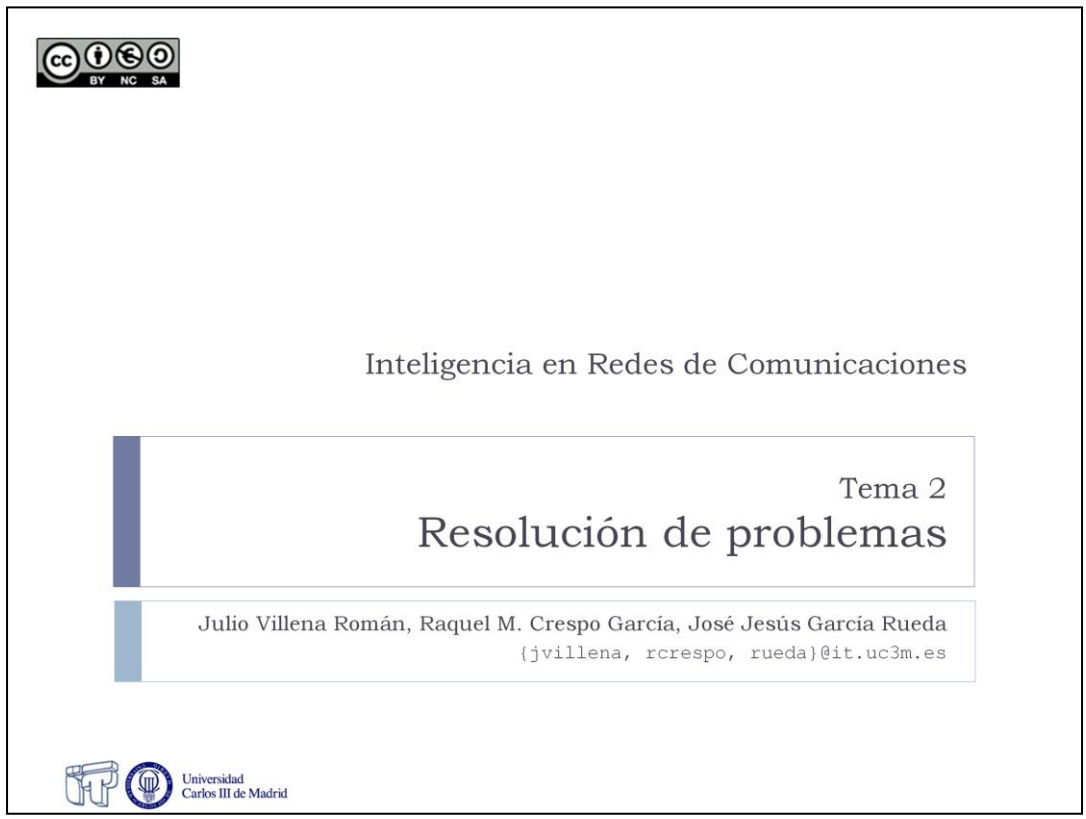


Bibliografía recomendada:

"Inteligencia Artificial: Resolución de problemas, algoritmos de búsqueda", Javier Béjar
http://www.bubok.es/libros/2050/inteligencia-artificial-resolucion-de-problemas-algoritmos-de-busqueda

La resolución de problemas mediante la búsqueda de una solución en el espacio de posibles estados en los que puede encontrarse el problema (por ejemplo, si el problema es ganar en un solitario, las cartas que haya sobre la mesa y su posición en un determinado instante sería un estado del problema) es una de las primeras áreas en las que la Inteligencia Artificial comenzó a trabajar, puesto que su ligación con las matemáticas y su naturaleza más relacionada con el problema en sí mismo que con los mecanismos de la mente la hacían más abordable.

En este tema veremos cómo se plantean formalmente este tipo de problemas, las principales técnicas de resolución por "fuerza bruta", esto es, peinando el espacio de estados hasta dar con la solución y cómo incorporar técnicas heurísticas para reducir los tiempos de búsqueda. Finalmente, se hará una pequeña incursión a la resolución de problemas que involucren a un adversario (juegos).

El problema de resolver problemas

- La resolución de problemas es uno de los procesos básicos de razonamiento que la inteligencia artificial trata de abordar
- El objetivo consiste en lograr que la máquina ayude a un experto humano a encontrar la solución a un determinado problema (de forma más rápida, más exacta, más fiable...)
- Pero...
  - ¿cómo expresar el problema de forma computacional?
  - ¿cómo puede resolverlo la máquina de forma eficiente?

Índice

- Formalización
- Estrategias de búsqueda de soluciones
  - Búsquedas sin información del dominio
  - Búsquedas heurísticas
- Problemas de satisfacción de restricciones
- Juegos

El problema de resolver problemas

- La resolución de problemas es una búsqueda en un espacio de estados, siendo:

  Estado = <*Q*, *R*, *C*>

  Q : estructura de datos que describen al estado

  R : reglas u operaciones que describen las transiciones en el espacio de estados

  C : estrategia de control

  de tal forma que encontrar la solución consiste en encontrar una secuencia de reglas $r_1 \ldots r_n$ que conduzcan desde el estado inicial q_o al estado final q_f

Universidad Carlos III de Madrid

IRC 2011/2012 – 4

Pongamos un ejemplo: supongamos que nuestro problema es encontrar la salida de un laberinto. Cada estado del problema vendría definido por:

•Q: Mi posición actual en el laberinto.

•R: Posiciones adyacentes a las que me está permitido ir desde aquí. Dichas posiciones vendrán determinadas por la aplicación de las reglas de cambio de estado del problema: no se puede atravesar paredes, sólo se puede saltar a posiciones adyacentes…

•C: De entre todas las posiciones adyacentes a las que puedo ir desde aquí, cómo elijo a cuál iré finalmente. Por ejemplo, en el caso de los laberintos, una estrategia tradicional es avanzar siempre pegado a la derecha, retrocediendo a la intersección anterior más próxima cuando me encuentre en un callejón sin salida (esto se corresponde, como veremos después, con una estrategia de "búsqueda en profundidad".

El estado inicial será la entrada del laberinto, y el estado final la salida del mismo.

Un posible solución podría ser: "derecha, arriba, arriba, derecha, abajo, derecha, abajo, izquierda, izquierda". Esto es, la secuencia de movimientos a realizar para salir del laberinto.

Definición formal de un problema

Pasos:

- Definir un espacio (conjunto) de estados
- Especificar uno o más estados iniciales
- Especificar uno o más estados finales (meta/objetivo)
- Definir reglas sobre las acciones disponibles

(abstracción del mundo real a un modelo simbólico)

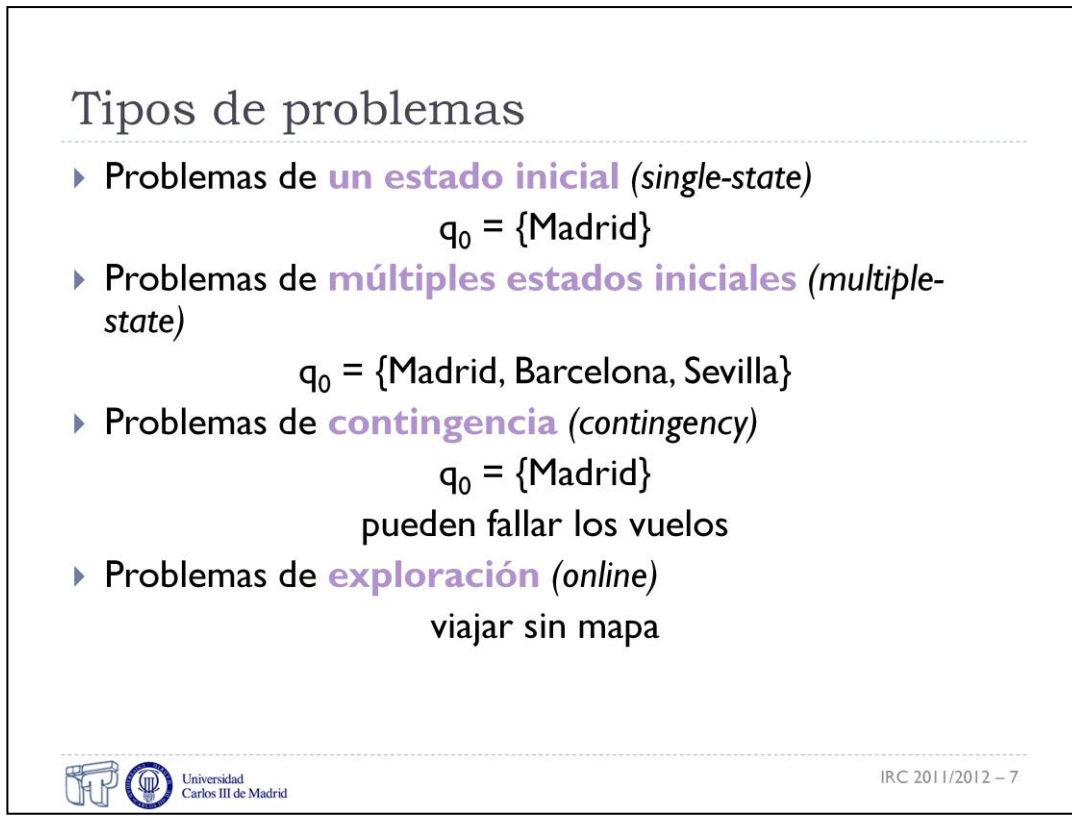
- El problema se resuelve usando las reglas en combinación con una estrategia de control
- La estrategia de control establece el orden de aplicación de las reglas y resuelve los conflictos

El problema del viajante de comercio

Universidad Carlos III de Madrid

IRC 2011/2012 – 6

http://es.wikipedia.org/wiki/Problema_del_viajante



El marco para la definición de problemas que se acaba de plantear da cabida un gran número de problemas de naturalezas muy diferentes.

Así, se puede pensar en problemas en los que el estado inicial es único (como la salida del laberinto, suponiendo que la búsqueda siempre comienza desde la entrada), o varios estados iniciales (típico problema que resuelven los dispositivos GPS, a los que se les puede indicar cualquier lugar como origen del trayecto: puedo preguntarle cómo ir de Madrid a Barcelona, pero también cómo ir de Sevilla a Barcelona. El problema es básicamente el mismo, pero comienzo la búsqueda en un estado diferente).

Igualmente, a los problemas se les pueden añadir condiciones de contingencia. Algo como: "índicame cómo ir de Madrid a Barcelona pero suponiendo que no pasamos por ninguna carretera de peaje"; "no pasar por ninguna carretera de peaje" supone una condición adicional en este caso, que obliga al sistema a buscar una solución alternativa.

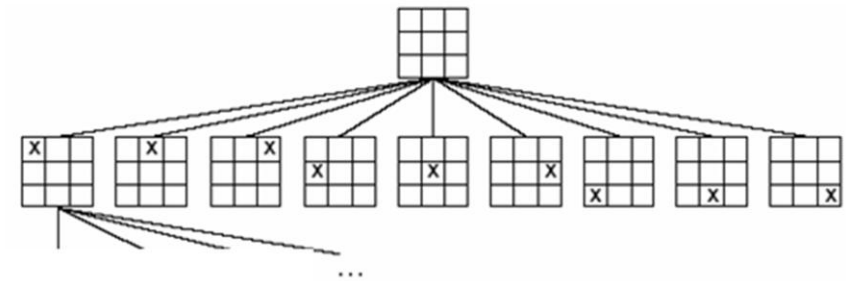
¿Y si el problema a resolver es encontrar un lugar que no sé dónde está? Aquí el planteamiento es diferente: si antes el GPS sabía dónde está Barcelona, y el problema consistía en encontrar un camino hasta allí, ahora me piden que parta desde por ejemplo Madrid y que empiece a recorrer carreteras hasta encontrarme con Barcelona. Éste es un ejemplo de problema de exploración.

Soporte computacional: Definiciones

- Un grafo es una estructura de información compuesta de **nodos** (piezas de información) + **arcos** (uniones entre ellos)
  - **Hojas**: nodos sin descendientes (los últimos)
  - **Camino**: sucesión de nodos siguiendo los arcos
  - **Ciclo**: camino cerrado (bucle)
  - **Grafo dirigido**: los arcos indican el sentido de la relación
  - **Grafo acíclico**: no tiene ciclos
  - **Grafo conexo**: entre dos nodos siempre hay un camino
- Un **árbol** es un grafo dirigido acíclico conexo en el que:
  - Hay un **único nodo raíz**
  - Cada nodo tiene un **único padre**
  - Para cada nodo existe un **único camino** que lo conecta con el nodo raíz
  - **Coste de un nodo**: coste de llegar al nodo desde la raíz a lo largo del mejor camino

Soporte computacional: Árboles

- Para modelar los problemas de búsqueda se usan árboles, en los que:
  - **nodos**: estados intermedios
  - **arco**: aplicación de un operador (movimientos válidos) a un estado



Universidad Carlos III de Madrid

IRC 2011/2012 – 9

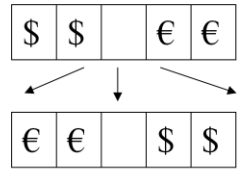
Lo que se modela mediante un árbol es, entonces, el espacio de estados del problema: en la raíz del árbol situaremos el estado inicial, y en las hojas del árbol se encontrarán los diferente estados finales (incluidos los estados solución). En las ramas estarán los estados intermedios por los que el problema puede pasar para llegar desde el estado inicial hasta el estado final.

Las transiciones entre estados se producen cuando se aplican sobre cualquier nodo las reglas de cambio de estado definidas en el problema. Estas reglas de cambio de estado suelen expresarse en forma de operadores. En el caso, por ejemplo, de que estuviésemos tratando de modelar el problema de resolver un determinado juego, las reglas de cambio de estado serían los movimientos válidos que el jugador puede realizar en cada momento.

El problema de "euros y dólares"

Situación inicial:

Objetivo:



- Estado: serie de monedas
- Reglas: r_1: desplazar $ al hueco de la derecha
  r_2: desplazar € al hueco de la izquierda
  r_3: saltar $ a la derecha
  r_4: saltar € a la izquierda
- Card(Q) = 27 (→sirve una búsqueda exhaustiva)

Universidad Carlos III de Madrid

IRC 2011/2012 – 10

En este juego se trata de invertir la posición de los símbolos de euro y los de dólar en una cuadrícula lineal de cinco posiciones. Los movimientos posibles son:

•Desplazar un euro a la izquierda (si el hueco adyacente está libre).

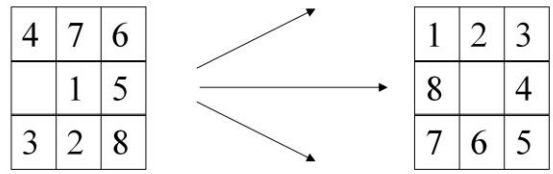
•Desplazar un dólar a la derecha (si el hueco adyacente está libre).

•Que un euro salte a un dólar hacia la izquierda (si el hueco destino está libre).

•Que un dólar salte a un euro hacia la derecha (si el hueco destino está libre).

El conjunto de todas las combinaciones de posibles posiciones de euros y dólares en la cuadrícula tiene un tamaño bastante discreto (27 posibles combinaciones, esto es, 27 posibles estados del problema), por lo que se puede pensar en aplicar técnicas de búsqueda exhaustiva: recorrer el espacio de estados en su totalidad hasta encontrar el estado solución.

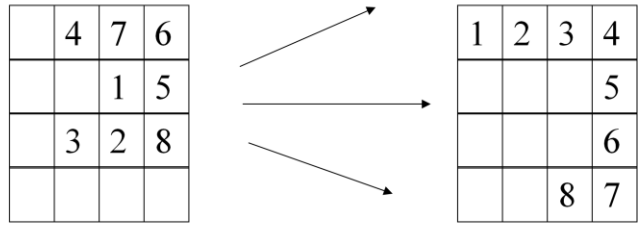
El problema del "puzzle-8"



- Estado: matriz
- Reglas: r_1: hueco a la derecha
  r_2: hueco a la izquierda
  r_3: hueco arriba
  r_4: hueco abajo
- Card(Q) = 9! = 362.880 (→sirve una búsqueda exhaustiva)

http://jc-info.blogspot.com/2009/04/problema-del-8-puzzle-y-busqueda-no.html

El problema del "puzzle-15"

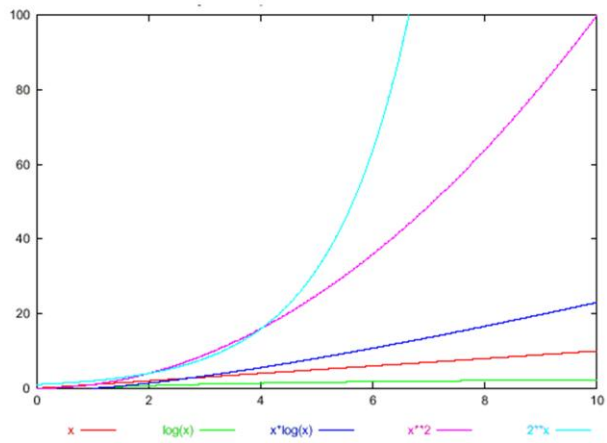


- Card(Q) = 16! ~ $2 \cdot 10^{13}$

  → ¡¡¡ya no sirve una búsqueda exhaustiva!!!

http://en.wikipedia.org/wiki/Fifteen_puzzle

Complejidad



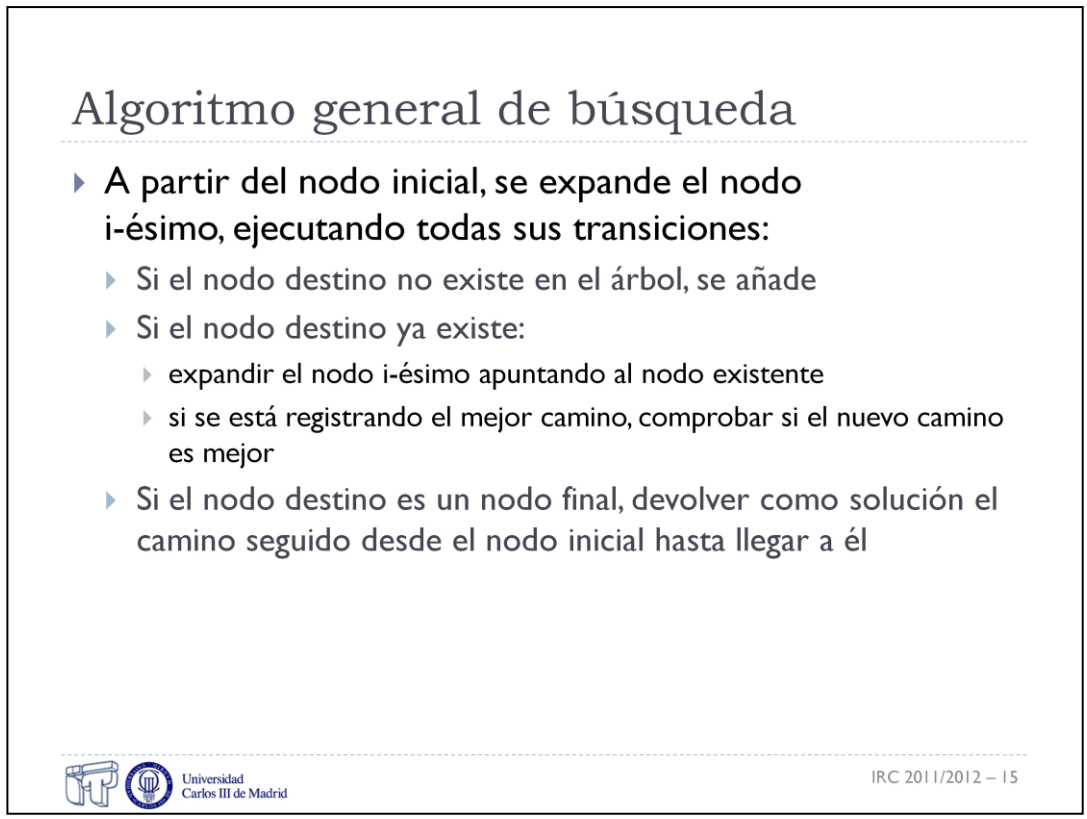
- La resolución de problemas en IA es un problema NP-completo (No-Determinista Polinómico, con crecimiento asintótico)

La explosión combinatoria de los espacios de estados a poco que los problemas a resolver tengan cierto tamaño es uno de los grandes obstáculos en IA. Al menos si se trata de abordar la resolución de dichos problemas mediante técnicas "de fuerza bruta", esto es, que supongan peinar el espacio de estados a ciegas hasta encontrar un estado solución.

Ante este tipo de problemas son necesarias otras estrategias, que acorten los tiempos de búsqueda tratando de aplicar información adicional que nos permita seleccionar unos caminos sobre otros en el árbol.

Estrategias de búsqueda

En definitiva, no sólo basta plantear los problemas de una manera formal, sino que hace falta encontrar la forma apropiada de decidir las reglas a aplicar desde el estado inicial para llegar al estado final y el orden en que éstas se aplican: las **estrategias de búsqueda**



Todas las estrategias de búsqueda que vamos a ver aquí caben dentro de lo que se denomina "el algoritmo general de búsqueda", que refleja las partes del proceso de búsqueda que son comunes a todas las estrategias.

Básicamente, se trata de ir descubriendo nuevos nodos a partir de los ya existentes (descubrir los nodos a los que me da acceso el nodo actual se denomina "expandir" dicho nodo"), almacenando los nuevos nodos descubiertos a la espera de que les llegue el turno de ser expandidos a su vez. Si se descubre un nodo que ya estaba descubierto previamente, habrá que comprobar si el camino seguido actualmente para llegar hasta dicho nodo ya expandido es más o menos costoso que el camino a través del cual descubrí el nodo por primera vez.

Si en algún momento se descubre un nodo que es solución del problema, el algoritmo termina y devuelve el camino menos costoso seguido entre el nodo inicial y el nodo solución.

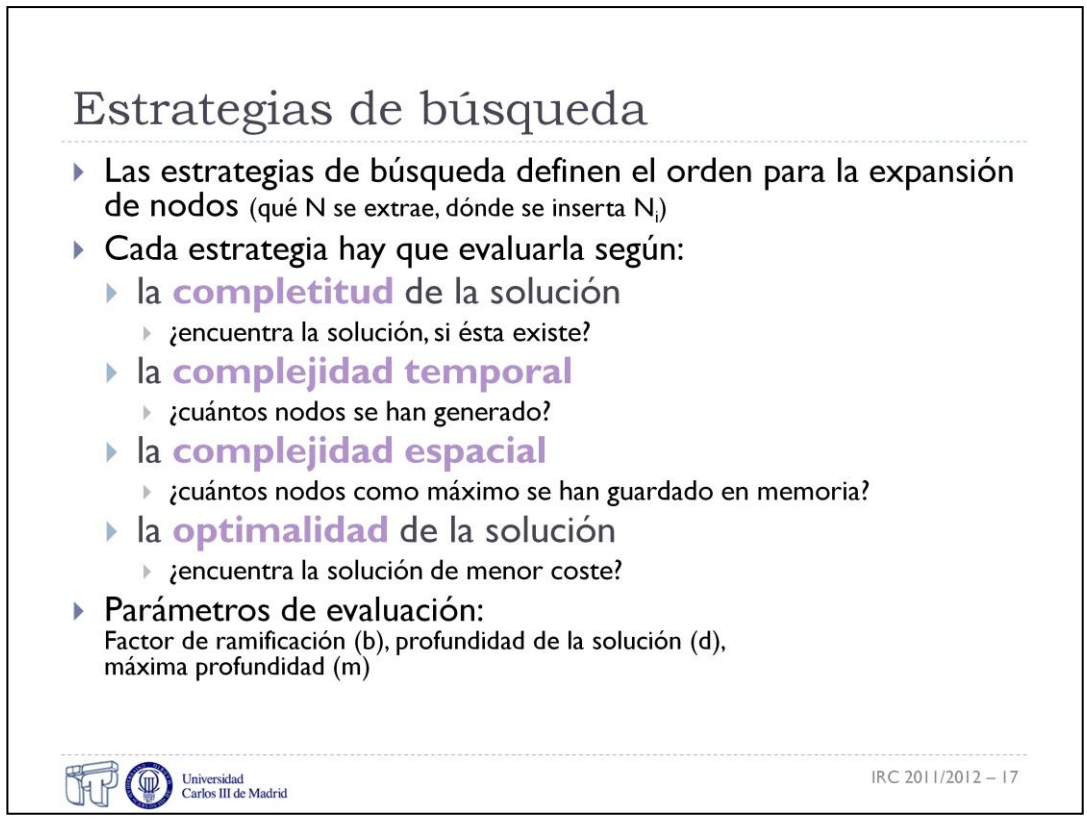
Algoritmo general de búsqueda (2)

Q,V: lista de nodos
S: nodo inicial

1) Inicializar Q = {S},V = {S}, $coste_S$=0
2) Si Q está vacía, devolver SIN_SOLUCIÓN
3) Sacar un nodo N de Q
4) Si N es nodo final, devolver N como solución
5) Para todos los descendientes N_i de N:
   - Enlazar N_i con N
   - Si N_i no está en V:
     $coste_{Ni}$=$coste_N$+$coste_{N\rightarrow Ni}$
     añadir N_i a Q
     añadir N_i a V
   - En otro caso:
     Si $coste_{Ni}$>$coste_N$+$coste_{N\rightarrow Ni}$
     $coste_{Ni}$=$coste_N$+$coste_{N\rightarrow Ni}$
     añadir N_i a Q
     En otro caso, no hay que hacer nada
6) Volver a 2

IRC 2011/2012 – 16

Ver también el apartado 1.3 del libro de Béjar.



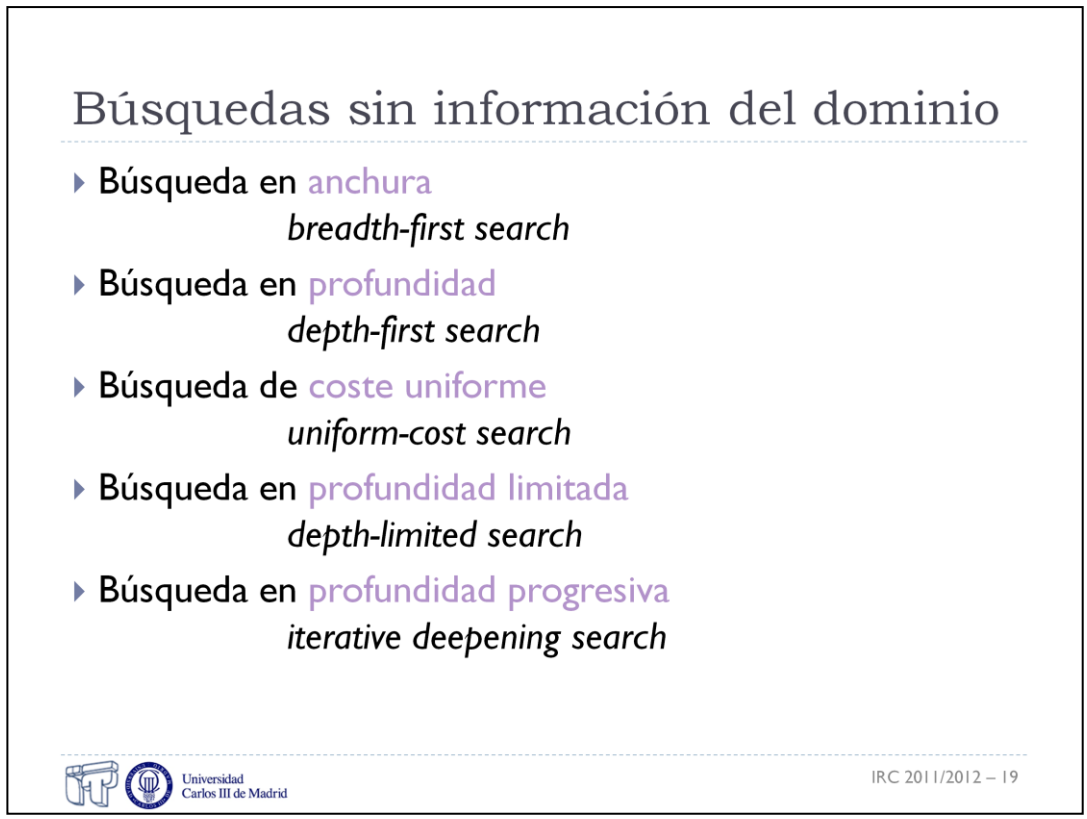
Lo que diferencia una estrategia de búsqueda de otra es el orden en que recorren los nodos ya descubiertos para expandirlos: por ejemplo, una estrategia puede decidir expandir primero los más recientemente descubiertos, mientras otra puede preferir expandir los nodos en el mismo orden en que se descubrieron.

Este aparente detalle en el algoritmo general de búsqueda genera estrategias muy diferentes, con propiedades totalmente distintas: unas pueden ser completas (capaces de encontrar la solución) mientras que otras pueden no encontrarla; unas pueden tener que recorrer un mayor número de nodos que otras antes de encontrar la solución; unas tendrán en todo momento una larga cola de nodos descubiertos esperando para ser procesados, mientras que otras necesitarán guardar tan sólo unos cuantos; por último, algunas serán capaces no sólo de encontrar la solución, sino de proporcionar también el camino de menor coste hasta ella.

Los parámetros de evaluación a los que se refiere la transparencia tienen que ver con la naturaleza del problema, no con la estrategia seleccionada. Sin embargo, de estos parámetros, esto es, de la naturaleza del problema, dependerá lo buena o mala que sea una estrategia para un problema dado.

Tipos de estrategias de búsqueda

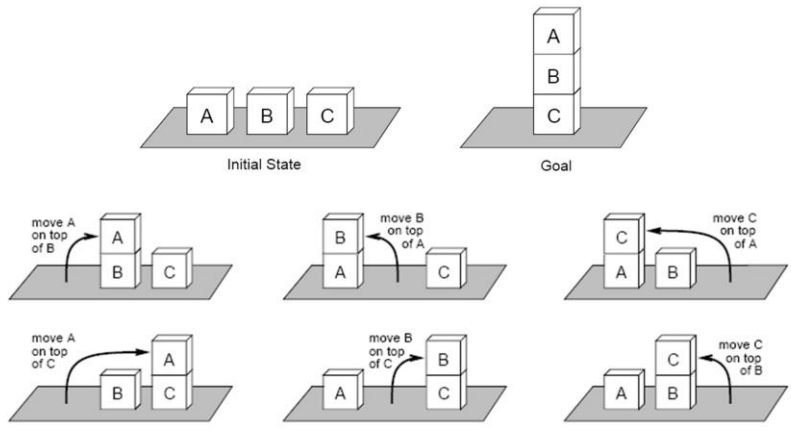
- Estrategias sin información del dominio o búsqueda a ciegas *(uninformed strategies)*
  - Sólo emplean la información en la definición del problema
  - "Fuerza bruta"
- Estrategias con información del dominio o estrategias heurísticas *(informed strategies)*
  - Emplean información del espacio de búsqueda para evaluar cómo va el proceso
  - La idea es utilizar una función de evaluación (heurístico) de cada nodo (del coste de llegar a él)



Éstos son los tipo de búsqueda a ciegas o "por fuerza bruta" que vamos a estudiar aquí.

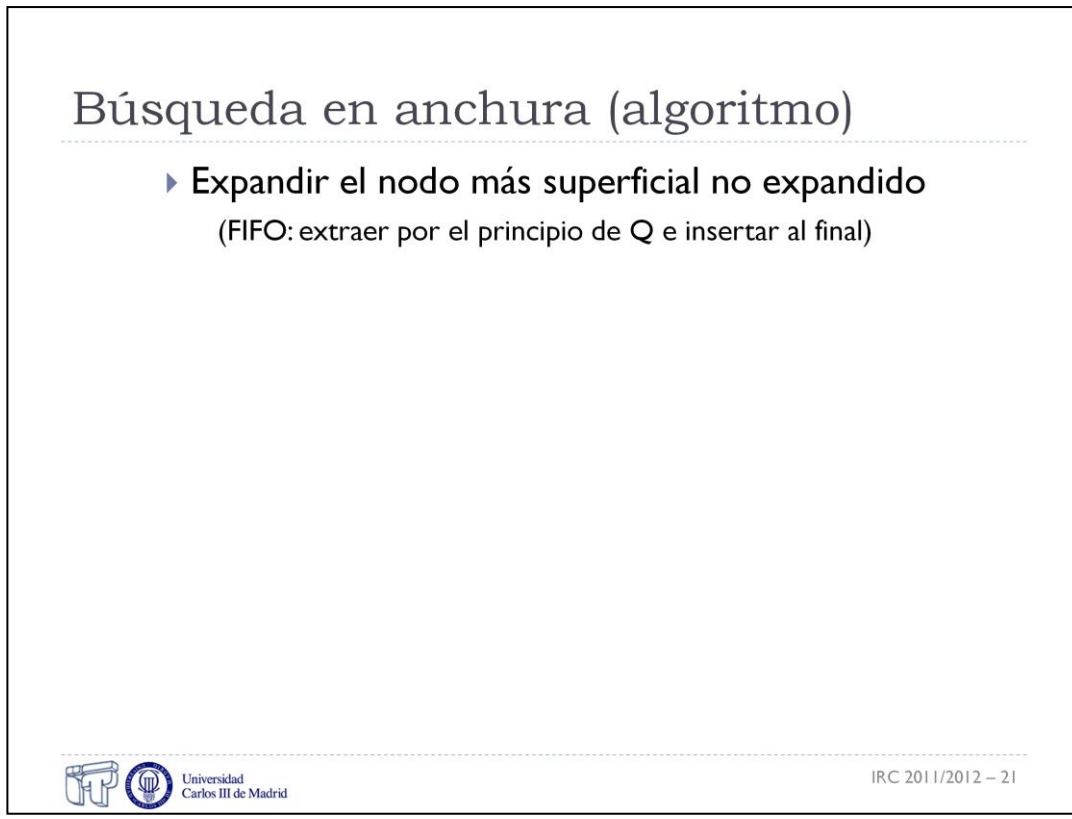
Las dos estrategias principales son la búsqueda en anchura y la búsqueda en profundidad, complementadas con la búsqueda de coste uniforme si es que estamos considerando costes diferentes para cada transición de un nodo a otro. Prácticamente todo el resto de estrategias de búsqueda serán variaciones de estas tres.

El mundo de los bloques

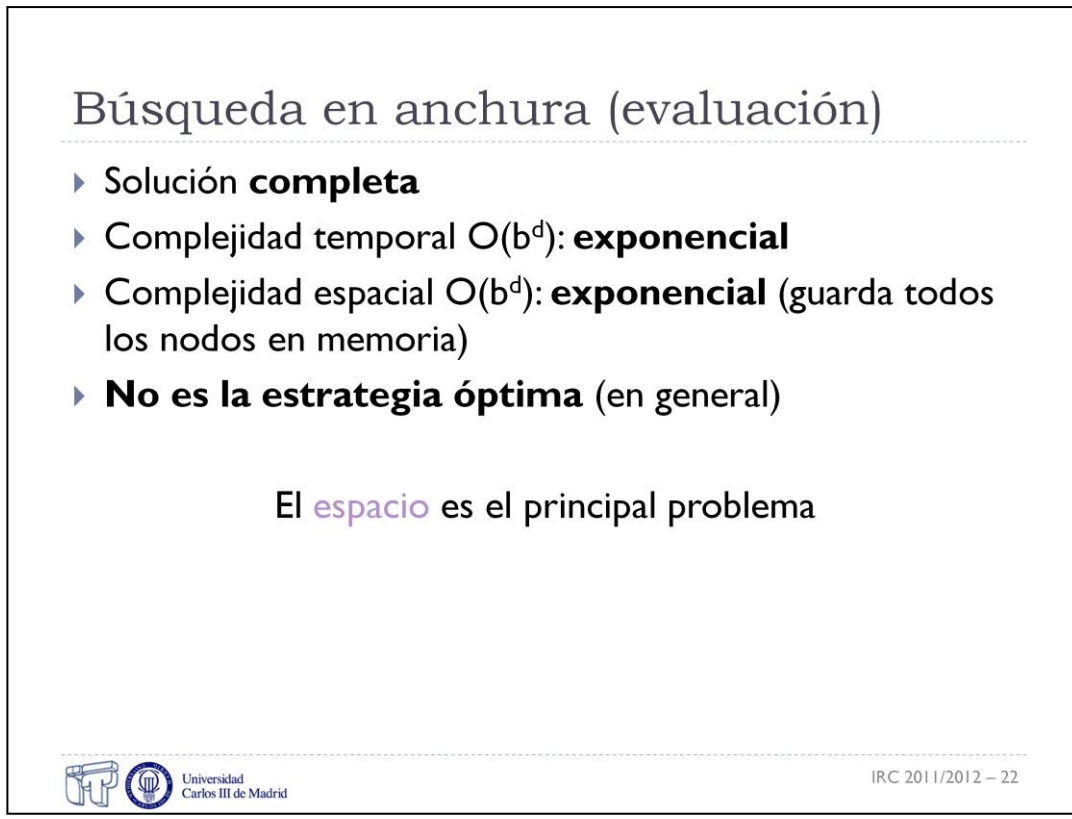


- SOAR *(a general cognitive architecture for developing systems that exhibit intelligent behavior)*

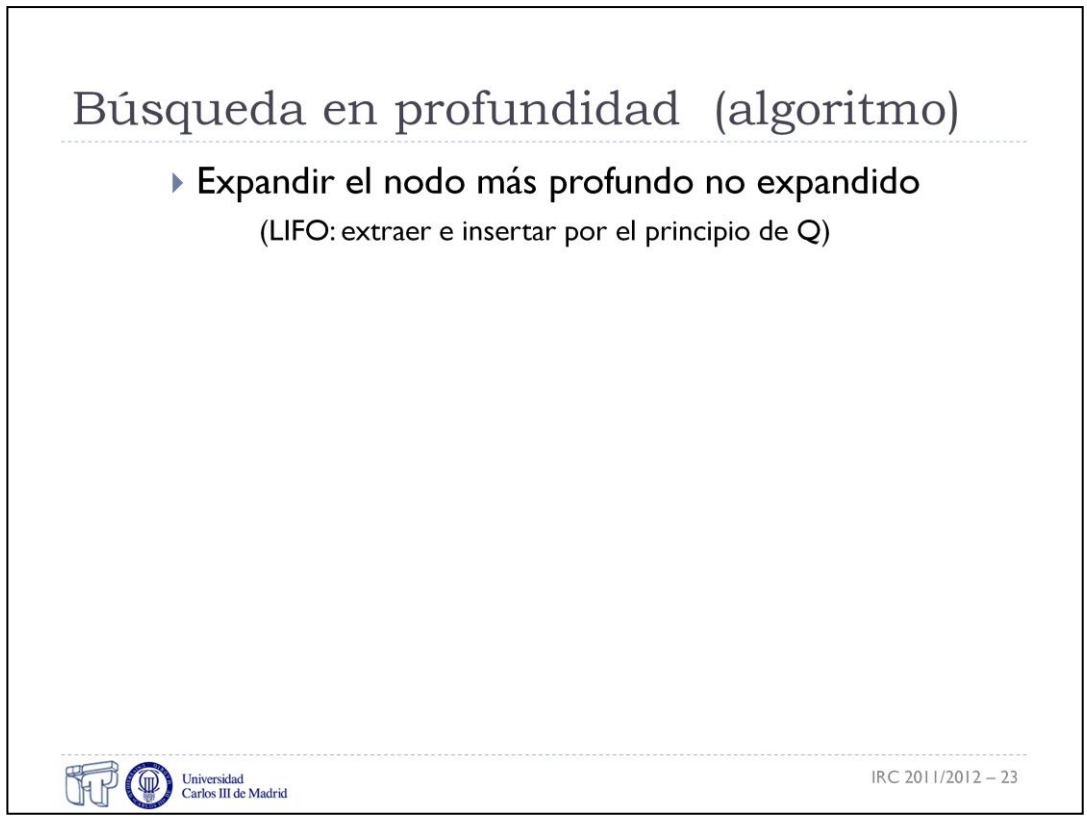
  http://sitemaker.umich.edu/soar/home



Ver apartado 2.2 del libro de Béjar.



El principal problema de la búsqueda en anchura es su coste en recursos, sobre todo en lo concerniente al espacio de almacenamiento en memoria que requiere: al ir expandiendo los nodos en el orden en que son descubiertos, se necesita tener una cola de nodos en memoria que irá creciendo exponencialmente con cada nuevo nodo expandido (saco el nodo expandido de la cola, pero en su lugar introduzco todos los nuevos nodos descubiertos a partir de él, que esperarán allí un largo tiempo antes de que les llegue a su vez el turno de ser expandidos. Mientras tanto, deberé mantenerlos en la cola, y estarán ocupando espacio en memoria).



Ver el apartado 2.3 del libro de Béjar.

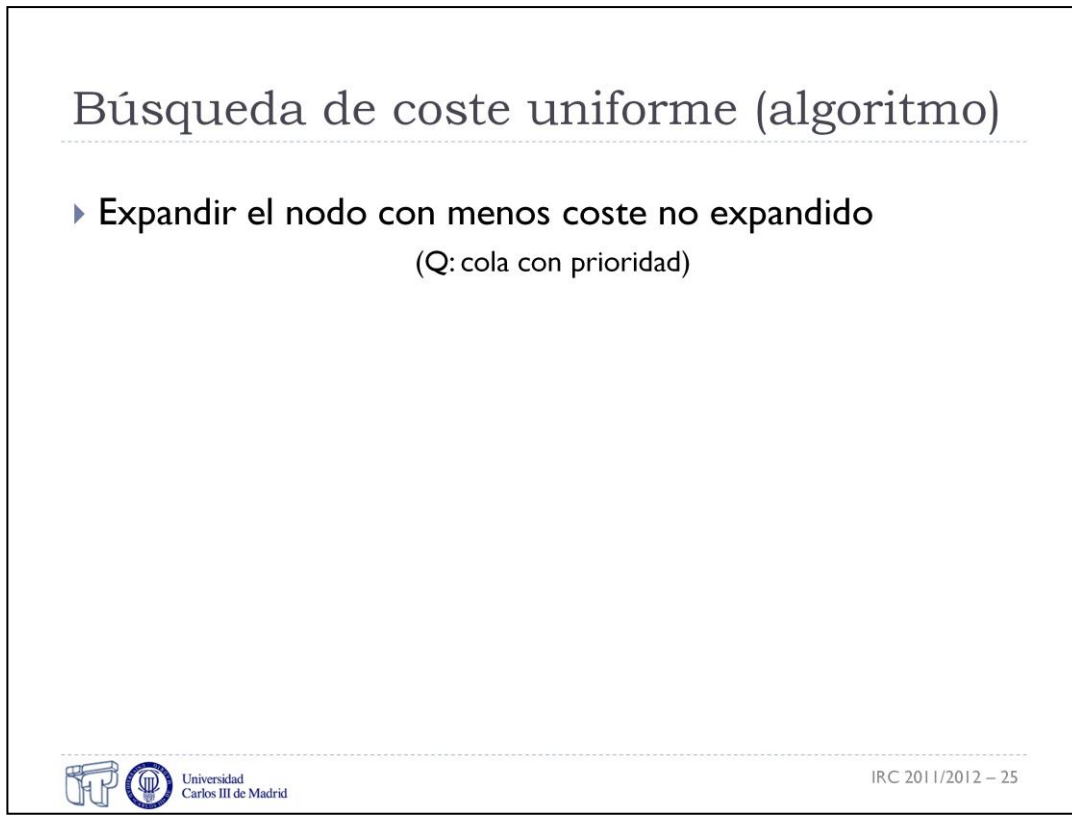
Búsqueda en profundidad (evaluación)

- **Solución no completa**
  - En espacios con ciclos puede haber bucles infinitos
  - Es necesaria una comprobación de estados repetidos
- Complejidad temporal $O(b^m)$:
  - **enorme** si profundidad máxima >> profundidad de la solución
  - Sin embargo, el algoritmo es **rápido** si el espacio de soluciones es denso
- Complejidad espacial O(bm): **lineal**
- **No es la estrategia óptima** (en general)

La búsqueda en profundidad, por su parte, soluciona el problema del almacenamiento en memoria (ahora sólo es necesario tener almacenados en cada momento los nodos de la rama que estemos siguiendo), pero a cambio podría no encontrar nunca la solución (en el caso de que el espacio de estados no fuese un árbol puro, sino un grafo con ciclos, esto es, caminos que vuelven una y otra vez a un mismo nodo ya expandido).

Esto obliga muchas veces a llevar un control de los nodos ya visitados, con lo que la ventaja espacial empieza a no ser tanta: hay que almacenar los nodos ya visitados.

Por otra parte, la existencia de ramas muy profundas en las que no se da ninguna solución puede suponer alargar mucho el tiempo de búsqueda (en este caso, una búsqueda en anchura daría con la solución más rápidamente).



La búsqueda en profundidad y la búsqueda en anchura suponen que el costo de saltar de un nodo a otro es el mismo sean cuales sean los nodos origen y destino del salto, por lo que el coste total de un camino se mide por el número de saltos, nada más.

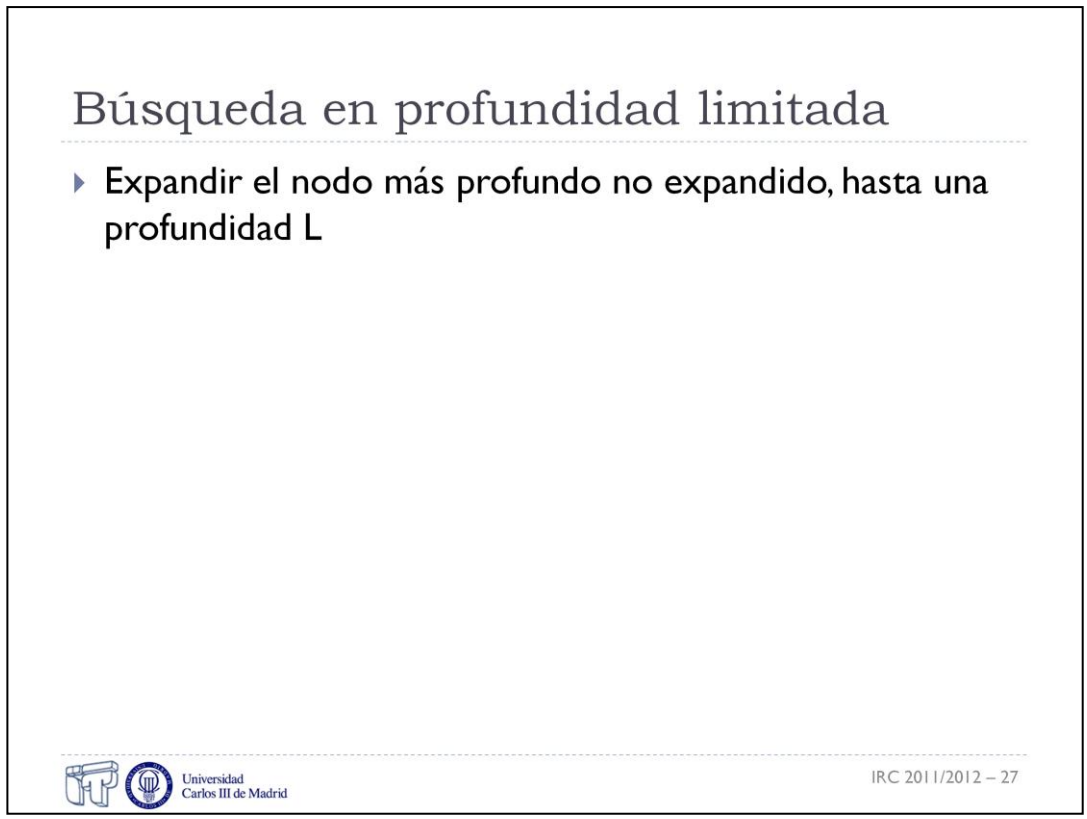
Cuando se introduce un peso a esos saltos, un coste diferente para cada uno de ellos, el número de saltos ya no es la variable a minimizar: hay que minimizar la suma total de los pesos en un determinado camino, esto es, el coste total del camino.

La búsqueda de coste uniforme evalúa el coste acumulado de todos los nodos descubiertos hasta el momento, y continúa la búsqueda por aquel cuyo coste es menor.

Búsqueda de coste uniforme (evaluación)

- Solución **completa**
- Complejidad temporal:
  nº de nodos con coste <= coste de la solución óptima
- Complejidad espacial:
  nº de nodos con coste <= coste de la solución óptima
- **Estrategia óptima**

- Pesos positivos: algoritmo de Dijkstra
- Pesos negativos: algoritmo de Bellman-Ford



De nuevo, ver apartado 2.3 del libro de Béjar.

Búsqueda en profundidad progresiva (algoritmo)

```
FOR L=0 to inf
        Aplicar búsqueda_en_profundidad_limitada(L)
        Si el resultado es válido, se devuelve
NEXT L
```


Ver el apartado 2.4 del libro de Béjar.

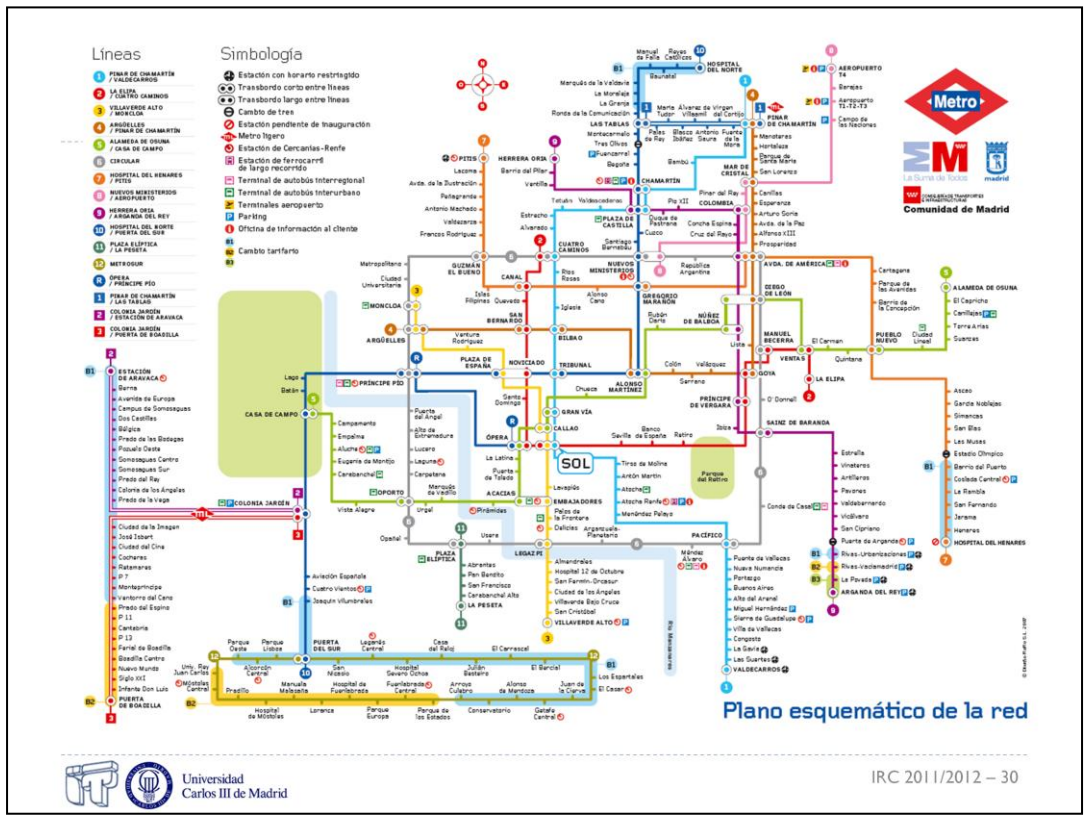
Búsqueda en profundidad progresiva (evaluación)

- Solución **completa**
- Complejidad temporal $O(b^d)$: **exponencial** (similar a otras estrategias)
- Complejidad espacial O(bd): **lineal**
- **Estrategia óptima** si el coste = 1

- Puede modificarse para utilizar búsqueda de coste uniforme

La estrategia de búsqueda en profundidad progresiva es probablemente la mejor de las que hemos visto hasta el momento, puesto que tiene las bondades espaciales de la búsqueda en profundidad pero eliminando los problemas temporales derivados de las ramas muy profundas y sin soluciones. En el caso de búsquedas sin pesos, se demuestra además que esta estrategia es óptima.

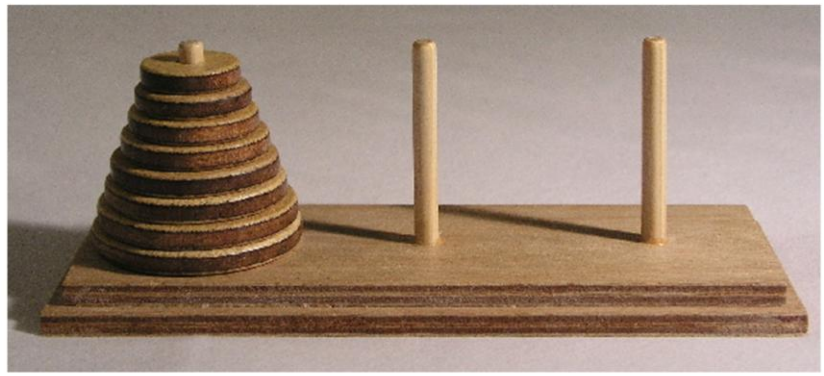
Curiosamente, y aunque la intuición en principio pueda decirnos lo contrario, no hay un incremento sustancial en el tiempo de búsqueda frente a otras estrategias, sobre todo cuando el tamaño del espacio de estados es grande.



¿Qué estrategia de búsqueda elegirías para encontrar el trayecto más corto entre dos estaciones de metro?

(Primer punto a analizar: ¿el coste de los trayectos es uniforme?)

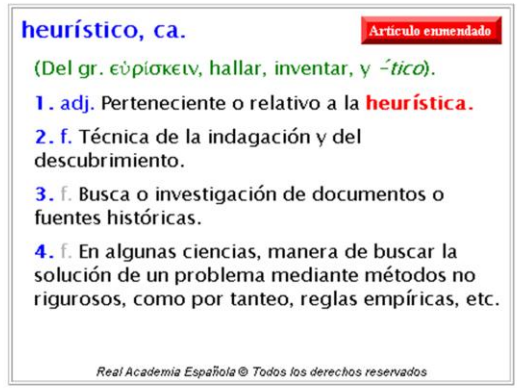
Las Torres de Hanoi



http://www.uterra.com/juegos/torre_hanoi.htm

http://es.wikipedia.org/wiki/Torres_de_Han%C3%B3i

Búsquedas heurísticas



heurístico, ca. Artículo enmendado

(Del gr. εὑρίσκειν, hallar, inventar, y *-tico*).

1. adj. Perteneciente o relativo a la **heurística.**

2. f. Técnica de la indagación y del descubrimiento.

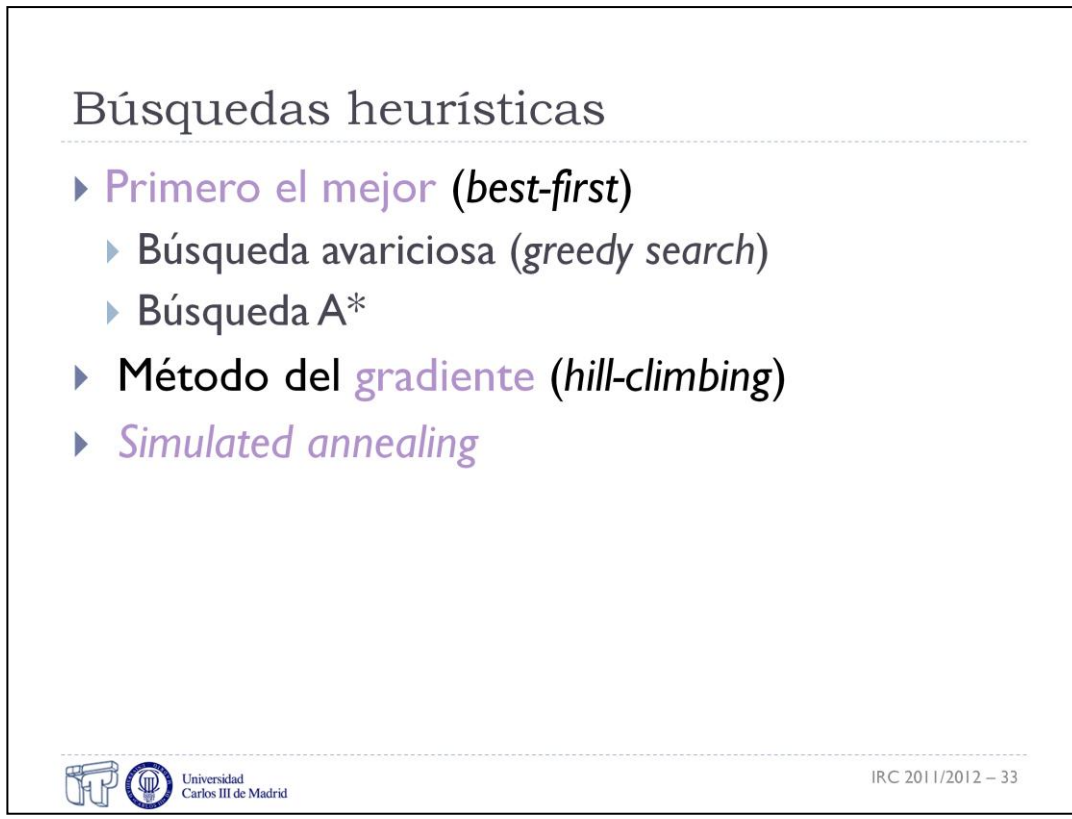
3. f. Busca o investigación de documentos o fuentes históricas.

4. f. En algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.

Real Academia Española © Todos los derechos reservados

- Se dice de aquel truco o regla empírica que ayuda a encontrar la solución de un problema (pero que no garantiza que se encuentre)

Ver el capítulo 3 del libro de Béjar.



La aplicación de técnicas heurísticas requiere de cierta capacidad para la predicción: típicamente, no sólo me voy a fijar en lo que ha pasado hasta el momento durante el proceso de búsqueda (esto es, cuál es el coste de llegar al estado actual) sino que voy a tratar de predecir, lo mejor posible, lo que pasará de aquí hasta el nodo solución.

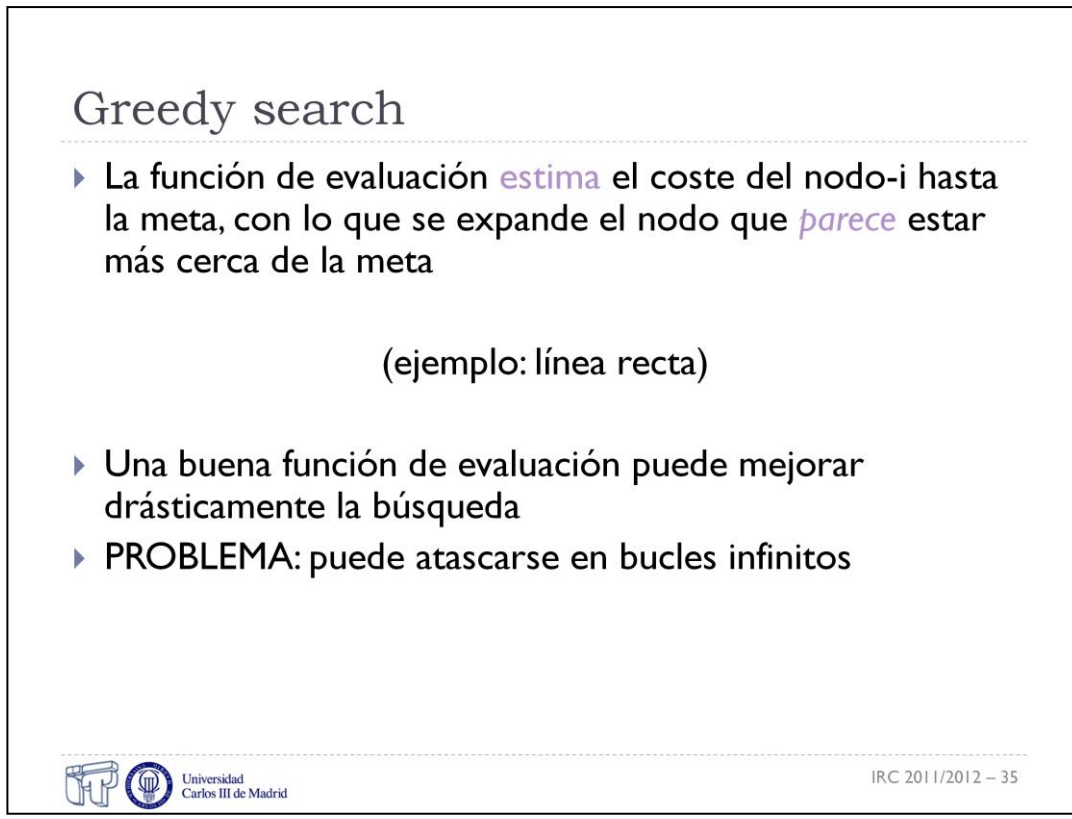
La idea es elegir, de todos los caminos que se me abren, de todos los nodos que podría expandir a continuación, aquel que parezca que me va a llevar más rápidamente hacia un estado solución.

Jugando con esta idea de predicción y con su implementación, obtenemos diferentes estrategias de búsqueda.

Primero el mejor

- Se utiliza una función de evaluación (heurística) para cada nodo y se expande el nodo mejor evaluado no expandido

  (misma idea que en la búsqueda de coste uniforme)
- Tema complejo y muy abierto a la "idea feliz"…
- Casos especiales:
  - búsqueda avariciosa (*greedy search*)
  - búsqueda A*



En toda estrategia que emplea la predicción como parte de la decisión de por dónde continuar la búsqueda, dicha predicción va a tomar típicamente la forma de una función de evaluación, que tratará de dar un valor numérico a la predicción.

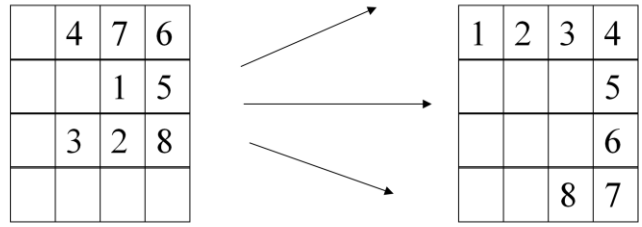
Si hasta el momento en este tema todo ha sido muy determinista, muy "así son las cosas" y así se aplican, al llegar a este punto aparece el talento y la experiencia del desarrollador del algoritmo de búsqueda: no hay reglas para dar con una buena función de evaluación. De ahí que se diga que estamos empleando la heurística: ¡la del programador, por supuesto! Éste deberá intuir, probar y tantear, hasta encontrar una función de evaluación que parezca comportarse lo suficientemente bien (esto es, que acelere el encuentro de la solución, disminuyendo en lo posible el espacio necesario para trabajar).

La pregunta a hacerse sería, por ejemplo: ¿cómo estimar matemáticamente la cercanía de un nodo actual a un hipotético nodo final que no sé dónde está?

Búsqueda A*

- La idea es evitar expandir caminos que ya son muy costosos
- Función de evaluación: f(n) = g(n) + h(n)
  - g(n): coste sufrido hasta alcanzar n
  - h(n): coste estimado desde n hasta la meta
  - f(n): coste estimado total hasta la meta pasando por n
- Se demuestra que la búsqueda A* alcanza la solución óptima, siempre que se utilice un heurístico "admisible" (que **no sobreestime** el coste real):
  - h(n) <= h*(n) (el coste real)

Ejemplos de heurísticos

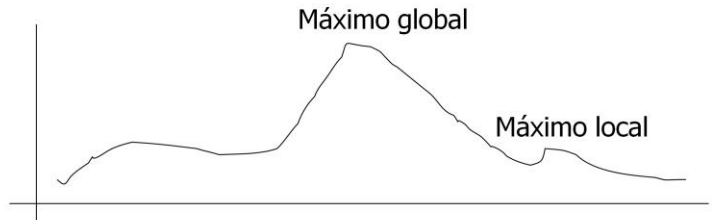


- Por ejemplo, se podría usar:
  - número de cuadros fuera de su sitio
  - distancia Manhattan (número de cuadros desde el sitio correcto de cada cuadro)

http://en.wikipedia.org/wiki/Fifteen_puzzle

Hill-climbing

- Trata de ir decidiendo el camino con menor coste hasta la meta
- Para ello se queda en cada salto con el nodo destino mejor valorado y sigue expandiendo por él
- PROBLEMA: puede atascarse en **máximos locales**, según el estado inicial



Ver apartado 4.2 del libro de Béjar.

Simulated annealing

- La idea es escapar de los máximos locales permitiendo movimientos "incorrectos", pero reduciendo gradualmente su tamaño y frecuencia
- Se utiliza como parámetro la ***temperatura*** T del proceso
- Si la temperatura se reduce suficientemente despacio, se alcanza la solución óptima
- Fue desarrollado en 1953 para modelado de procesos físicos

Ver apartado 4.3 del libro de Béjar.

Problemas de satisfacción de restricciones

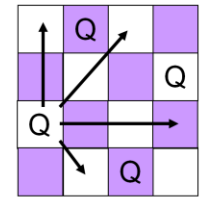
Constraint satisfaction problem (CSP)

- **Objetivo**: descubrir un estado del problema que satisfaga un conjunto de restricciones
- Los CSPs son problemas especiales en los que:
  - los estados están definidos por los valores asignados a un conjunto de variables
  - el objetivo está definido por restricciones en los valores de las variables

Ver capítulo 6 del libro de Béjar.

Ejemplo clásico: 4 reinas

- Poner cuatro reinas (damas) sin que ninguna pueda atacar a las demás
  (→ sin que haya dos en la misma columna y no estén en diagonal)



- Variables: Q_1, Q_2, Q_3, Q_4
- Dominio: {1,2,3,4}
- Restricciones:
  - Q_i != Q_j
  - $|Q_i - Q_j|$!= |i - j|
- Significado: (Q_1, Q_2) pueden valer (1,3) (1,4) (2,4) (3,1) (4,1) (4,2)

Universidad Carlos III de Madrid

IRC 2011/2012 – 41

Las restricciones, traducidas a lenguaje vulgar, serían en este caso:

•Que cada reina esté en una columna diferente.

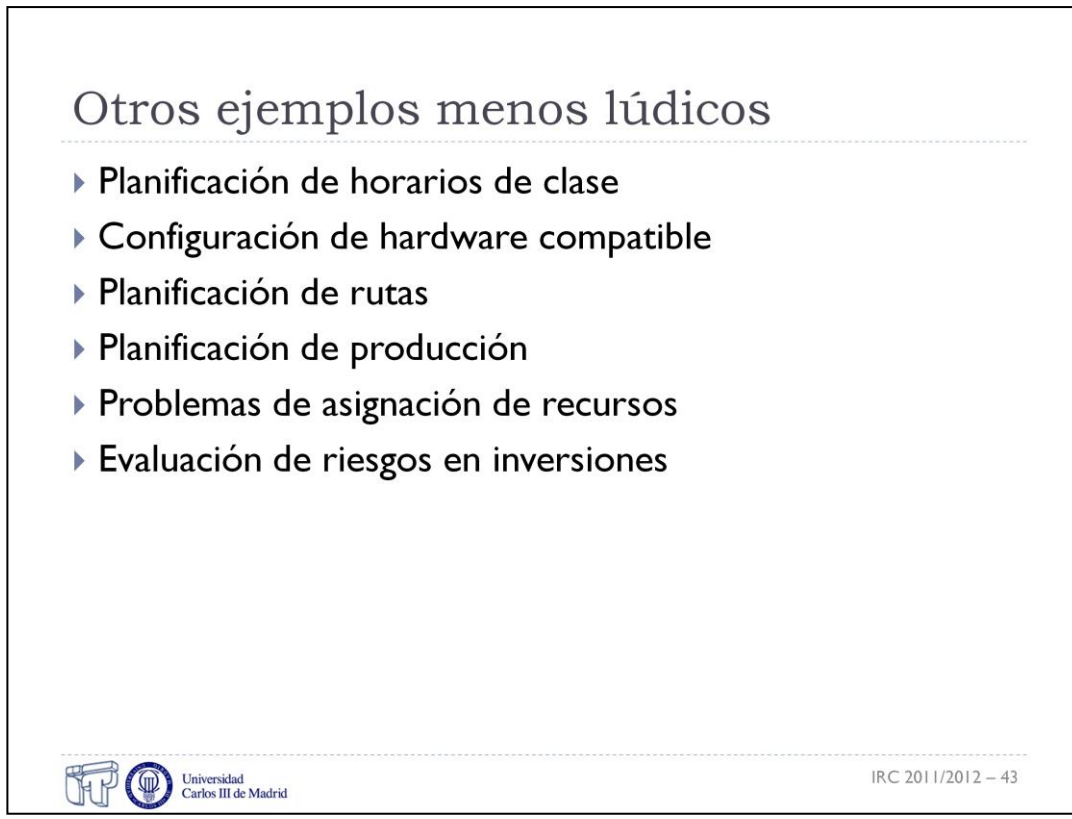
•Que no haya dos reinas en la misma diagonal.

Podría haberse incluido en el conjunto de las restricciones el que cada reina esté en una fila diferente, pero eso, en este caso, se ha "dado por sabido": no serán estados de nuestro problema aquellos en los que hubiera más de una reina en una misma fila.

Ejemplo: Sudoku

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

http://sudoku.3ontech.com



Los problemas de satisfacción de restricciones son de lo más común en la vida diaria: ¿cómo repartir a los invitados a un banquete por las mesas sin sentar juntos a ninguno que se lleve mal? ¿Cómo escoger el lugar en el que pasar las vacaciones, teniendo en cuenta los gustos de toda la familia? ¿Cómo repartir tareas entre los componentes de mi grupo, para que a cada uno le toque algo que le guste y sepa hacer bien? Todos ellos, a poco que nos pongamos, pueden ser expresados como problemas de satisfacción de restricciones.

En general, como vemos, suelen ser problemas que involucren un reparto de recursos, estando los grados de libertad de dicho reparto limitados: "no se te ocurra sentar al tío Alberto junto a la tía Emilia, que se llevan a matar".

Podemos imaginarnos un espacio de estados inicialmente enorme (las posibilidades de sentar a la gente en el banquete son casi infinitas), al que le vamos recortando pedazos con cada nueva restricción que introducimos (todos los estados que involucren que el tío Alberto y la tía Emilia están sentados a la misma mesa, son inmediatamente descartables, esto es, son estados no válidos).

Búsqueda en CSP (enfoque inicial)

- El enfoque inicial es abordar una búsqueda normal
- Estado inicial: variables sin asignar
- Operadores: asignar valores a variables no asignadas
- Objetivo: todas las variables asignadas, todas las restricciones cumplidas
- Estrategia: (por ejemplo) búsqueda en profundidad

- **DESVENTAJAS**: Baja eficiencia porque el orden de asignación es irrelevante y no se comprueban las restricciones no cumplidas

Búsqueda con retroceso *(backtracking)*

- Igual que el anterior, empleando búsqueda en profundidad pero:
  - fijando el orden de asignación de variables
  - comprobando violación de restricciones permitiendo sólo sucesores válidos
- Este es el algoritmo básico sin información del dominio para los CSPs
- Es capaz de resolver hasta 15-reinas

Ver ápartado 6.2.1 del libro de Béjar.

Comprobación hacia delante *(forward checking)*

- Idea:
  - recordar los valores válidos que quedan para las variables no asignadas
  - detener la búsqueda cuando no quedan valores válidos para una variable
- La comprobación hacia delante evita a priori asignaciones erróneas
- Es capaz de resolver hasta 30-reinas

http://www.animatedrecursion.com/advanced/the_eight_queens_problem.html

Ver apartado 6.2.3 del libro de Béjar.

Búsqueda heurística en CSPs

- Idea: tomar decisiones más "inteligentes" sobre:
  - qué valor asignar a la siguiente variable
  - qué variable asignar la siguiente
- El uso de heurísticos "inteligentes" mejora significativamente el proceso de búsqueda
- Se puede resolver hasta 1000-reinas

... ¿y más de eso?

Como siempre, si a los mecanismos de fuerza bruta le añadimos de alguna manera la intuición y la experiencia previa del desarrollador, es probable que lleguemos a mejores estrategias.

Siguiendo con el ejemplo del banquete, ¿no sería una buena idea asignar asiento primero a los invitados más problemáticos? Eso me ahorraría muchísimos intentos de poblar la mesa con invitados que en último momento dejarían de ser válidos porque, en los pocos sitios que quedan libres hacia el final del proceso, no hay manera de colocar a los invitados díscolos sin que se deje de cumplir alguna de las restricciones. Y vuelta a empezar…

Asignado asiento primero a los más problemáticos, disminuyo la probabilidad de encontrarme con incompatibilidades hacia el final del proceso.

Juegos

- Problemas de búsqueda donde interviene al menos un adversario
  - Tus movimientos por sí solos no aseguran la victoria: es necesaria una estrategia de oposición
  - En general, el tiempo disponible para cada movimiento impone soluciones aproximadas (no sirve la fuerza bruta)
  - Dos tipos de juegos:
    - Deterministas: no interviene el azar (4-en-raya, ajedrez, damas)
    - No deterministas: el azar está presente (backgammon, parchís, monopoly…)
- ¿Por qué gustan tanto en IA?
  - Divertidos
  - Difíciles
  - Fáciles de formalizar y con un número pequeño de acciones

(Los juegos son para la IA como la F1 es para la ingeniería del automóvil)

IRC 2011/2012 – 48

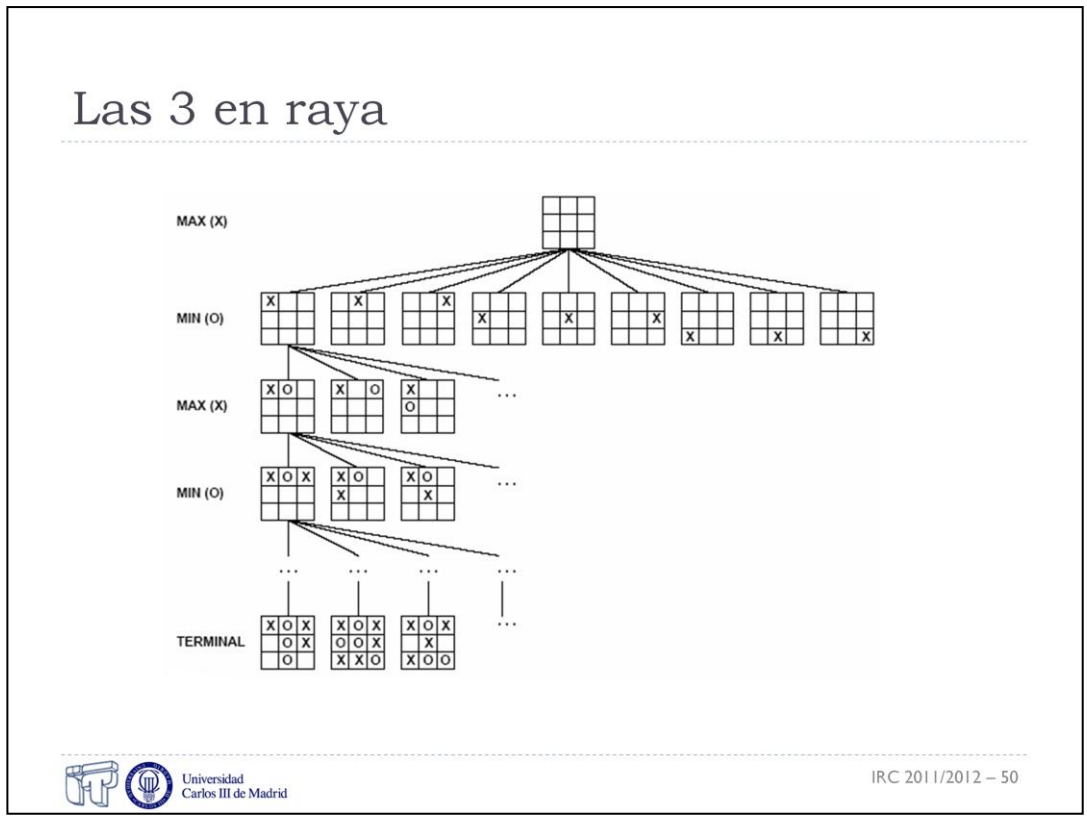
Ver capítulo 5 del libro de Béjar.

Estrategia Minimax

- Consiste en elegir el mejor movimiento para uno mismo (MAX) suponiendo que el adversario (MIN) escogerá el mejor para sí mismo **(que también juega a ganar)**
- Pasos:
  - Generar el árbol de juego, alternando movimientos *(ply)* de MAX y MIN y asignándoles los valores apropiados (MAX>0, MIN<0)
  - Calcular la función de utilidad de cada nodo final, recorriendo recursivamente los nodos hasta el estado inicial
  - Elegir como jugada a realizar aquel primer movimiento que conduce al nodo final con mayor función de utilidad

IRC 2011/2012 – 49

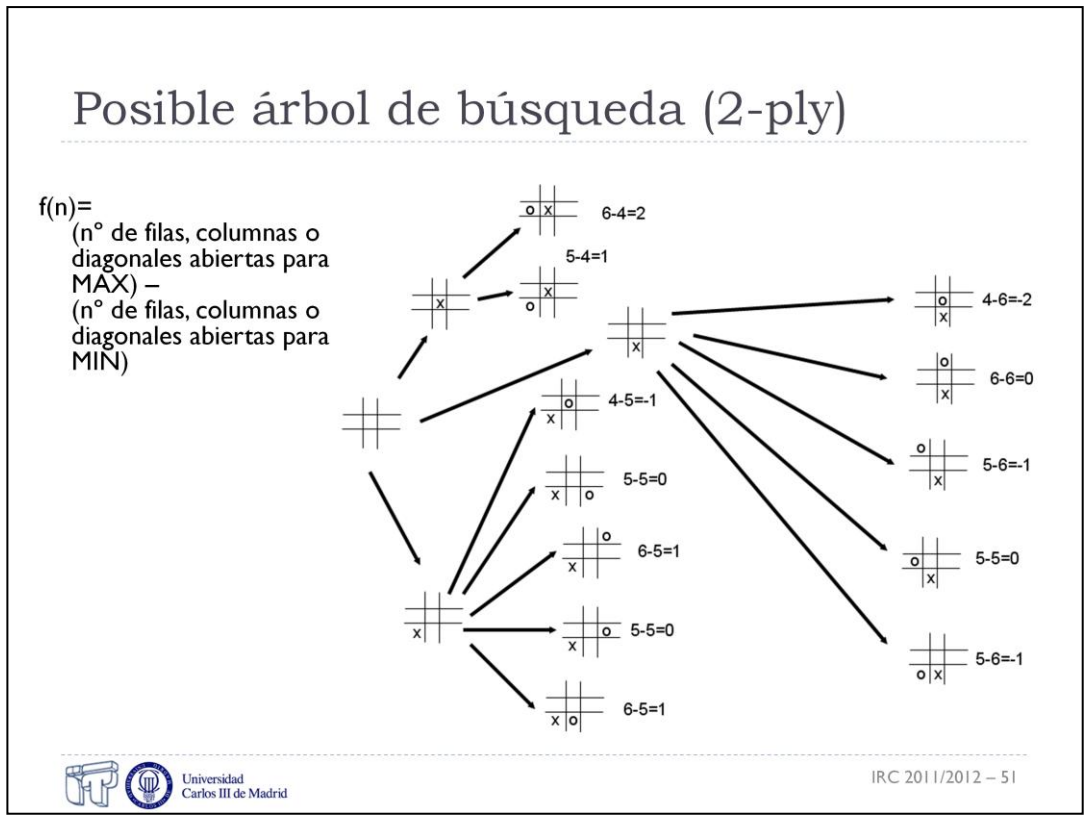
Ver el apartado 5.3 del libro de Béjar.



En el juego de las tres en raya, nos permite aplicar los principios del minimax de manera más o menos abordable con lápiz y papel (al menos los primeros niveles del árbol… luego el tamaño de éste se dispara).

Los estado del problema serían las distintas distribuciones de símbolos en el tablero, y los operadores de cambio de estado los movimientos posibles de cada jugador en su turno. De esta forma, se pasa de un estado a otro cuando uno de los jugadores escribe su símbolo en alguna de las casillas libres del tablero. Esto supone descender un nivel más en el árbol de estados del problema.

Como los movimientos de los jugadores se van alternando, así también lo hacen los niveles del árbol: si un nivel refleja los movimientos posibles del jugador X, el siguiente reflejará las posibles respuestas del jugador O.



Como en cualquier minimax, lo más importante es definir una buena función de utilidad.

En el caso del ejemplo, se está utilizando una función de utilidad muy sencilla: un nodo será mejor para el jugador al que le toca mover si el tablero representado por el nodo tiene más opciones abiertas para ganar él mismo, y menos opciones abiertas para que gane el contrario.

Nos encontramos aquí un caso muy similar al de la función de evaluación en las búsquedas heurísticas: que la estrategia de búsqueda ofrezca mejores o peores resultados depende sobre todo de lo acertado de la función. Y eso a su vez depende de la habilidad del desarrollador… En el caso del juegos, por lo que vemos los algoritmos no juegan solos, entonces: el desarrollador del algoritmo de búsqueda es el verdadero estratega.

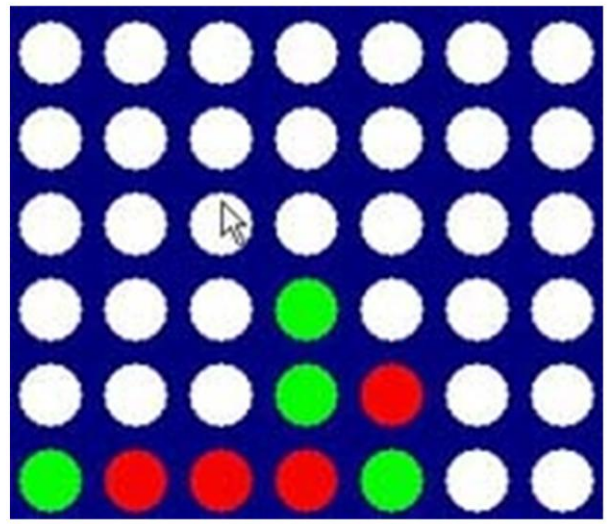
Problema de minimax

- El número de estados del juego es exponencial al número de movimientos
  - (esto es, en la mayoría de juegos, generar el árbol completo es inviable en recursos de memoria)
- Solución: no examinar todos los estados
  - Generar un árbol parcial (anticipar sólo los N movimientos siguientes)
  - Usar heurísticos:
    - Memoria de partidas anteriores
    - Movimientos preferidos
  - Poda alfa-beta *(alpha-beta pruning)*
    - Alfa = valor de la mejor jugada hasta el momento para MAX
    - Beta = valor de la mejor jugada hasta el momento para MIN

    →No expandir (podar) los caminos que no proporcionen mejoras sobre el mejor camino hasta el momento

Ver apartado 5.4 del libro de Béjar.

Las 4 en Raya



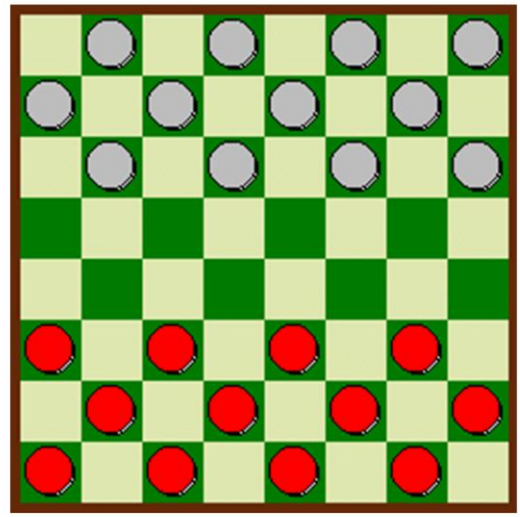
http://www.it.uc3m.es/jvillena/irc/demos/cuatroenraya/cuatroenraya.html

Universidad Carlos III de Madrid

IRC 2011/2012 – 53

¿Serás capaz de ganar al minimax que hay detrás de esta implementación de las 4 en raya? No es fácil, y sin embargo la función de utilidad empleada es relativamente sencilla… Pero ya se sabe que lo eficaz normalmente no está reñido con lo sencillo, sino todo lo contrario (en muchas ocasiones).

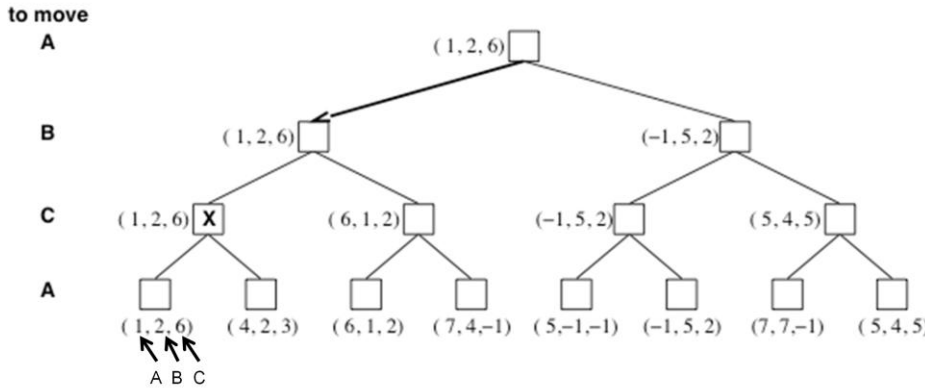
Las Damas



http://www.it.uc3m.es/jvillena/irc/demos/damas/damas.html

Juegos con varios jugadores

- La función de utilidad se convierte en un vector de valores



Universidad Carlos III de Madrid

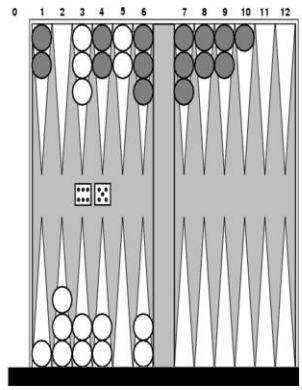
IRC 2011/2012 – 55

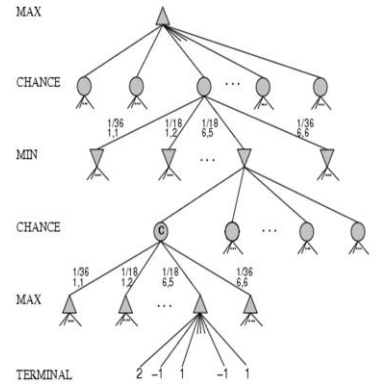
Aunque minimax está concebido para juegos de dos jugadores, se puede modificar para aplicarlo a juegos con varios jugadores.

El cambio más importante que tiene lugar en este caso es en la función de utilidad: ya no calcularemos un único valor, que nos diga cuán bueno es un determinado estado con respecto al otro jugador, sino que necesitaremos calcular tantos valores como rivales tengamos en el juego, para poder evaluar mejor la conveniencia global de cada movimiento.

Obviamente, la decisión final de qué movimiento hacer, ante los resultados de la función de utilidad, se complica…

Juegos con azar





- Se modelan las probabilidades de la jugada mediante un jugador ficticio (CHANCE)
- Por tanto, el valor de la función de utilidad es simplemente aproximado

También se puede incluir la influencia del azar, introduciendo para ello un jugador ficticio que hará las veces de "elemento azaroso" (por ejemplo, del lanzamiento de un dado).

Este jugador ficticio se incluirá en el árbol como si de cualquier otro jugador se tratase, aunque probablemente sus operadores de cambio de estado serán diferentes a los del resto (los "movimientos" que puede hacer un dado son los distintos números que pueden salir al lanzarlo, no los movimientos permitidos para el resto de jugadores).

La principal consecuencia de esto es que la función de utilidad deja de ser plenamente confiable, porque siempre, para alcanzar un determinado estado recomendado por dicha función, dependeremos de que el azar juegue a nuestro favor…

Juegos con suma no nula

- Minimax se puede aplicar a juegos con suma cero y en los que el oponente juega a ganar
  - la ganancia o pérdida de MAX se equilibra exactamente con las pérdidas o ganancias de MIN
- Sin embargo, muchas de las situaciones del mundo real habitualmente tienen suma no nula
  - los participantes pueden beneficiarse o perder al mismo tiempo
  - Ejemplos: ciclismo, las actividades económicas, la guerra
  - En estas situaciones, la moraleja es que resulta mejor maximizar el beneficio conjunto

Paradigma: El dilema del prisionero

La policía arresta a dos sospechosos. No hay pruebas suficientes para condenarles, y tras haberles separado, les visita a cada uno y les ofrece el mismo trato:

	Tú lo niegas	**Tú confiesas**
Él lo niega	Ambos sois condenados a 6 meses	Él es condenado a 10 años; tú sales libre
Él confiesa	Él sale libre; tú eres condenado a 10 años	Ambos sois condenados a 6 años.

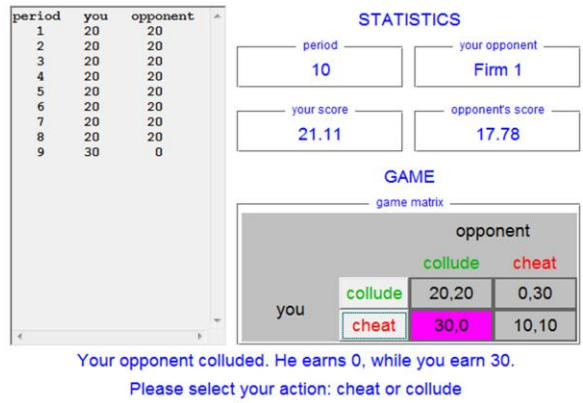
- Estrategia dominante: confesión
  - independientemente de lo que decida el otro, puedes reducir tu condena confesando
  - Sin embargo, el resultado no es óptimo
- Estrategia óptima: colaboración (equilibrio de Nash)
- Variaciones:
  - La decisión en realidad depende de la matriz de costes
  - Dilema del prisionero iterado

IRC 2011/2012 – 58

http://es.wikipedia.org/wiki/Dilema_del_prisionero

El dilema del prisionero iterado

- La decisión cambia según lo que el oponente haya hecho en anteriores ocasiones (si se conoce al oponente)



period	you	opponent
1	20	20
2	20	20
3	20	20
4	20	20
5	20	20
6	20	20
7	20	20
8	20	20
9	30	0

STATISTICS

period: 10 | your opponent: Firm 1

your score: 21.11 | opponent's score: 17.78

GAME

game matrix

		opponent: collude	opponent: cheat
you	collude	20,20	0,30
you	cheat	30,0	10,10

Your opponent colluded. He earns 0, while you earn 30.

Please select your action: cheat or collude

http://www.gametheory.net/Web/PDilemma/