técnicas de resolución entre aquellas que son utilizadas en esta disciplina.

En nuestro caso, nos centraremos en los siguientes temas en las técnicas

relacionadas con los sistemas de producción y la búsqueda en espacios de estados.

En los temas siguientes utilizaremos como ejemplos para ilustrar las nociones

teóricas algunos problemas que no son aplicaciones reales de la Inteligencia

Artifical. Se trata más bien de ejemplos que han sido elegidos porque son

suficientemente manejables, tanto en cuanto a la facilidad de su enunciado como en cuanto al volumen de datos que es necesario explorar para obtener una solución.

He aquí algunos de ellos:

El 8-puzzle.

•Entrada: Un tablero 3x3 donde aparecen distribuidos los dígitos 1, 2, 3, 4, 5,

6, 7 y 8, quedando por tanto una de las casillas del tablero en blanco. Por

ejemplo, un tablero tal y como aparece en la Figura 1.



Figura 1

•Salida: Alcanzar una disposición distinta de los dígitos en el tablero.

Habitualmente se toma como configuración de llegada el tablero que tiene

el dígito 1 en la primera casilla de la primera línea (esquina superior

izquierda) y los dígitos 2, 3, … , 8 se distribuyen uno a continuación de

otro en el sentido de las agujas del reloj, quedando la posición central del

tablero sin ocupar (ver Figura 2)



Figura 2

•Medios: si una casilla del tablero está al lado de la casilla vacía se puede

desplazar sobre ella. (En particular, está prohibido usar un destornillador

para sacar las piezas y resituarlas como a uno le parezca … )

El problema de las garrafas de vino.

•Entrada: dos garrafas vacías, una de cuatro litros y otra de tres.

•Salida: la garrafa de cuatro litros contiene exactamente dos litros de vino.

•Medios: se dispone de un depósito con mucho vino (más de 100 litros … ) y las únicas operaciones permitidas son llenar cada garrafa en el depósito, vaciarlas en el depósito y pasar contenido de una garrafa a la otra, hasta que la primera se vacíe o la segunda se llene. (En particular, no es lícito suponer que se dispone de otra garrafa que puede contener exactamente dos litros … )

• El problema del granjero, el lobo, el carnero y la lechuga.

•Entrada: un granjero, un lobo, un carnero y una lechuga se encuentran en la orilla derecha de un río.

•Salida: todos ellos han pasado a la orilla izquierda del río.

•Medios y restricciones: se dispone de una barca que debe ser conducida por el granjero y que sólo tiene capacidad para uno de los otros elementos; el lobo se comerá al carnero si los deja juntos sin compañía en uno de los lados; el carnero se comerá la lechuga si los deja solos. 

**Ejercicios.** A partir de las siguientes descripciones de los problemas que se

indican, detallar con mayor precisión la entrada, la salida y los medios de

resolución.

1) *El problema del viajante*. Un representante de comercio tiene que visitar una

serie de ciudades viajando entre ellas por carretera. Dispone de un plano de la

región con la información de las distancias entre las ciudades. Desea partir de la ciudad en la que está para volver a ella tras haber visitado el resto de ciudades, pero habiendo hecho el mínimo número de kilómetros posible.

2) *Las n reinas*. Se trata de situar n reinas (siendo n un número entero positivo

mayor que 3) del ajedrez en un tablero n x n (el normal tiene 8 x 8 casillas) de modo que no se "coman" entre sí.

## Sistemas de producción y búsqueda

Si particularizamos la definición de problema dada en el apartado anterior del

siguiente modo:

•Entrada: una descripción del estado inicial del mundo.

•Salida: una descripción (parcial) del estado del mundo deseado.

•Medios: una descripción de acciones que puedan transformar un estado del mundo en otro, acciones que son consideradas como operadores o reglas.

Tendremos el marco conceptual que permite resolver el problema por medio de un sistema de producción.

Un *sistema de producción* consiste en:

•una base de datos/hechos/conocimiento con información sobre el problema;

•un conjunto de reglas (operadores);

•una estrategia de control;

•un aplicador de reglas: ciclo de reconocimiento-actuación.

La parte más importante de los sistemas de producción es la estrategia de control (también llamada, en el contexto de los sistemas expertos, mecanismo o motor de inferencia). Es la que establece el proceso de búsqueda de la solución. Si la descripción del problema viene dada por una noción de estado (del mundo) podemos entender que un sistema de producción es empleado para una búsqueda en un espacio de estados (así se relacionan estos dos paradigmas clásicos de la Inteligencia Artificial).

Con esta perspectiva, uno de los cometidos de la estrategia de control consiste en decidir qué operador aplicar en cada momento:

•¿se puede aplicar?

•¿produce algún cambio en el estado?

•¿qué estado elegir para aplicar los operadores?

•¿qué sucede si hay varios operadores posibles a aplicar? (resolución de conflictos)

Los operadores, que admiten una interpretación como reglas, son acciones simples que pueden transformar un estado en otro. Generalmente se distinguen en los operadores una parte izquierda (el patrón o condición), que determina la aplicabilidad de la regla (se dice también que si se verifica la condición la regla queda *sensibilizada*), y una parte derecha que describe la operación o acción a llevar a cabo al aplicar el operador (o *disparar* la regla).

Así, una regla u operador consta de dos partes:

•Precondición: descripción parcial del estado del mundo que debe ser verdad para realizar una acción.

•Instrucciones para crear el nuevo estado.

Taxonomías de los sistemas de producción/búsqueda/estrategia de control:

•Hacia adelante / hacia atrás / bidireccional

•Irrevocable / tentativa

•Informada / no informada

Ventajas de los sistemas de producción:

•Separación de conocimiento (reglas) y control (ciclo de reconocimientoactuación).

•Modularidad de las reglas/operadores:

• no hay interacciones sintácticas entre reglas;

• la comunicación se establece a través de ciertas estructuras de

almacenamiento comunes, que se denominan de modo genérico

memoria de trabajo. (No es estrictamente necesario que exista una

variable o estructura de datos explícita con ese nombre.)

•Control dirigido por patrones:

• programas en IA necesitan flexibilidad;

• reglas pueden dispararse en cualquier secuencia;

• estado > reglas aplicables > camino de la solución.

•Traza y explicación:

• secuencia de aplicación de reglas;

• cada regla es una pieza de conocimiento con su justificación para

un cambio de estado.

•Independencia del lenguaje:

• el modelo es independiente de la representación elegida para

reglas y memoria de trabajo;

• es posible siempre que el lenguaje soporte el reconocimiento de

patrones.

•Modelo plausible del mecanismo humano de resolución de problemas.

Intérpretes generales de sistemas de producción:

•Lenguajes basados en reglas:

• OPS5 (Brownston, en 1985, de CMU, en Common Lisp usa RETE);

• CLIPS (C Language Integrated Production System) (de NASA,

permite la integración con lenguajes como C y Ada).

•Lenguajes basados en lógica: el más extendido es PROLOG.

•Armazones de sistemas expertos:

• EMYCIN;

• en entornos de IA (KEE, KnowledgeCraft, LOOPS, … ).

•Arquitecturas generales de resolución de problemas:

• GPS (Newell, Shaw, Simon) 1963;

• SOAR (Newell, Laird, … ) 1987.

## Representación de Problemas

Una vez decidido que el problema planteado va a ser resuelto por medio de un

sistema de producción/búsqueda en el espacio de estados, debemos elegir una

representación adecuada para el problema. Esto debe ser resuelto en tres niveles:

•Conceptual.

•Lógico.

•Físico.

y en cada uno de ellos deberemos elegir representaciones para dos cosas:

•Estados.

•Operadores.

El nivel conceptual hace referencia a qué consideramos como estado del problema y qué como regla u operador válido, pero independientemente de cualquier estructura de datos o descripción de algoritmos que vayamos a usar. En el nivel lógico se elige una estructura de datos para los estados y se determina el formato de codificación de los operadores (puede ser también una estructura de datos o se puede hacer proceduralmente). Por último en el nivel físico se concreta para el lenguaje de programación elegido la implementación de estados y operadores determinada en el nivel lógico. Los niveles lógico y físico están confundidos en ocasiones y, además, la fase clave para un procesamiento eficiente está en el nivel conceptual.

Por ejemplo, en el 8-puzzle se podría decir que el estado incluye información sobre el material o el tamaño del tablero, etc. Sin embargo, nada de eso es relevante. Basta con conocer la "situación relativa" de los dígitos.

En general, la noción de estado elegida debería describir:

•todo lo que es necesario para resolver el problema;

•nada que no sea necesario para resolver el problema.

Para el problema del 8-puzzle:

•localización de cada casilla y el hueco;

•nada más.

Obsérvese que, un vez fijado el nivel conceptual, todavía tenemos muchas opciones en el nivel lógico:

•matriz 3 x 3;

•vector de longitud 9;

•conjunto de hechos: {(superior-izda = 2), (superior-centro = 1), … }.

(A su vez, dependiendo de las estructuras permitidas en un lenguaje de

programación concreto, para cada una de ellas son posibles diferentes elecciones, pero esto nos interesa menos: es más un problema de programación que de IA.)

En cualquier caso, la elección de la representación para los estados está muy

ligada a la que se haga para los operadores, puesto que ambos elementos deben "cooperar" para la resolución del problema.

En cuanto a los operadores, deben ser tan generales como sea posible para así reducir el número de reglas distintas. En el caso del 8-puzzle:

•4 x 9! operadores para pasar de un estado cualquiera a sus, como máximo 4, estados sucesores;

•4 x 8 operadores que mueven cualquier ficha arriba, abajo, derecha o

izquierda;

•4 operadores que mueven el hueco arriba, abajo, derecha o izquierda.

Evidentemente, la última representación es la más adecuada.

La elección de una representación para los estados y los operadores define

implícitamente una relación de adyacencia en el conjunto de estados factibles para el problema. A este conjunto con la estructura de adyacencia definida se le denomina *espacio de estados*. El espacio de estados es una noción clave en los sistemas de producción, puesto que es el lugar en el que se va a desarrollar la búsqueda, y en el que se van a poder comprender las distintas variantes

involucradas (en particular, las taxonomías: hacia adelante/hacia atrás, etc.).

Un fragmento del espacio de estados para el 8-puzzle, donde se ha elegido como representación para los operadores los 4 operadores citados antes, puede verse en la Figura 3.

Veamos qué sucede con la representación del problema de las garrafas de vino.

Conceptualmente, los únicos datos relevantes en cada momento para el sistema son las cantidades de vino que contienen las garrafas de cuatro y tres litros (puesto que la cantidad de vino que haya en el depósito no influye en la resolución de problema, como ha quedado fijado al explicar los medios de que se dispone en el problema). Los operadores sobre esos estados serían:

•llena-cuatro: llenar, cogiendo del depósito, la garrafa de cuatro litros;

•llena-tres: llenar, cogiendo del depósito, la garrafa de tres litros;

•vacia-cuatro: vaciar en el depósito la garrafa de cuatro litros;

•vacia-tres: vaciar en el depósito la garrafa de tres litros;

•echa-la-cuatro-a-la-tres: echar de la garrafa de cuatro litros en la garrafa de tres;

•echa-la-tres-a-la-cuatro: echar de la garrafa de tres litros en la garrafa de cuatro.

En el nivel lógico, un estado sería identificado como un par de números (x,y) donde, por ejemplo, x es el número de litros que contiene la garrafa de cuatro litros e y el número de litros de la garrafa de tres litros. En este nivel cada uno de los 6 operadores anteriores deberían ser especificado como una regla, con una precondición (o premisa) y una acción asociada (o consecuencia). Por ejemplo:

llena-cuatro (x, y):

precondición: x < 4;

acción: construir el estado (4,y).

**Ejercicios.**

1) Especificar como reglas el resto de operadores.

2) Dibujar el espacio de estados correspondiente, a partir del estado inicial (0,0)

hasta que encontréis un estado objetivo, es decir un estado de la forma (2,y).

## Espacios de búsquedas

El espacio de estados es el conjunto de estados factibles para un problema junto a la estructura de adyacencia definida implícitamente por los operadores o reglas. Según la estructura que los operadores definan sobre el conjunto de estados hablaremos de árbol o grafo de estados. Las posibilidades relevantes (por cómo influyen en el proceso de búsqueda) son:

•Arbol

•Grafo dirigido acíclico

•Grafo dirigido (no necesariamente acíclico)

•Grafo no dirigido

Empezando por el final, la estructura de *grafo no dirigido de estados* aparece

cuando si de un estado e se puede pasar a un estado e' por aplicación de un

operador siempre existe otro operador (o el mismo) que permite pasar de e' a e.

Este es el caso, por ejemplo, del 8-puzzle con la representación con la que venimos trabajando. Con esta estructura, si no se toman las medidas oportunas, la búsqueda puede caer en ciclos que impidan la terminación de la ejecución.

Si no se da la circunstancia anterior, existen estados e, e' tales que de e se puede pasar a e', pero de e' no se puede pasar a e. En este caso, la dirección u orientación de la adyacencia es importante y se habla de *grafo dirigido de estados*.

Por ejemplo, en la representación dada para el problema de las garrafas, del estado (3,0) se puede pasar al estado (0,0) (aplicando el operador vacia-cuatro), pero no se puede pasar (directamente; esto es, por aplicación de una única regla) de (0,0) a (3,0). Pero, en general, que no se pueda pasar directamente no significa que no se pueda pasar por una secuencia de aplicaciones de reglas. Eso posibilita la aparición de ciclos (dirigidos) como el siguiente: (0,0) --> (0,3) --> (3,0) --> (0,0). En este caso, la aparición de bucles infinitos en la búsqueda que se citaron para los grafos no dirigidos de estados también pueden reproducirse y será necesario tenerlo en cuenta en el diseño de estrategias de control.

Si en el grafo dirigido no aparecen ciclos (dirigidos) hablaremos de *grafo dirigido acíclico de estados*. Para poner un ejemplo de un tal espacio de estados, pensemos en la siguiente representación para el problema de las n reinas. Conceptualmente, los estados son configuraciones del tablero en las que, como mucho, aparecen n reinas (pudiendo no aparecer ninguna: el estado inicial). Tenemos n2 reglas que consisten en poner una reina en cada una de las casillas, siendo las únicas precondiciones el que no haya ya una reina en esa casilla y que todavía queden reinas disponibles (o dicho de otro modo: que en el tablero aparezcan menos de n reinas). En esta situación es claro que se puede llegar a una configuración de más de un modo, pero que no podemos encontrar ciclos dirigidos (puesto que cada aplicación de una regla aumenta el número de reinas en el tablero y nunca quitamos reinas). Si el espacio de estados es un grafo dirigido acíclico no aparecerán problemas de no terminación de la ejecución como los evocados en los dos casos anteriores. Sin embargo, si en el sistema de producción no se toman las precauciones adecuadas, un mismo estado (y todo el subgrafo de sus descendientes) puede ser explorado en muchas ocasiones con el consiguiente perjuicio en la eficiencia (en tiempo y en espacio) del sistema.

Por último, si el espacio de estados es un grafo dirigido acíclico en el que a cada estado se puede llegar sólo de una manera tenemos el caso de un *árbol de estados*. Esta es la estructura combinatorial más sencilla y la que produce menos problemas al realizar la búsqueda en ella. Para encontrar un ejemplo, pensemos en la misma representación para el problema de las n reinas que hemos dado en el caso del grafo dirigido acíclico. La única modificación es que ahora para que una regla sea disparada es necesario que haya una y exactamente una reina en cada una de las filas precedentes a la de la casilla que corresponde a la regla (o que el tablero esté vacío en el caso de reglas correspondientes a casillas de la primera fila). Esta ligera modificación hace que el espacio de estados sea un árbol, lo que simplifica considerablemente la labor de búsqueda de una solución. Este ejemplo muestra claramente que la fase de diseño conceptual es la más importante para conseguir un sistema de producción eficaz.

Es claro que un sistema de producción que considere los espacios de estados más generales (los que corresponden a grafos no dirigidos) funcionará correctamente para aquellos espacios más especializados. Sin embargo, en ocasiones, cuestiones de eficiencia aconsejan que las estrategias de control se particularicen para espacios de estados que son árboles o grafos dirigidos acíclicos.

Otras clasificaciones de los espacios de estados se refieren a su simplicidad, su finitud o infinitud, y a la existencia de pesos en los arcos.

Comenzando de nuevo por el final, los espacios de estados con pesos en las aristas aparecen en sistemas de producción en los que el disparo de las reglas se supone que tiene un coste. Habitualmente en estos casos los problemas de búsqueda aparecen bajo la forma de problemas de optimización.

Para hablar de *espacios de estados infinitos* (todos los que aparecen en los

problemas anteriores son finitos) vamos a introducir toda una clase de sistemas de producción que tienen un interés teórico (son una fuente de ejemplos para ilustrar las diferencias entre distintos tipos de espacios de estados y distintas estrategias de control), histórico (fueron introducidos por el lógico Post en los años 30 para estudiar problemas de calculabilidad teórica y son la base matemática para los actuales sistemas de producción) y práctico (muchos sistemas importantes, en particular los relacionados con el tratamiento del lenguaje natural, están directamente basados en ellos). Nos refererimos a los *sistemas (simbólicos) de reescritura*. En ellos los estados son palabras (secuencias de caracteres) en un cierto alfabeto (los elementos del alfabeto son los caracteres) y las reglas son del tipo siguiente: <cadena1> --> <cadena2>. Una de estas reglas se activa sobre una cadena cuando ésta contiene como subcadena a <cadena1>; el nuevo estado obtenido por aplicación de la regla es la cadena inicial en la que se ha reemplazado

una ocurrencia de <cadena1> por <cadena2>.

Por ejemplo, en un sistema de reescritura con alfabeto {A,B} y reglas A --> AA, AB --> B, cualquier estado inicial que contenga una A tendrá una rama infinita.

Esto puede producir problemas de no terminación de la búsqueda de una índole diferente a los comentados en los casos de los grafos con ciclos. Pero que un espacio de estados sea infinito no significa que en él la búsqueda no pueda tener éxito. En el ejemplo anterior, si el estado inicial es ABAB y los estados objetivos son aquellos en los que no aparece el caracter A, es sencillo encontrar una secuencia de aplicación de reglas que llevan del estado inicial al estado objetivo BB.

En el mismo ejemplo anterior puede verse que una misma regla puede estar

sensibilizada sobre un mismo estado por varias razones distintas, lo que producirá varios disparos distintos de la misma regla sobre el mismo estado. Así sobre ABAB la primera regla está sensibilizada dos veces y los estados que se obtienen por los disparos son, respectivamente, AABAB y ABAAB. En el caso anterior, los dos estados construidos son distintos pero puede también suceder que dos disparos distintos de la misma regla sobre el mismo estado produzcan un único estado. Es el caso de la primera regla del ejemplo, que es sensibilizada tres veces sobre AABAB.

Las dos sensibilizaciones correspondientes a las dos primeras ocurrencias de A producirán como resultado un solo estado: AAABAB. En este último caso (cuando entre dos vértices aparece más de una arista dirigida) se dice que el *grafo de estados* es *no simple*. En la mayoría de las aplicaciones los *espacios de estados* que aparecen son *simples* (compruébese con todos los ejemplos aparecidos hasta el momento), por lo que en el resto del texto nos olvidaremos de la posibilidad de que algunos grafos de estados pueden ser no simples.

Terminamos este apartado haciendo una comparación entre la teoría algorítmica de grafos y la búsqueda en Inteligencia Artificial. Los principales esquemas algorítmicos (recorrido en anchura, en profundidad, caminos óptimos, algoritmos voraces, etc.) son comunes a ambos campos, pero en la teoría de grafos se supone que el grafo que se recorre, o en el que se busca, es explícitamente conocido. Se trata de un dato para el problema que habitualmente está almacenado en una estructura de datos (lo que excluye el tratamiento de grafos infinitos). Sin embargo, la búsqueda en espacios de estados no supone que el árbol o grafo esté previamente generado. No asume tampoco que se tengan que generar todos los estados (sólo serán tratados los estados necesarios para resolver el problema).

Esto es imprescindible si se tiene en cuenta que los problemas que ataca la

Inteligencia Artificial suelen ser sumamente costosos en espacio y tiempo. Lo

anterior es cierto incluso en los ejemplos que parecen más inofensivos. Por ejemplo, el grafo del 8-puzzle tiene … ¡362.880 vértices! (explosión combinatoria). De hecho, en la búsqueda en espacios de estados, ni siquiera es necesario que el espacio de estados aparezca de ningún modo reconocible dentro de los algoritmos (aunque el espacio de estados siempre será un marco en el que entender el desarrollo de la ejecución de los algoritmos). Si el espacio de estados es (parcialmente) almacenado o tenido en cuenta por el algoritmo de búsqueda, obtenemos la clase más importante de sistemas de producción / estrategias de búsqueda: la *exploración en grafos de estados*. (Habitualmente en este tipo de sistemas de producción el almacenamiento consiste en listas de *nodos*: estados enriquecidos con algún tipo de información que, en particular, codifica las relaciones de adyacencia

**¿Qué significa encontrar una solución?**

Para alcanzar una solución del problema debemos disponer de una descripción de un estado deseado del mundo. Está descripción puede ser completa (por ejemplo, diciendo exactamente cuál es el estado objetivo, caso del 8-puzzle) o puede ser parcial (describiendo algunas propiedades de los estados que consideramos objetivo, como sucede en el problema de las garrafas). En este segundo caso, la forma más habitual de describir los estados objetivos es una función que recibe como argumento un estado y devuelve verdad si el estado es un estado objetivo y falso en otro caso.

En general, la solución consiste en una secuencia de operadores que transforman el estado inicial en el estado objetivo. Un ejemplo de solución para el 8-puzzle (tal y como ha sido enunciado más arriba) aparece en la Figura 12.

Pero podemos resolver de este modo dos problemas distintos:

*• El problema de decisión*: consiste simplemente en decir si sí o no se puede alcanzar el objetivo desde el estado inicial.

*• El problema de explicación*: en este caso se debe indicar cómo se ha llegado desde el estado inicial hasta el objetivo.

En ocasiones, el estado objetivo mismo ya contiene toda la información necesaria para resolver los dos problemas (por ejemplo, en las n reinas), pero en general no será cierto (por ejemplo el 8-puzzle). En este último caso, el sistema de producción debe disponer de un modo de almacenamiento de los caminos que van siendo explorados.

En cualquiera de los dos problemas, como hemos dicho, la resolución consiste en encontrar una secuencia de operadores que transformen el estado inicial en el estado objetivo, por lo que tiene sentido estudiar en ambos casos las siguientes restricciones sobre las *secuencias solución*:

• Encontrar la secuencia más corta.

• Encontrar la secuencia menos costosa (en este caso se supondrá que los disparos de las reglas tienen asociado un coste o, equivalentemente, que el espacio de estados tiene pesos en las aristas).

• Encontrar cualquier secuencia lo más rápido posible.

En general, al evaluar una estrategia habrá que tener en cuenta el siguiente

compromiso:

tiempo de cálculo <==> calidad de la solución.



Una última variante para todos los problemas de búsqueda requiere no sólo que se encuentre *un* camino solución, sino *todos* los que existan (por ejemplo, en el problema de las n reinas no se trataría de encontrar una configuración del tablero en la que las n reinas no se coman, sino todas las configuraciones con esa propiedad). Estas variantes se denominan genéricamente *problemas de la*

*Biblioteca Británica* (con tiempo y papel suficiente, unos cuantos monos provistos de máquinas de escribir serían capaces de re-escribir todos los volúmenes del British Museum … ).

## Juegos, sistemas expertos, Estrategias de control no informadas

## Esquema algorítmico genérico de búsqueda

Veamos una primera aproximación a un algoritmo genérico de búsqueda en un

espacio de estados. Habitualmente los algoritmos de búsqueda trabajan con estructuras de datos llamadas nodos que contienen en particular información sobre un estado. El esquema algorítmico siguiente está expresado usando esas estructuras.



En los pasos 6-7 se encuentra implicada la parte del sistema de producción que dispara las reglas: el ciclo de reconocimiento (en el paso 6) y actuación (aplicación de O a E). El modo concreto en que se organice ese condicional dependerá de las decisiones de diseño que hayamos adoptado previamente. En particular, esta versión del algoritmo parece excluir que un mismo operador pueda ser aplicado varias veces sobre el mismo estado (véase el ejemplo del sistema de reescritura tratado en 2.5), pero no es difícil adaptarla para que cubra todos los casos.

El tipo de búsqueda depende del paso número 7 del algoritmo anterior. Se tratará de *búsquedas no informadas* si el algoritmo no conoce nada del problema en concreto que debe resolver. En nuestro contexto eso quiere decir que el paso 7 se realiza con criterios independientes del dominio del problema.

Un concepto importante que aparece cuando la estrategia de control reposa sobre la noción de nodo es el de *árbol de búsqueda*. El árbol de búsqueda es la estructura combinatorial que va siendo construida en el proceso de búsqueda. Sus vértices son nodos y existe una arista dirigida de un nodo a otro si el segundo es construido a partir del primero en el paso 7 del algoritmo anterior. Esta noción está íntimamente relacionada con la de espacio de estados, pero es necesario no confundir ambas. En particular, la estructura combinatorial construida en la ejecución del algoritmo es siempre un árbol puesto que los nodos en el paso 7 son construidos y no reutilizados. Esto es independiente de que nodos distintos (porque han sido construidos en distintos momentos de la ejecución) tengan asociado el mismo estado. Si esta situación se produce, significa que en el espacio de estados hay ciclos (dirigidos o no) y por tanto que se trata de un grafo de estados y no de un árbol de estados.

Otra noción importante es la de *profundidad de un nodo* (en el árbol de búsqueda): se trata del número de aristas que tiene el camino que lo une con el nodo raíz (ese camino es único, por tratarse de un árbol).

## Búsqueda en anchura

En este tipo de búsqueda, el paso 7 del algoritmo anterior se particulariza del

siguiente modo: los nuevos nodos son añadidos *al final* de la lista ABIERTOS. Así se consigue que los nodos en ABIERTOS estén ordenados según su profundidad, en orden decreciente: los menos profundos al principio, los más profundos al final. La lista ABIERTOS tiene de este modo una forma de acceso que la convierte en una *cola* (o fila de espera): los datos que primero entran (paso 7) son los primeros en salir (paso 3).

Los nodos de igual profundidad se ordenan arbitrariamente. En realidad, en el

algoritmo precedente, ese orden arbitrario es directamente heredado del que exista entre los operadores del sistema de producción (en el bucle '**para cada** operador O **hacer**').

Mediante esta estrategia, el árbol de búsqueda se va generando por niveles de

profundidad. Hasta que todos los nodos de un nivel no han sido revisados no se revisa ninguno del siguiente nivel. Para facilitar la comprensión del proceso de búsqueda vamos a introducir algunos espacios de estados artificiales (que no provienen de ningún sistema de producción real), que serán árboles o grafos etiquetados por letras. Las letras representan los estados.

Por ejemplo, en la figura 1 aparece un árbol de estados, con estado inicial I y con un único estado objetivo O.



Introduciremos ahora una notación para describir el árbol de búsqueda. En el árbol de búsqueda no se dibujarán los nodos (cuya estructura, en general, puede ser compleja) sino tan solo los estados (letras, en los ejemplos artificiales) y junto a ellos la información relevante sobre los nodos (en el caso de la búsqueda en anchura, ninguna). La información relativa a la adyacencia entre los nodos queda recogida en el gráfico por las flechas dirigidas. En el árbol de búsqueda aparecerán vértices correspondientes a todos los nodos que van siendo generados (es decir , todos aquellos que en algún momento del proceso hayan estado en la lista de ABIERTOS) y se numerarán secuencialmente (con números subrayados) los nodos conforme son examinados.

El árbol de búsqueda correspondiente a la búsqueda en anchura en el espacio de estados de la figura 1 ha sido recogido en la figura 2. Puesto que los nodos de igual profundidad generados a partir de un mismo padre se ordenan de un modo arbitrario, es necesario indicar cuál es el criterio seguido. En el ejemplo, hemos expandido los estados de "izquierda a derecha".



**Ejercicio**. Dibujar, con las convenciones anteriores, el árbol de búsqueda en

anchura para el espacio de estados de la figura 1, con expansión de "derecha a

izquierda". Compárese el número de estados generados y examinados con respecto al árbol de la figura 2.

La búsqueda en anchura es una *estrategia exhaustiva*: el recorrido por niveles

examina todos los estados que son accesibles desde el estado inicial. En particular, si un estado objetivo es alcanzable, la búsqueda en anchura lo encuentra. La demostración es sencilla: el estado objetivo aparecerá en un cierto nivel de profundidad y, en un número finito de pasos (suponiendo un número finito de operadores y un número finito de posibles aplicaciones de un operador sobre un estado), llegará a ese nivel. Este mismo razonamiento muestra que la búsqueda en anchura siempre encuentra la secuencia solución más corta (una de las que tienen el mínimo número de aristas). Atención: esto no significa en absoluto que realice un número pequeño de operaciones. Se trata de una propiedad de la secuencia solución encontrada, no del proceso por el que ha sido construida (de hecho, veremos que la búsqueda en anchura puede llegar a ser extremadamente ineficaz).

De lo anterior se sigue que, si existe solución, incluso en un espacio de estados con ciclos, como el del problema de las garrafas (la Figura 3 presenta *parte* de ese espacio de estados), el algoritmo de búsqueda en anchura no entrará en bucles infinitos, aunque sí que explorará en numerosas ocasiones la misma zona del espacio de estados, lo que puede perjudicar considerablemente su eficiencia.



**Ejercicio.** Estudiar cómo sería el árbol de búsqueda para el problema de las

garrafas con la exploración en anchura en la versión que conocemos.

Sin embargo, si el estado objetivo no es alcanzable y el espacio de estados

contiene ciclos dirigidos, la búsqueda en anchura en esta versión no terminará

nunca la ejecución. Como ejemplo, consideremos una variante del problema de las garrafas en el que las dos garrafas tienen capacidades de 4 y 2 litros y el objetivo es conseguir que en la garrafa de cuatro litros haya exactamente 3. Es fácil comprobar que este problema no tiene solución (las garrafas siempre contendrán un número par de litros) y en él la búsqueda en anchura, si no tomamos las precauciones adecuadas, entraría en un bucle infinito.

**Ejercicio.** Representar ese problema de las garrafas. Dibujar el correspondiente grafo de estados. Comprobar que el árbol de búsqueda en anchura es infinito.

Por tanto, la versión genérica explicada en el apartado anterior deberá ser adaptada para cubrir todos los casos posibles de los espacios de estados. Sin embargo, esta versión es muy adecuada, por ser más eficaz que sus variantes, cuando el espacio de estados es un árbol. Este es, por ejemplo, el caso del problema de las n reinas.

**Ejercicios.** Consideremos el problema de las 4 reinas. Se pide:

1) comenzar a dibujar el espacio de búsqueda en anchura con la representación del problema;

2) modificar la definición de regla sensibilizada para evitar que se examinen

configuraciones del tablero que no pueden llevarnos a una solución;

3) estudiar cómo cambia el árbol de búsqueda en anchura con la modificación del ejercicio anterior.

En cuanto a la estimación de la eficacia de la búsqueda en anchura, podemos

identificar genéricamente la complejidad en tiempo con el número de nodos

generados (esta simplificación ignora, entre otras cosas, los costes de la

verificación de la aplicabilidad de las reglas y de aplicación de las mismas) y la

complejidad en espacio espacio con la longitud máxima que puede alcanzar la lista ABIERTOS.

En las complejidades influyen dos factores que son denotados b y d. El número b es el *factor de ramificación* (branching factor): media del número de nodos generados desde un nodo. El número d mide la *profundidad* del estado objetivo (aquí se supone que existe solución): mínimo del número de aplicaciones de reglas necesarias para llegar del estado inicial a un estado objetivo.

No es difícil entonces comprobar que la complejidad en tiempo es de orden

exponencial O(bd). Puesto que en ABIERTOS se extraen los elementos de uno en uno pero se añaden (en media) de b en b, concluimos que la complejidad en

espacio es también exponencial O(bd). Esta complejidad espacial hace que el

algoritmo sea impracticable para problemas grandes. Por ejemplo, para el problema del 8-puzzle, b = 3, d = 20 y bd = 3.486.784.401. En el caso de las garrafas de vino, b = 3, d = 6, bd = 729.

En resumen, las características de la búsqueda en anchura en esta primera versión son las siguientes. Si existe una solución en el espacio de estados

(independientemente de la estructura combinatorial de éste), la encuentra y además con la secuencia solución más corta. Si el espacio de estados es un grafo dirigido acíclico la ejecución termina en todos los casos (si no existe solución, habría que imponer además que el grafo fuese finito), pero puede ser muy ineficaz por explorar en muchas ocasiones las mismas ramas (en particular, en ABIERTOS pueden aparecer muchos nodos con el mismo estado asociado). Si el grafo tiene ciclos dirigidos (o es un grafo no dirigido), el algoritmo puede no terminar si no existe solución y si ésta existe el algoritmo puede ser muy ineficaz por las mismas razones que en el caso de los grafos dirigidos acíclicos.

Por todo lo anterior, vemos que este algoritmo sólo es recomendable en el caso de que el espacio de estados sea un árbol. Por eso, lo denominaremos *búsqueda en anchura en árbol*, aunque ya hemos visto que puede funcionar correctamente (es decir, encontrar una solución) en casos más generales. Independientemente de estas consideraciones, podemos afirmar que la búsqueda en anchura es demasiado ineficaz como para ser aplicada

en casos prácticos reales.

**Ejercicio.** Como ejemplo de que en un espacio de estado infinito si existe una

solución la búsqueda en anchura la encuentra, descríbase el árbol de búsqueda en anchura del sistema de reescritura presentado en el apartado 2.5, donde el estado inicial es ABAB y son estados objetivo aquellas cadenas en las que no aparece A.

## Búsqueda en grafo

Se puede considerar una nueva versión del esquema genérico de búsqueda en el que los estados ya examinados son almacenados en una lista llamada (de estados) CERRADOS. Cuando se expande un estado sólo dan lugar a nuevos nodos aquellos estados que no *aparezcan* en ABIERTOS o que no *estén* en

CERRADOS. Nótese que ABIERTOS es una lista de nodos (mientras que

CERRADOS es una lista de estados), por lo que lo que habrá que comprobar es que un estado no es igual a ninguno de los estados asociados a los nodos

ABIERTOS.

El esquema genérico que permite realizar búsquedas seguras y no redundantes en espacios de estados que sean grafos (dirigidos o no, acíclicos o no) aparece más abajo.

Vemos que las únicas diferencias con el esquema que hemos denominado

búsqueda en árbol son:

•En la línea 1' se inicializa CERRADOS con la lista vacía.

•En la línea 5' se añade el estado que ya ha sido examinado a CERRADOS.

•En las líneas 7' y 7'' se comprueba si el estado generado es nuevo.



Se podrían considerar variantes en las que las líneas 7' o 7'' no son incluidas.

Particularizando a la búsqueda en anchura las consecuencias serían las que

siguen. Si no incluimos la línea 7'' tendremos una versión que no cometerá cierto tipo de redundancias (en concreto, nunca re-examinará dos estados que

aparecieran en la versión anterior simultáneamente en ABIERTOS), pero no tiene en cuenta otras redundancias (cuando vuelve a aparecer un estado que ya ha sido examinado; nótese que esto puede darse tanto en grafos dirigidos acíclicos como en otros que tengan ciclos dirigidos) y, sobre todo, puede entrar en bucles infinitos si los estados objetivos no son alcanzables y hay ciclos dirigidos. Si eliminamos la línea 7' (manteniendo la 7'') este último problema no puede darse, pero un estado puede ser examinado varias veces (puesto que puede aparecer en dos nodos distintos de ABIERTOS).

En conclusión, la versión con las dos líneas tiene las siguientes propiedades en el caso de la búsqueda en anchura:

• Cada estado del espacio de estados es examinado, a lo más, una vez.

• Si existe solución, encuentra una de longitud mínima, independientemente del tipo de espacio de estados de que se trate (árbol, grafo dirigido acíclico, grafo dirigido o no-dirigido y cada uno de ellos finito o infinito).

• Si no existe solución, siempre terminará la ejecución salvo en el caso (inevitable para las estrategias exhaustivas) en el que el espacio de estados es infinito.

Como se ve las propiedades teóricas de la búsqueda en anchura en grafo son

bastante buenas. Si embargo, no hay que olvidar que dicha mejora tiene un precio en cuanto al coste en tiempo (gestión de CERRADOS y algoritmos de búsqueda en ABIERTOS y CERRADOS) y, sobre todo, en espacio (la lista CERRADOS siempre aumenta, nunca se eliminan elementos de ella), por lo que si sabemos que el espacio de estados es un árbol será preferible utilizar la versión inicial. Por ejemplo, sería un derroche utilizar esta versión que acabamos de dar con la representación elegida para el problema de las n reinas.

Este algoritmo de búsqueda en grafo permite clasificar a los estados en cuatro

clases. En un instante cualquiera del proceso de búsqueda, distinguimos 4 tipos de estados:

• Los estados todavía no generados (que, por tanto, no aparecen asociados a ningún nodo de ABIERTOS ni están en la lista CERRADOS). Estos estados podemos considerar que tienen una existencia "potencial", puesto que hasta que no es generado un estado no aparece en ABIERTOS ni forma parte de CERRADOS. En general, muchos estados del espacio de estados no llegarán a ser nunca generados en el proceso de búsqueda.

• Los estados ya generados, pero todavía no examinados. Son aquéllos que aparecen en nodos de ABIERTOS.

• Los estados examinados, pero no expandidos. En realidad se trata de una clase con, a lo más, un elemento que es el estado que aparece ligado al nodo valor de la variable N.

• Los estados expandidos, que son aquellos que constituyen la lista CERRADOS.

Es importante hacer notar que la noción del árbol de búsqueda no varía (sigue

tratándose de un árbol), aunque sí cambia su relación con el espacio de estados. En este nuevo algoritmo tenemos que una adyacencia del espacio de estados que sea examinada puede dar lugar a lo más a una adyacencia en el árbol de búsqueda, pero también puede no dar lugar a ninguna (en el caso en que el espacio de estados no sea un árbol). En el algoritmo de búsqueda en árbol, cada adyacencia examinada del espacio de estados daba lugar al menos a una adyacencia en el árbol de búsqueda, pero podía dar lugar a varias (en el caso en que el espacio de estados no sea un árbol).

Una última observación respecto a este algoritmo se refiere a que, contrariamente a lo que sucedía en la anterior versión, la igualdad entre estados pasa a ser muy importante, puesto que es necesario realizar tareas de localización de estados en listas.

## Búsqueda en profundidad

La estrategia de búsqueda en profundidad es una variante del esquema genérico de búsqueda. Comentamos en primer lugar la versión del punto 3.1, que hemos llamado búsqueda en árbol (después comentaremos la búsqueda en profundidad en grafo, que corresponde al esquema presentado en 3.5).

En este tipo de búsqueda, el paso 7 del algoritmo se particulariza del siguiente

modo: los nuevos nodos son añadidos *al comienzo* de la lista ABIERTOS. Así se consigue que los nodos en ABIERTOS estén ordenados según su profundidad, en orden creciente: los más profundos al principio, los menos profundos al final. La lista ABIERTOS tiene de este modo una forma de acceso que la convierte en una *pila*: los datos que primero entran (paso 7) son los últimos en salir (paso 3). Al igual que sucedía en la búsqueda en anchura, los nodos de igual profundidad se ordenan arbitrariamente, según el orden de las reglas u operadores. Veremos más adelante que la búsqueda en profundidad es mucho más sensible a ese ordenamiento de las reglas que la búsqueda en anchura.