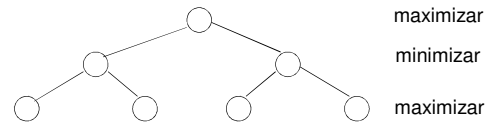


Juegos y Búsqueda Competitiva

- La mayoría de los juegos con 2 jugadores requiere jugadas sucesivas de cada jugador
 - Cada transición es un movimiento
 - La mayoría de juegos no triviales no permite
 - Búsqueda exhaustiva (árboles muy grandes)
 - Necesita utilizar búsqueda y heurística
- Definiciones:
 - Estado inicial: configuración inicial e indicación de quien debe iniciar el movimiento
 - Operadores: definen los movimientos permitidos
 - Ply: número de niveles en el árbol incluyendo la raíz
 - Prueba terminal: define cuando el juego termina
 - Función de utilidad provee un valor numérico para el resultado del juego
- Los juegos envuelven competición
 - Los competidores trabajan para conseguir ganar
 - Los árboles difieren de los anteriores ya que las jugadas de cada jugador tienen objetivos contradictorios
 - No existe una búsqueda para un mismo objetivo
 - Validación estática: valor numérico que representa la calidad de la configuración
 - Realizada por un validador estático
 - Valores positivos ventaja para un jugador
 - Valores negativos ventaja para el otro

Procedimiento Minimax

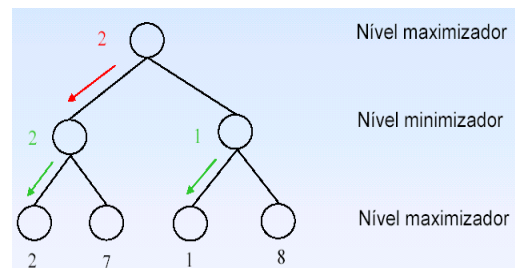
- Maximizador: jugador esperando por valores positivos
- Minimizador: jugador esperando por valores negativos
- Árbol de juego consiste de capas sucesivas de maximización y minimización
- Se asume que en cada capa el jugador desea el valor mas ventajoso para poder ganar.



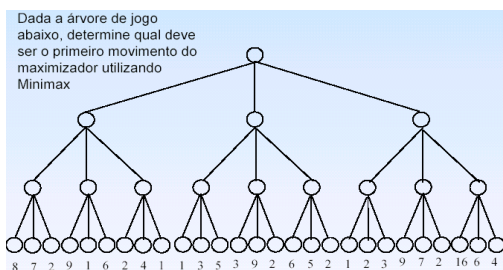
Procedimiento Minimax

- Imagine que somos siempre el maximizador
 - Cuando el examinamos el árbol siempre queremos tener el mayor valor estático en cada nivel
 - Asumir que el oponente también tiene acceso a la misma información y los valores de cada nivel
 - Por lo tanto el oponente intentara prevenir maximizar la evaluación
 - Procedimiento minmax: estrategia de búsqueda en árboles de juegos en el cual:
 - Ply en el nivel p: árbol extendida hasta la profundidad p
 - Evaluación estática realizada para todas las configuraciones expandidas
 - Suponer que el oponente hará que realice movimientos indeseables mejor para el y peor para Ud.
- Para ejecutar una búsqueda minmax utilizando el procedimiento minmax
 - Si el limite de búsqueda fue alcanzado, calcule la evaluación estática de la posición recurrente relativa al jugador apropiado. Reporte el resultado
 - En caso contrario si el nivel es minimizador use el procedimiento minmax en los hijos de la posición actual y reporte el resultado mínimo
 - En caso contrario, el nivel es maximizador. Use el procedimiento minmax en los hijos, y reporte el valor máximo

Ejemplo Minmax



Ejercicio Minmax



Procedimiento Minmax

- Puntos fuertes
 - Asume que el oponente es tan inteligente como UD.
 - Búsqueda continua mientras el adversario esta pensando
- Puntos débiles
 - Una simple evaluación estática es descriptivamente pobre
 - Requiere que todo el árbol sea generado
 - Puede ser computacionalmente costoso

Procedimiento Alfa Beta

- Poda: corte de los ramos menos productivos
 - No es necesario explorar todos los ramos
 - Una decisión cualitativa sobre la viabilidad de la búsqueda de un ramo en particular es realizada
- Principio alfa beta
 - Si se tiene certeza que un ramo es pobre entonces no pierda tiempo en explorar dicho ramo.

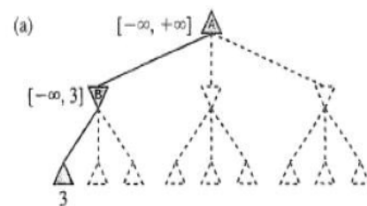
Procedimiento Alfa Beta

- Procedimiento alfa beta
 - Junto con el minmax previene que se realice una evaluación innecesaria de todos los ramos del árbol
 - La decisión de ignorar un ramo es realizada basada en el conocimiento de aquello que el oponente hará para que un movimiento claramente favorable este disponible para él

Procedimiento Alfa Beta

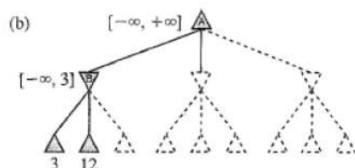
- Para ejecutar una búsqueda minmax usando el procedimiento alfa beta
 - Si el nodo es la raíz, $\alpha = -\infty$, $\beta = +\infty$
 - Si el límite de búsqueda fue conseguido, calcule la evaluación estática y reporte el resultado
 - Si el nivel es minimizador
 - Hasta que todos los hijos sean examinados o hasta que $\alpha \geq \beta$
 - Use el procedimiento alfa beta con los valores de α y β , en un nodo hijo; anote el valor calculado
 - Compare el valor con β si el valor reportado es menor que β , hacer que β sea igual a ese valor
 - Reporte β
 - Caso contrario el nivel es maximizador
 - Hasta que todos los hijos sean examinados o hasta que $\alpha \geq \beta$
 - Use el procedimiento alfa beta con los valores de α y β , en un nodo hijo; anote el valor calculado
 - Compare el valor con α si el valor reportado es menor que α , hacer que α sea igual a ese valor
 - Reporte α

Procedimiento Alfa Beta



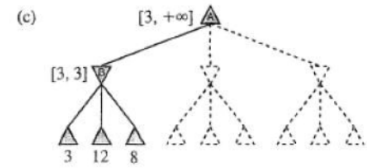
Como la primera hoja debajo de B tiene valor 3 y siendo B un nodo MIN, tiene como valor máximo a 3.

Procedimiento Alfa Beta



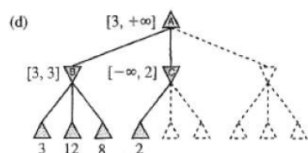
La segunda hoja de B tiene valor de 12, por lo tanto MIN evitara este movimiento, debido a que el número es mayor que 3.

Procedimiento Alfa Beta



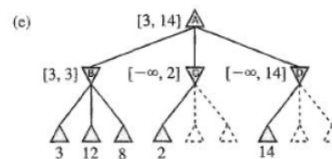
El siguiente nodo tiene valor de 8, quedando el valor de B en 3 y hasta el momento seria 3 para el nodo raíz.

Procedimiento Alfa Beta



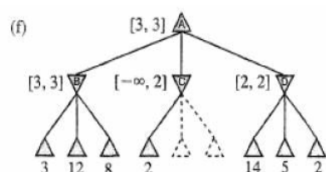
La primera hoja de C es de valor 2, como C es un nodo MIN con el máximo valor de 2 y B vale 3, el nodo raíz MAX nunca elegiría a C, y no es necesario revisar sus nodos sucesores.

Procedimiento Alfa Beta



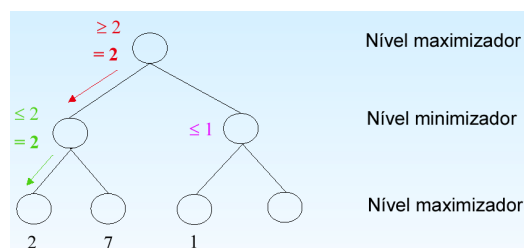
La primera hoja de D es 14 y como es mas alto que el 3 de B, existe la necesidad de seguir los demás nodos hojas.

Procedimiento Alfa Beta



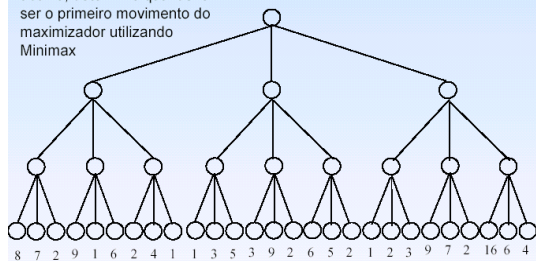
El siguiente valor de D es 5, seguimos explorando; el tercer valor es 2 así que D vale 2. La decisión de MAX en la raíz es moverse a B, quedando como valor final 3

Procedimiento Alfa Beta



Procedimiento Alfa Beta

Dada a árvore de jogo abaixo, determine qual deve ser o primeiro movimento do maximizador utilizando Minimax



Procedimiento Alfa Beta

- En el mejor caso puede ser demostrado que el procedimiento alfa beta corta la tasa de crecimiento en $\frac{1}{2}$ o sea $b^d = b^{d/2}$
 - Donde b es el número de hijos de cada nodo
 - d es la profundidad del árbol
- A pesar de ser menor, el crecimiento exponencial aun esta presente

Procedimiento Alfa Beta

- Ventajas
 - Método eficiente para determinar cuando ciertos ramos deben ser explorados
 - Nuevamente se asume que el oponente tiene la misma información heurística que ud.
- Punto negativo
 - Aun existe el crecimiento exponencial
- Parece paradójico que todo un ramo sea eliminado

Procedimientos Heurísticos

- Los programas de búsqueda en juegos son generalmente en tiempo real y tienen restricciones de tiempo
 - El valor de p es crítico debido al tiempo para análisis
 - p pequeño análisis poco aceptable
 - p grande mucho tiempo requerido
 - Difícil determinar la profundidad a ser analizada dadas las características de software y hardware

Procedimientos Heurísticos

- Profundidad progresiva
 - Analizar hasta profundidad 1, luego profundidad 2, 3 así sucesivamente hasta que el tiempo lo permita
 - Cuando se limita profundidad puede limitar la búsqueda
 - Las decisiones están basadas en informaciones limitadas, como consecuencia decisiones que parecen buenas pueden llevar a decisiones indebidas si la búsqueda continua

Constraint Satisfaction Problems Problema de Satisfacción de Restricciones

Constraint Satisfaction Problems (CSP)

- Problema de Satisfacción de Restricciones
 - tipo de problema que impone **propiedades estructurales** adicionales a la **solución** a ser encontrada
 - hay una demanda mas refinada que en la búsqueda clásica
 - Ejemplo: ir de Recife a Cajazeiras máximo con 3 tanques de gasolina y 7 horas de viaje
- Un CSP consiste en:
 - un conjunto de **variables** que pueden asumir valores dentro de un dominio dado
 - un conjunto de **restricciones** que especifican **propiedades de la solución**
 - valores que esas variables *pueden* asumir

CSP: Formulación

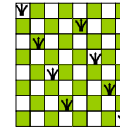
- **Estados:** definidos por valores posibles de las variables
- **Estado inicial:** ninguna variable instanciada aun
- **Operadores:** atribuyen valores las variables
- **Test de termino:** verificar si todas las variables estan instanciadas obedeciendo las restricciones del problema
- **Solución:** conjunto de valores de las variables instanciadas
- **Costo del camino:** número de pasos de atribución

CSP: características de las restricciones

- El conjunto de valores que la variable puede asumir es llamado de **dominio** (D_i)
 - El dominio puede ser **discreto** (fabricantes de una pieza de carro) o **continuo** (peso de las piezas del carro)
- Cuanto a la dimension, de las restricciones pueden ser
 - unárias (sobre una única variable)
 - binárias (sobre dos variables) - ex. 8-reynas
 - n-árias - ex. palabras cruzadas
 - la restriccion unária es un sub-conjunto del dominio, mientras que la n-ária es un producto cartesiano de los dominios
- En cuanto a la naturaleza, de las restricciones pueden ser
 - absolutas** (no pueden ser violadas)
 - Preferenciales** (deben ser satisfechas en lo posible)

Ejemplo

- Juego de las 8-reynas
 - Variables: localización de las reynas
 - valores: posibles posiciones del tablero
 - Restriccion binária: dos reynas no pueden estar en la misma columna, línea o diagonal
 - solución: valores para los cuales la restriccion es satisfecha



Busqueda ciega para CSP

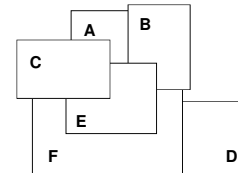
- Funcionamiento
 - estado inicial: variables sin atribucion
 - aplica operador: instanciar una variable
 - test de parada: todas las variables instanciadas sin violaciones
- Análisis
 - Puede ser implementada con **busqueda en profundidad limitada** (l = número de variables)
 - es completa
 - factor de expansión: $\sum_i |D_i|$
 - El test de parada es descompuesto en un conjunto de restricciones sobre las variables

Ejemplo: coloración de mapas

Simulación paso a paso...

A = green
B = green
C = green
D = green
E = green
 F = green (falla c/ C, E, B, D)
F = red
 E (falla c/ C, A, B)
 E = red (falla c/ F)
E = blue
 C (falla c/ A)
 ...
 Solucion muy a la ligera

Variables : A,B,C,D,E,F
dominio: {green,red,blue}
restricciones: A ≠ B; A ≠ C; A ≠ E; B ≠ E; B ≠ F; C ≠ E; C ≠ F; D ≠ F; E ≠ F



Backtracking en la Busqueda Ciega

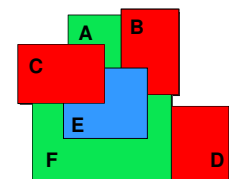
- Problema de la busqueda en profundidad
 - perdida de tiempo, pues continua a pesar que una restriccion ya haya sido violada
 - no se puede remediar el error
- Solución : Backtracking
 - despues de realizar una atribución, se verifica que las restricciones no sean violadas
 - caso haya violación \Rightarrow backtrack

Ejemplo: coloración de mapas

Simulación paso a paso...

A = green
 B = green (falla c/ A)
B = red
 C = green (falla c/ A)
C = red
D = green
 E = green (falla c/ A)
 E = red (falla c/ B e C)
E = blue
 F = green (falla c/ D)
 F = red (falla c/ C)
 F = blue (falla c/ E)
 F backtrack
 E backtrack
D = red
 E = green (falla c/ A)
 E = red (falla c/ B)
E = blue
F = green

variables: A,B,C,D,E,F
dominio: Da=Db...=Df={green,red,blue}
restricciones: A ≠ B; A ≠ C; A ≠ E; B ≠ E; B ≠ F; C ≠ E; C ≠ F; D ≠ F; E ≠ F

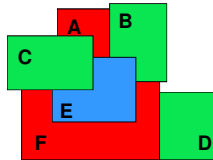


Exemplo: coloração de mapas

Podria ser mas complicado
comezando por red...

A=red
B=green
C=blue
D=red
E=? Backtracking
D=green
E=? Backtracking
D=blue
E=? Backtracking
D=? Backtracking
C=green
D=green
E=blue
F=red

Variações : A,B,C,D,E,F
domínio: Da=Db...=Df={green,red,blue}
restricciones: A ≠ B; A ≠ C; A ≠ E; B ≠ E; B
≠ F; C ≠ E; C ≠ F; D ≠ F; E ≠ F



Backtracking no basta...

- Problema del backtracking:
 - no tiene sentido mover la 7a. reyna para intentar posicionar la última
 - El problema esta mas arriba...
 - El *backtrack* tiene que ser de mas de un paso
- Soluciones
 - Verificación de arco-consistencia (*forward checking*)
 - Propagación de restricciones

Busqueda Ciega - Refinamientos

- Verificación previa (*forward checking*)
 - idía: mirar al frente para detectar situaciones insolubles
 - ej. En el restaurante *self-service* o en el bar...
- Algoritmo:
 - Despues cada atribución, elimina del dominio de las variables no instanciadas los valores incompatibles con las atribuciones hechas hasta ahora
 - Si un dominio se torna vacio, *backtrack* inmediatamente
- Es mas eficiente!

Propagación de Restricciones

- Forward checking* es un caso particular de verificación de *arco-consistencia*
 - un estado es arco-consistente si el valor de cada variable es consistente con las restricciones sobre esta variable
 - arco-consistencia es obtenida por sucesivas eliminaciones de valores inconsistentes
- Propagación de restricciones (*constraint propagation*)
 - una consecuencia de la verificación de arco-consistencia
 - cuando un valor es eliminado, otros pueden tomarse inconsistentes y tienen que ser eliminados también
 - es como una onda que se propaga: las seleccón queda cada vez mas restringidas

Propagacion de restricciones Ejemplo: coloracion de mapas

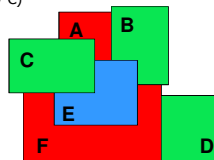
Passo a passo...

variáveis: A,B,C,D,E,F
domínios = {red,green,blue}

A=red
=> B, C, E = {green,blue} (restricciones c/ A)
=> D, F = {red,green,blue}
B=green
=> E = {blue}, F = {red, blue} (restricciones c/ B)
=> C = {green,blue}, D = {red,green,blue}
C=green
=> E = {blue}, F = {red, blue} (restricciones c/ C)
=> D = {red,green,blue}
D=red, E=blue, F=?

Backtracking!!

D=green, E=blue, F=red



Heurísticas para CSP

- Intentan reducir el factor de expansion del espacio de estados
- Donde puede entrar una heurística?
 - Ordenando la seleccón de la variable a instanciar
 - Ordenando la seleccón del **valor** a ser asociado a una variable
- Existen 3 heurísticas para esto...
 - Variable mas restrictiva:** variable involucrada en el mayor número de restricciones es preferida
 - Variable mas restringida:** variable que puede asumir menos valores es preferida
 - valor menos restrictivo:** valor que deja mas libertad para futuras selecciones

Variável mas restrictiva

(variable envuelta en el mayor número de restricciones)

Candidatas!: E, F, ...resto

E = green
Candidatas: F, ...resto

F = red
Candidatas: A, B, C, D

A = red
Candidatas: B, C, D

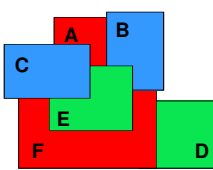
B = blue
Candidatas: C, D

C = blue

D = green

SIN BACKTRACK!!

1 en orden de prioridad



Variable mas restringida

(variable que puede asumir menos valores)

Candidatas: todas

A = green
Candidatas: B, C, E, ...

B = red
Candidatos: E, F, ...

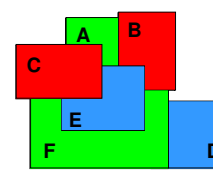
E = blue
Candidatos: C, F, D

C = red
Candidatos: F, D

F = green

D = blue o red

SIN BACKTRACK!!



Valor menos restrictivo

(valor que deja mas libertad)

Variables : A,B,C,D,E,F
dominio: Da=Db...=Df={green,red,blue}
restricciones: A ≠ B; A ≠ C; A ≠ E; B ≠ E; B ≠ F; C ≠ E; C ≠ F; D ≠ F; E ≠ F

Comenzando con

A = green
B = red
C = ??? red es mejor que blue

