



## Tema 3. Búsqueda en juegos y Búsqueda para problemas de satisfacción de restricciones

## Búsqueda en juegos

- Búsqueda en juegos con adversario
  - Juegos como problemas de búsqueda
  - Estrategias básicas de búsqueda en juegos
  - Técnicas complementarias

## Juegos

- XVIII (1760), [Wolfgang Kempelen](#): ajedrecista mecánico



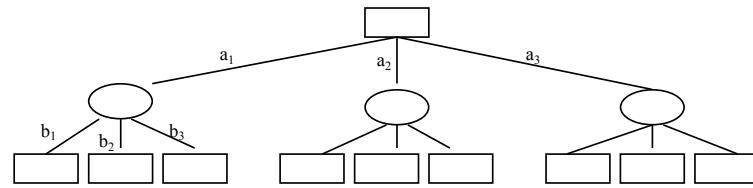
- XIX, [Charles Babbage](#): máquina analítica, ajedrez
- '50 Alan Turing, ajedrez  
<http://www.chessgames.com/perl/chessgame?gid=1356927>
- 1963, [Arthur Samuel](#), damas
- 1997, [Deep Blue](#), ajedrez, gana al campeón mundial de ajedrez
- Ajedrez
  - Cada jugada ▶ 35 posibles movimientos
  - Cada partida ▶ aproximadamente 100 jugadas
  - Búsqueda exhaustiva imposible**

## Juegos como problemas de búsqueda

- Imposible generar todo el árbol de búsqueda
  - Generar hasta un determinado nivel de profundidad
  - Aplicar alguna [función de evaluación f\(N\)](#)
    - Devuelve un valor numérico que indica cómo de bueno es un estado
    - MAX maximizará esta función y MIN minimizará** dicha función.
    - En algunos casos la función nos puede devolver valores como PIERDE, GANA o EMPATA, siempre referidos a MAX
    - Objetivo del análisis del árbol: determinar valor del nodo raíz (inicio de la jugada). A este valor se le denomina valor MiniMax**

## Juegos como problemas de búsqueda

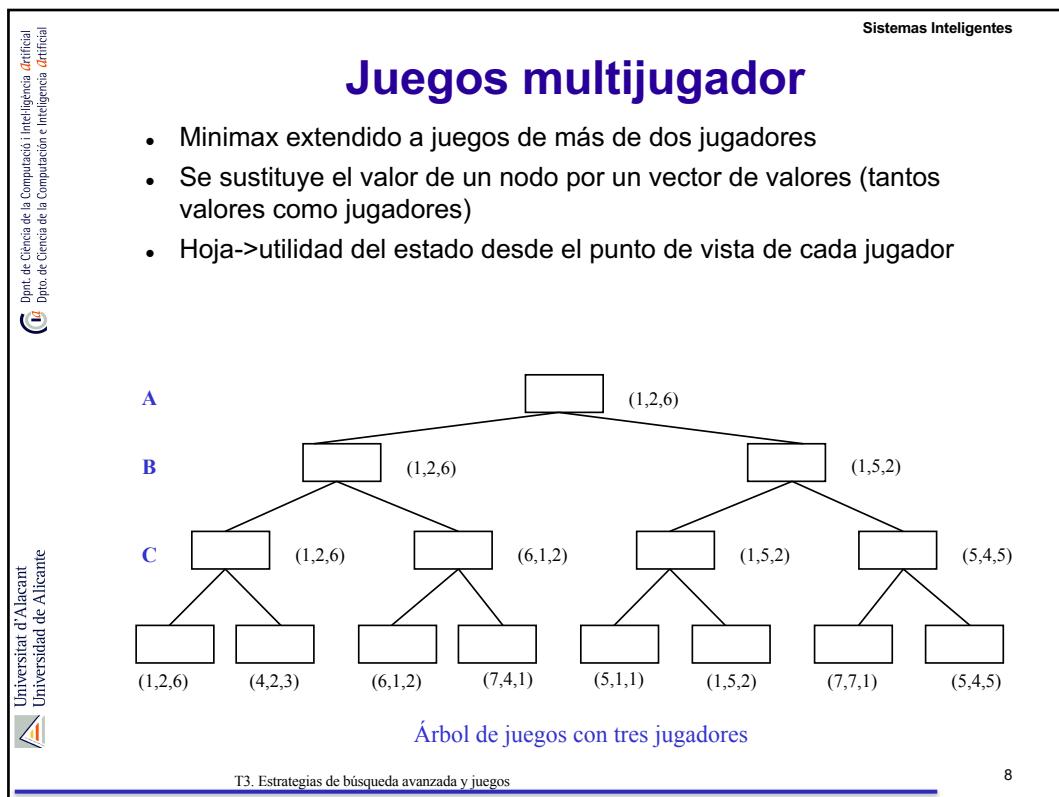
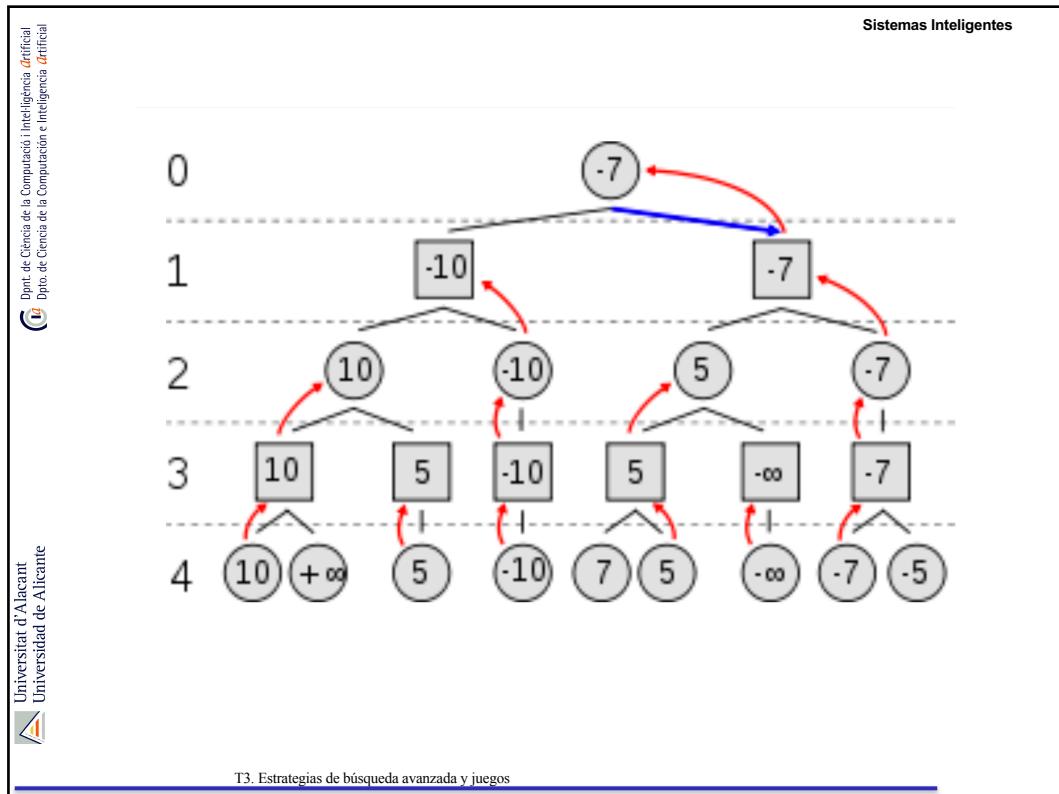
- Estado ( $N$ ): configuración del juego en un momento dado
- Árbol de juego. Cada arista de ese árbol indica un posible movimiento. Una rama completa contempla una posible jugada



- En cada nivel se van alternando los jugadores
- Factor de ramificación ( $B$ ): número de posibles movimientos que se pueden realizar

## Estrategia exhaustiva: MiniMax

- Genera todos los nodos del árbol hasta la profundidad deseada
- Evalúa cada nodo hoja
- Asigna un valor al nodo raíz
  - o si la decisión la toma el jugador MIN, asociar a ese nodo el mínimo de los valores de sus hijos, y el máximo en caso de MAX
- Cuando decimos que aplicamos el algoritmo lo que realizamos es calcular el valor de  $V(N)$  de un determinado nodo.



## Estrategia de poda: $\alpha$ - $\beta$

La poda alfa beta es una técnica de búsqueda que reduce el número de nodos evaluados en un árbol de juego por el Minimax.

$\alpha$  es el valor de la mejor opción hasta el momento a lo largo del camino para MAX, esto implicará por lo tanto la elección del valor más alto

$\beta$  es el valor de la mejor opción hasta el momento a lo largo del camino para MIN, esto implicará por lo tanto la elección del valor más bajo.

El valor MiniMax de un nodo estará siempre acotado

$$\alpha \leq V(N) \leq \beta$$

Al principio inicializamos  $\alpha = -\infty$  y  $\beta = \infty$  al no tener ninguna evidencia.

Esta búsqueda alfa-beta va actualizando el valor de los parámetros según se recorre el árbol. El método realizará la poda de las ramas restantes cuando el valor actual que se está examinando sea peor que el valor actual de  $\alpha$  o  $\beta$  para MAX o MIN, respectivamente.

```

Algoritmo  $\alpha$ - $\beta$  -  $V(N, \alpha, \beta)$ 
Entrada: Nodo  $N$ , valores  $\alpha$  y  $\beta$ .
Salida: Valor minimax de dicho nodo.

Si  $N$  es nodo hoja entonces devolver  $f(N)$ .
sino
  Si  $N$  es nodo MAX entonces
    Para  $k = 1$  hasta  $b$  hacer
       $\alpha = \max[\alpha, V(N_k, \alpha, \beta)]$ 
      Si  $\alpha \geq \beta$  entonces devolver  $\beta$  FinSi.
      Si  $k = b$  entonces devolver  $\alpha$  FinSi.
    FinPara.
  sino
    Para  $k = 1$  hasta  $b$  hacer
       $\beta = \min[\beta, V(N_k, \alpha, \beta)]$ 
      Si  $\alpha \geq \beta$  entonces devolver  $\alpha$  FinSi.
      Si  $k = b$  entonces devolver  $\beta$  FinSi.
    FinPara.
  FinSi
FinSi

```



<http://homepage.ufsp.pt/jtorres/ensino/ia/alfabeta.html>

T3. Estrategias de búsqueda avanzada y juegos

## Técnicas complementarias

- Uso de movimientos de libro
- Espera del reposo
- Técnica de bajada progresiva
- Poda heurística
- Continuación heurística



T3. Estrategias de búsqueda avanzada y juegos

## Uso de movimientos de libro

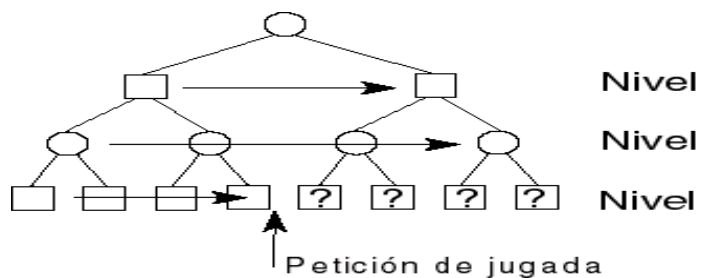
- Imposible seleccionar un movimiento consultando la configuración actual del juego en un catálogo y extrayendo el movimiento correcto.
- Razonable para algunas partes de ciertos juegos.
  - En ajedrez, tanto la secuencia de apertura como los finales están muy estudiados.
- **El rendimiento del programa puede mejorarse si se le proporciona una lista de movimientos** (movimientos de libro) que deberían realizarse en dichos casos.
- Se usa el libro en las aperturas y los finales combinado con el procedimiento MiniMax para la parte central de la partida
- **Conocimiento + búsqueda**

## Espera del reposo

- Busca evitar el efecto horizonte
- Condición adicional de corte de recursión en minimax sería alcanzar una situación estable
- Si se evalúa un nodo y éste cambia su valor de manera drástica después de explorar un nivel más, la búsqueda debe continuar

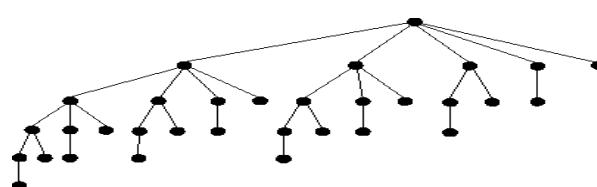
## Técnica de bajada progresiva

- **Restricciones de tiempo:** algoritmos presentados anteriormente no adecuados.
- **Técnica de bajada progresiva**
  - Recorrer nodos por niveles
  - Al llegar la petición de jugada, devolver la solución del último nivel que se haya completado.



## Poda heurística

- Objetivo: **reducir B** desarrollando únicamente los **mejores movimientos** de cada nivel.
- **$g(N)$ :** Función adicional de evaluación
  - De bajo coste.
  - Versión simplificada de  $f(N)$ .
  - Reordenación de nodos: el primer nodo de un nivel es el de mayor  $g(N)$ .
- Factor de ramificación:
 
$$\text{Factor}(Nodo) = \text{Factor}(\text{Padre}(Nodo)) - \text{Rango}(Nodo)$$



## Continuación heurística

- Intento de evitar el **efecto horizonte**.
  - Provocado por la limitación en profundidad: solo se puede tener conocimiento hasta la profundidad seleccionada.
- Secuencia:
  - Desarrollar en anchura hasta un determinado nivel.
  - Seleccionar un subconjunto de nodos terminales para desarrollar búsquedas más profundas.
  - Selección dada por un conjunto de heurísticas directamente relacionadas con el juego.

## Búsqueda para problemas de satisfacción de restricciones

- Formulación de CSPs como redes de restricciones
- Ejemplos
- Métodos de resolución:
  - Esquema backtracking
  - Esquema Forward Checking
  - Esquema de propagación de restricciones

## Problemas de satisfacción de restricciones (CSP)

Conjunto de **variables** definidas sobre **dominios** finitos y conjunto de **restricciones** definidas sobre subconjuntos de dichas variables.

$(V, D, \rho)$

→ un conjunto de **variables**

$$V = \{V_1, V_2, \dots, V_n\} \quad V = \{V_i\}_{i=1..n}$$

→ definidas sobre **dominios** discretos  $D_i$  (conjunto finito de posibles valores)

$$D = \{D_1, D_2, \dots, D_n\} \quad D = \{D_j\}_{j=1..n}$$

→ un conjunto de **restricciones** definidas sobre subconjuntos de dichas variables

$$\rho = \{\rho_1, \rho_2, \dots, \rho_n\} \quad \rho = \{\rho_k\}_{k=1..n}$$

## Problemas de satisfacción de restricciones (CSP)

Ejemplo:

$$X::\{1,2\}, Y::\{1,2\}, Z::\{1,2\}$$

$$X = Y, X \neq Z, Y > Z$$

Solución: encontrar asignaciones de valor a las variables que satisfagan todas las restricciones.

$$(X = 2, Y = 2, Z = 1)$$

Solución al problema: la relación n-aria que satisface todas las restricciones del problema

Dependiendo de los requerimientos del problema hay que encontrar todas las soluciones o sólo una

## Redes de restricciones

- Un CSP se puede representar como un grafo.
- Sobre el grafo se puede definir una red de restricciones:

Quintupla  $\langle V, E, c, l, a \rangle$

- $V$ : conjunto de nodos.
- $E$ : conjunto de aristas.
- $c: E \rightarrow V^k$ ,  $k \leq n$ ; función de asignación de aristas a tuplas de nodos.
- $l: E \rightarrow \rho$ ; función de asignación de aristas a restricciones.
- $a$ : permutación que define el orden de selección para resolver el problema.

## CSP binario

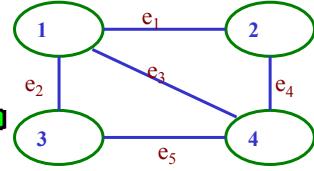
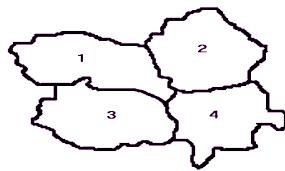
- Variables
$$V = \{V_1, V_2, \dots, V_n\}$$
- Dominios discretos y finitos
$$D = \{D_1, D_2, \dots, D_n\}$$
- Restricciones binarias

$$\{R_{ij}\}$$

Todo problema **n-ario** se puede formular como un problema **binario**

- Ejemplos de CSP binarios:
  - Coloreado de mapas
  - Asignación de tareas para un robot
  - N-reinas
  - Generación de crucigramas

## Ejemplo: Coloreado de mapas



$$V = \{V_1, V_2, V_3, V_4\}$$

$$D_i = \{\text{rojo, azul, verde}\}, \forall i, 1 \leq i \leq 4$$

$$E = \{e_1, e_2, e_3, e_4, e_5\}$$

$$\rho_k(V_i, V_j) = \{<v_i, v_j> | v_i \in D_i, v_j \in D_j, v_i \neq v_j\} \forall k, 1 \leq k \leq 5$$

$$c(e_1) = <V_1, V_2>, c(e_2) = <V_1, V_3>, c(e_3) = <V_1, V_4>, \dots$$

$$l(e_j) = \{<\text{azul, verde}>, <\text{azul, rojo}>, <\text{verde, azul}>, <\text{verde, rojo}>, <\text{rojo, azul}>, <\text{rojo, verde}>\} \forall j, 1 \leq j \leq 5$$

$$a = \{V_1, V_4, V_2, V_3\}$$

## Generación de crucigramas

- Dada una rejilla y un diccionario, construir un crucigrama legal

Slot horizontal de tres letras

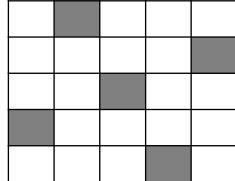
C	O	L
O		
Z		

4 palabras de 1 letra  
4 palabras de 3 letras  
2 palabras de 5 letras

- Formulación:
  - variables** : grupo de casillas para una palabra (*slots*)
  - dominios** : palabras del diccionario con la longitud adecuada
  - restricciones** : misma letra en la intersección de dos palabras
- Características :
  - CSP binario, discreto (dominios grandes)

## N-reinas

- Posicionar  $n$  reinas en un tablero de ajedrez  $n \times n$ , de forma que no se ataquen



$n = 5$

- Formulación: (1 reina por fila)
  - variables : reinas,  $X_i$  reina en la fila  $i$ -ésima
  - dominios : columnas posibles  $\{1, 2, \dots, n\}$
  - restricciones :
    - la misma columna
    - la misma diagonal
- Características :
  - Dominios discretos y restricciones binarias

## Criptoaritmética

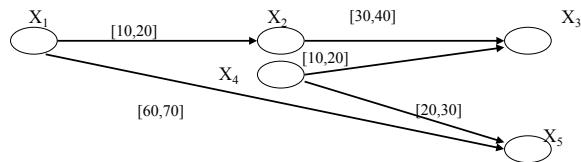
- Sustituir cada letra por un dígito distinto (distinta cifra, distinta letra) de manera que la suma sea correcta
- Formulación:
  - variables :  $G, O, T, A, U, C_1, C_2, C_3$   
 $(C_1, C_2, C_3$  variables de acarreo)
  - dominios :  $O, T, U \in \{0, \dots, 9\}$   
 $G, A \in \{1, \dots, 9\}$   
 $C_1, C_2, C_3 \in \{0, \dots, 4\}$
  - restricciones :
    - letras distintas  $G \neq O, G \neq T, \dots, A \neq U$
    - suma correcta
 

$(unidades)$ $(decenas)$ $(centenas)$ $(unidades millar)$	$5^*A = 10^*C_1 + A$ $5^*T + C_1 = 10^*C_2 + U$ $5^*O + C_2 = 10^*C_3 + G$ $5^*G + C_3 = A$
--	--
- Características :
  - Dominios discretos y restricciones múltiples

$$\begin{array}{r}
 \text{GOTA} \\
 \text{GOTA} \\
 + \text{GOTA} \\
 \hline
 \text{AGUA}
 \end{array}$$

## Restricciones temporales

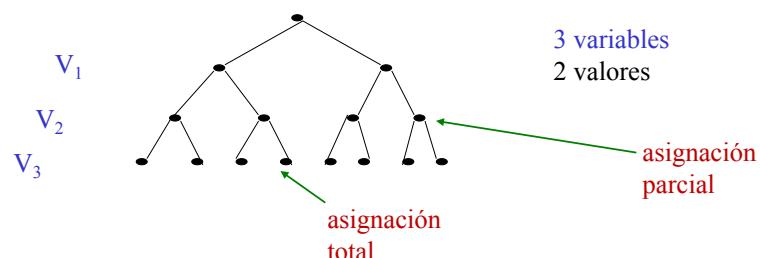
- Dado un conjunto de sucesos que ocurren en intervalos temporales con ciertas relaciones, encontrar un asignación temporal consistente



- Formulación:
  - variables** : sucesos
  - dominios** : intervalo temporal para cada suceso
  - restricciones** : distancia temporal permitida entre sucesos; relaciones temporales antes, después, solapado, etc.
- Características :
  - Dominios continuos y restricciones binarias

## Árbol de interpretaciones

- Partimos de un nodo raíz que supervisa el proceso.
- Cada nivel corresponde a una asignación de valor para una característica de datos. El orden de descenso viene especificado por a.
- Cada nodo identifica una posibilidad de asignación (Variable, valor).
- La solución se construye de forma incremental de tal forma que cada hoja es una interpretación.



## Métodos de resolución

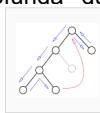
### Búsqueda

**Generación y test** : generar de forma sistemática y exhaustiva cada una de las posibles asignaciones a las variables y comprobar si satisfacen todas las restricciones. Hay que explorar el espacio definido por el producto cartesiano de los dominios de las variables.

**Backtracking** : se trata de construir la solución de forma gradual, instanciando variables en el orden definido por la permutación dada



**Backjumping**: parecido al BT pero el retroceso no se hace a la variable instanciada anteriormente sino a la variable más profunda que está en conflicto con la variable actual.



Explorar el espacio de estados hasta encontrar una solución, demostrar que no existe o agotar los recursos

## Métodos de resolución

### Inferencia

- **Consistencia de arco**
- Consistencia de caminos
- K-consistencia

Deducir un problema equivalente que sea más fácil de resolver

### Algoritmos híbridos

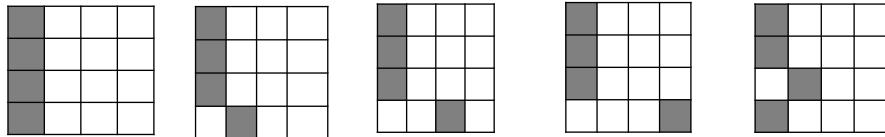
- **Forward Checking**
- Maintaining Arc Consistency
- Heurísticas

Combinación de las aproximaciones anteriores. Sobre un esquema de búsqueda se incorporan métodos de inferencia

## Generación y test

Estrategia:

- generación y test de todas las asignaciones totales posibles
  1. Generar una asignación de todas las variables
  2. Comprobar si es solución. Si es, stop, sino ir a 1



Eficiencia:

- es muy poco eficiente
- genera muchas asignaciones que violan la misma restricción

## Backtracking

Estrategia:

- Construir una solución parcial : asignación parcial que satisface las restricciones de las variables involucradas
- Extender la solución parcial, incluyendo una variable cada vez hasta llegar una solución total
- Si no se puede extender: backtracking
  - cronológico: se elimina la última decisión
  - no cronológico : se elimina una decisión anterior

## Backtracking recursivo

```

procedimiento Backtracking( $k, V[n]$ ) ; Llamada inicial: Backtracking(1,  $V[n]$ )
inicio
     $V[k] = Selección(d_k)$  ; Selecciona un valor de  $d_k$  para asignar a  $x_k$ 
    si Comprobar( $k, V[n]$ ) entonces
        si  $k = n$  entonces
            devolver  $V[n]$  ; Es una solución
        si no
            Backtraking( $k + 1, V[n]$ )
        fin si
    si no
        si quedan_valores( $d_k$ ) entonces
            Backtraking( $k, V[n]$ )
        si no
            si  $k = 1$  entonces
                devolver  $\emptyset$  ; Fallo
            si no
                Backtraking( $k - 1, V[n]$ )
            fin si
        fin si
    fin si
fin Backtracking

```

## Limitaciones del backtracking

- **Trashing e inconsistencia de nodo**

Relacionado con las **restricciones unarias**. Sigue cuando un dominio contiene un valor que no satisface una restricción unaria.

- **Inconsistencia de arista**

Relacionado con las **restricciones binarias**. Sigue cuando existe una restricción binaria entre dos variables de tal forma que para un determinado valor de la primera variable no existe ninguna asignación posible para la segunda.

- **Dependencia de la ordenación**

El orden de selección de las variables es un factor crítico. Se han desarrollado diversas heurísticas de selección de variable y de valor.

**Variable:** Orden estático y Orden dinámico

**Valor:** p.e. los que conducen a un CPS más simple

## Forward checking

- En cada etapa de la búsqueda, FC comprueba hacia delante la asignación actual con todos los valores de las futuras variables que están restringidas con la variable actual.
- Los valores de las variables futuras que son inconsistentes con la asignación actual son temporalmente eliminados de sus dominios.
- Si el dominio de una variable futura se queda vacío, la instanciación de la variable actual se deshace y se prueba con un nuevo valor. Si ningún valor es consistente, entonces se lleva a cabo el backtracking cronológico.

## Forward checking

1. Seleccionar  $x_i$ .
2. Instanciar  $x_i \leftarrow a_i : a_i \in D_i$ .
3. Razonar hacia adelante (forward-check):
  - Eliminar de los dominios de las variables  $(x_{i+1}, \dots, x_n)$  aún no instanciadas, aquellos valores inconsistentes con respecto a la instanciación  $(x_i, a_i)$ , de acuerdo al conjunto de restricciones.
4. Si quedan valores posibles en los dominios de todas las variables por instanciar, entonces:
  - Si  $i < n$ , incrementar  $i$ , e ir al paso (1).
  - Si  $i = n$ , salir con la solución.
5. Si existe una variable por instanciar, sin valores posibles en su dominio, entonces retractar los efectos de la asignación  $x_i \leftarrow a_i$ . Hacer:
  - Si quedan valores por intentar en  $D_i$ , ir al paso (2).
  - Si no quedan valores:
    - Si  $i > 1$ , decrementar  $i$  y volver al paso (2).
    - Si  $i = 1$ , salir sin solución.

## Forward checking

```

funcion FC(i variable): booleano
    para cada a ∈ factibles[i] hacer
        Xi ← a
        si i=N solución retorna CIERTO
        sino
            si forward (i,a)
                si FC(i+1) retorna CIERTO
                Restaurar (i)
            retorna FALSO
funcion forward(i variable, a valor): booleano
    para toda j=i+1 hasta N hacer
        Vacío ← CIERTO
        para cada b ∈ factibles[j] hacer
            si (a,b) ∈ Rij vacío ← FALSO
            sino eliminar b de factible[j]
                Añadir b a podado[j]
            si vacío retorna FALSO
        retorna CIERTO
procedimiento restaura(i variable)
    para toda j=i+1 hasta N hacer
        para todo b ∈ podado[j] hacer
            si Xi responsable filtrado b
                Eliminar b de podado[j]
                Añadir b a factible[j]

```

T3. Estrategias de búsqueda avanzada y juegos

37

## Forward Checking

Ejemplo Forward Checking:

Variables *x, y*  
 $Dx = Dy = \{1,2,3,4,5\}$   
 Restricción  $x < y - 1$

Inicialmente  $CDx = CDy = \{1,2,3,4,5\}$

Si asignamos  $x = 2$ , entonces:  
 Los únicos valores que puede tomar *y* son 4, 5  
 por tanto  $CDy = \{4,5\}$

..

..

Si asignamos  $x = 4$ , entonces:  
 No hay asignación posible compatible con la restricción.  
 por tanto  $CDy = \{\}$   
 Deshacer  $x = 4$  y backtracking

T3. Estrategias de búsqueda avanzada y juegos

38

## Propagación de Restricciones

- Transformar el problema en otro más sencillo sin inconsistencias de arco.
- Propiedad de consistencia de arista  
 $c(e_p) = \langle V_i, V_j \rangle$  es consistente si y sólo si para todo valor asignable a  $V_i$  existe al menos un valor en  $V_j$  que satisface la restricción asociada a la arista.
- Un CSP puede transformarse en una red consistente mediante un algoritmo sencillo (AC3) que examina las aristas, eliminando los valores que causan inconsistencia del dominio de cada variable.
- Después del proceso:
  - No hay solución
  - Hay más de una solución
  - Hay una única solución

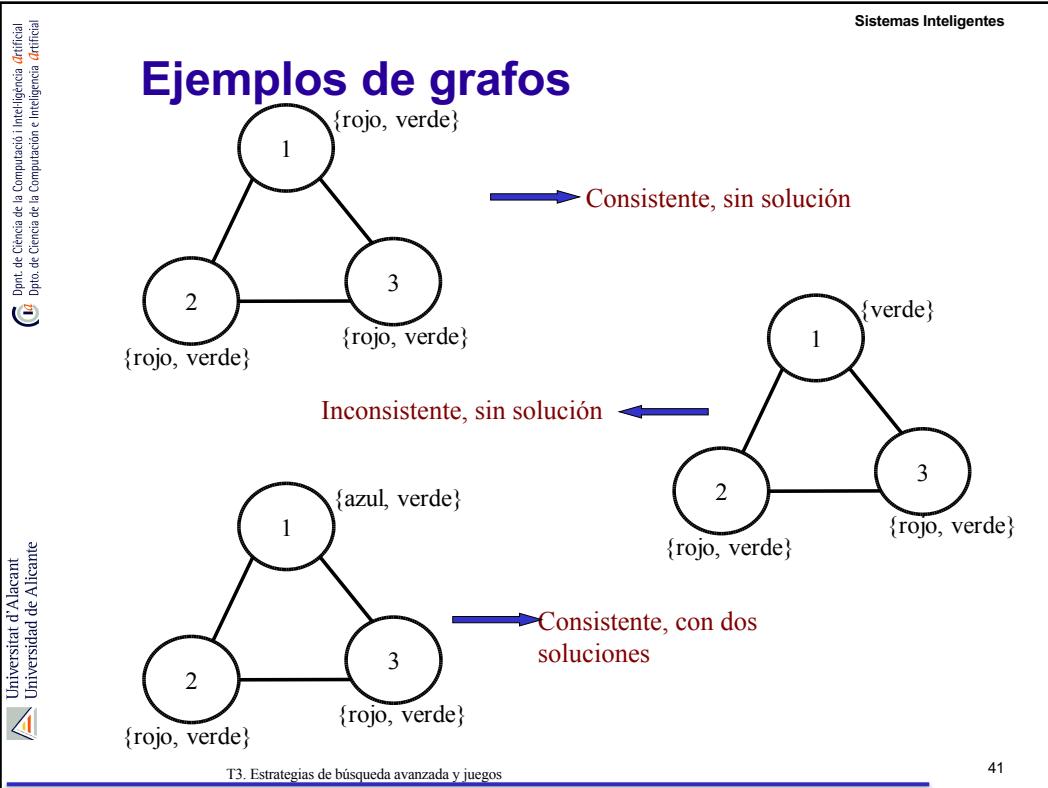
## Algoritmo AC3

```

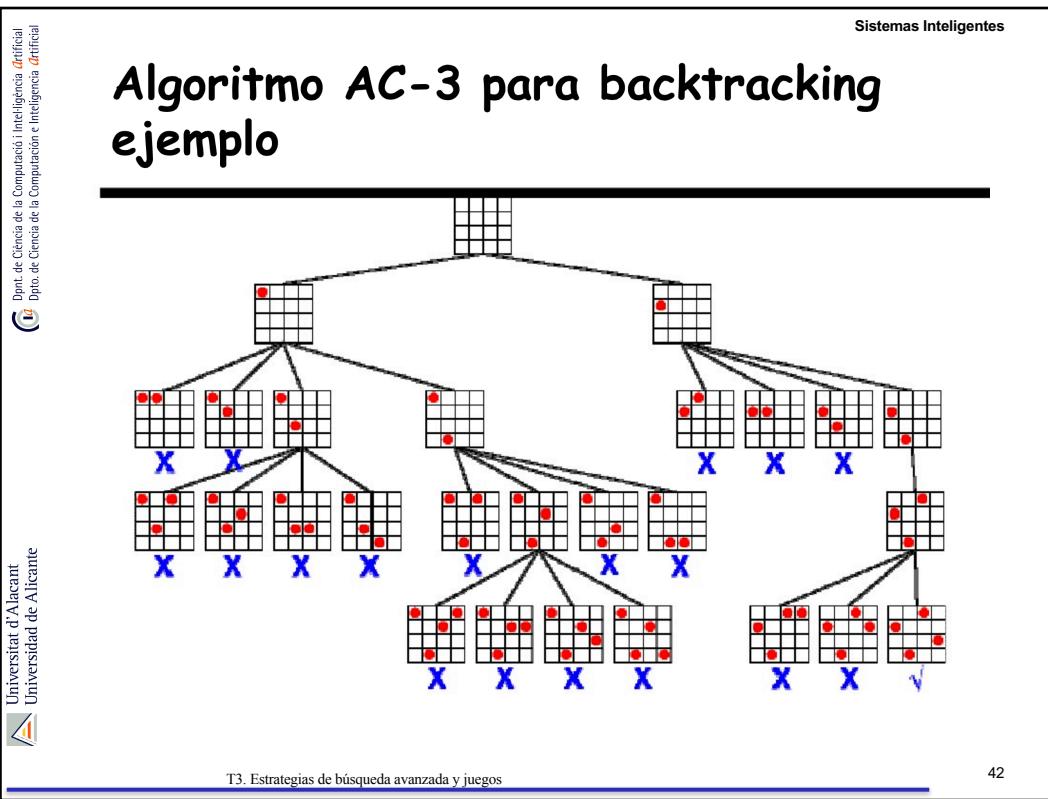
 $Q = \{c(e_p) = \langle V_i, V_j \rangle | e_p \in E, i \neq j\}$ 
Mientras  $Q \neq \emptyset$  hacer
   $\langle V_k, V_m \rangle = \text{seleccionar\_y\_borrar}(Q)$ 
  cambio = falso
  Para todo  $v_k \in D_k$  hacer
    Si no_consistente ( $v_k, D_m$ ) entonces
      borrar ( $v_k, D_k$ )
      cambio = cierto
    FinSi
  FinPara
  Si  $D_k = \emptyset$  entonces salir_sin_solución FinSi
  Si cambio = cierto entonces
     $Q = Q \cup \{c(e_r) = \langle V_i, V_k \rangle | e_r \in E, i \neq k, i \neq m\}$ 
  FinSi
FinMientras

```

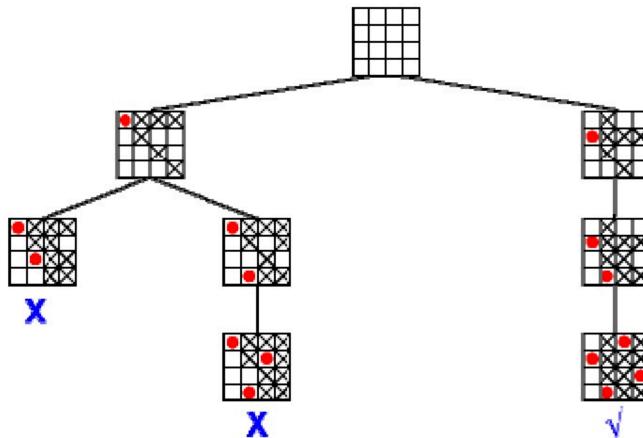
## Ejemplos de grafos



## Algoritmo AC-3 para backtracking ejemplo



## Algoritmo AC-3 para Forward checking: ejemplo



## Tema 3. Estrategias de búsqueda. Bibliografía

- Stuart Russell, Peter Noving. "Inteligencia Artificial. Un enfoque Moderno" Ed. Pearson. Prentice Hall. 2004.