Inteligencia Artificial

Búsqueda Adversaria

Sandra Roger

Facultad de Informática
Universidad Nacional del Comahue

2º Cuatrimestre de 2016

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

¿Qué vimos?

- Definiciones de IA (pensamiento vs.comportamiento - humanidad vs. racionalidad).
- RRS
- Agentes Basados en Conocimiento
- Agentes Solucionadores de Problemas
- Búsqueda Ciega: definición de un problema de búsqueda. Algoritmos
- Búsqueda Heurística: Admisibilidad.
- Búsqueda No Exhaustiva

HOY

Búsqueda Adversaria

¿Qué vimos?

- Definiciones de IA (pensamiento vs.comportamiento - humanidad vs. racionalidad).
- RRS
- Agentes Basados en Conocimiento
- Agentes Solucionadores de Problemas
- Búsqueda Ciega: definición de un problema de búsqueda. Algoritmos
- Búsqueda Heurística: Admisibilidad.
- Búsqueda No Exhaustiva

HOY

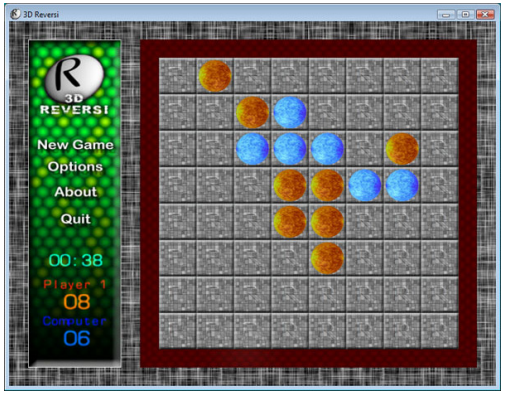
Búsqueda Adversaria

Referencia Bibliográfica

S. Russell y P.Norvig
Artificial Intelligence: A Modern Approach (Third Edition).
Capítulo 5

Juegos

Fútbol



Reversi

Ajedrez

Truco

Juegos

Definición

Un juego consiste en un conjunto de reglas que rigen una situación competitiva en la cual dos o más agentes eligen estrategias diseñadas para maximizar sus propias ganancias o para minimizar las ganancias de sus oponentes.

Las reglas especifican las acciones posibles de cada jugador, la cantidad de información recibida por cada uno de ellos mientras se juega y la cantidad de ganancia o pérdida en varias situaciones.

Tipos de Juegos

JUEGOS	Determinísticos	Con Azar
Información Perfecta	Tic-tac-toe, Damas Reversi, Ajedrez	Backgammon, Monopolio
Información Imperfecta	Batalla Naval	Poker

Tipos de Juegos

JUEGOS	Determinísticos	Con Azar
Información Perfecta	Tic-tac-toe, Damas Reversi, Ajedrez	Backgammon, Monopolio
Información Imperfecta	Batalla Naval	Poker

Características de los Juegos

- Dos jugadores o agentes.
- Movimientos intercalados. Cada decisión es hecha en forma secuencial (no hay movimientos simultáneos).
- Suma Cero: la ganancia de uno es la pérdida del otro.Por ejemplo, en el ajedrez uno gana (1) y el otro pierde (-1); o bien empatan (0).
- No interviene el azar: por ejemplo, dados.
- Información Perfecta: ambos jugadores tienen acceso a toda la información sobre el estado del juego. No se ocultan información uno al otro.

No Suma Cero: Dilema del Prisionero

Enunciado Clásico

La policía arresta a dos sospechosos. No hay pruebas suficientes para condenarlos y, tras haberlos separado, los visita a cada uno y les ofrece el mismo trato. Si uno confiesa y su cómplice no, el cómplice será condenado a la pena total, diez años, y el primero será liberado. Si uno calla y el cómplice confiesa, el primero recibirá esa pena y será el cómplice quien salga libre. Si ambos confiesan, ambos serán condenados a seis años. Si ambos lo niegan, todo lo que podrán hacer será encerrarlos durante seis meses por un cargo menor.

No Suma Cero: Dilema del Prisionero

Resumiendo:

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

Juegos

Teoría de los Juegos	Terminología en IA
Dos jugadores	Dos agentes
Determinísticos	Determinísticos
Por turnos alternados	Acciones por turnos alternados
Información Perfecta	Totalmente observable
Suma a cero	Valores de Utilidad iguales y opuestos

Juegos

Teoría de los Juegos	Terminología en IA
Dos jugadores	Dos agentes
Determinísticos	Determinísticos
Por turnos alternados	Acciones por turnos alternados
Información Perfecta	Totalmente observable
Suma a cero	Valores de Utilidad iguales y opuestos

Juegos

Teoría de los Juegos	Terminología en IA
Dos jugadores	Dos agentes
Determinísticos	Determinísticos
Por turnos alternados	Acciones por turnos alternados
Información Perfecta	Totalmente observable
Suma a cero	Valores de Utilidad iguales y opuestos

Juegos

Teoría de los Juegos	Terminología en IA
Dos jugadores	Dos agentes
Determinísticos	Determinísticos
Por turnos alternados	Acciones por turnos alternados
Información Perfecta	Totalmente observable
Suma a cero	Valores de Utilidad iguales y opuestos

Juegos

Teoría de los Juegos	Terminología en IA
Dos jugadores	Dos agentes
Determinísticos	Determinísticos
Por turnos alternados	Acciones por turnos alternados
Información Perfecta	Totalmente observable
Suma a cero	Valores de Utilidad iguales y opuestos

Formalización como un problema de búsqueda

- Nodo Inicial: incluye la posición inicial del juego y determina el jugador que va a mover.
- Función Sucesor: indica los movimientos legales desde un estado y el estado resultante luego de jugar esa jugada.
- Test Terminal: determina cuando terminó el juego (estados terminales).
- Función de Utilidad: Da valores numéricos a los estados terminales. Recordemos la condición de suma cero.

Formalización como un problema de búsqueda

Árbol de Juego

- El estado inicial y los movimientos legales definen el árbol de juego para un juego.
- Cada nivel del árbol corresponde a las jugadas de uno de los participantes. Llamaremos a los jugadores MAX y MIN.
- Consideramos un turno completo cuando ambos jugadores han jugado. En el árbol del juego, se consideran dos niveles de profundidad un turno.

Solución Óptima

En un problema de búsqueda, la solución óptima es una secuencia de movimientos que lleva desde el estado inicial del juego a un estado meta (estado terminal) ganador.

Formulación como un Problema de Búsqueda

Ajedrez

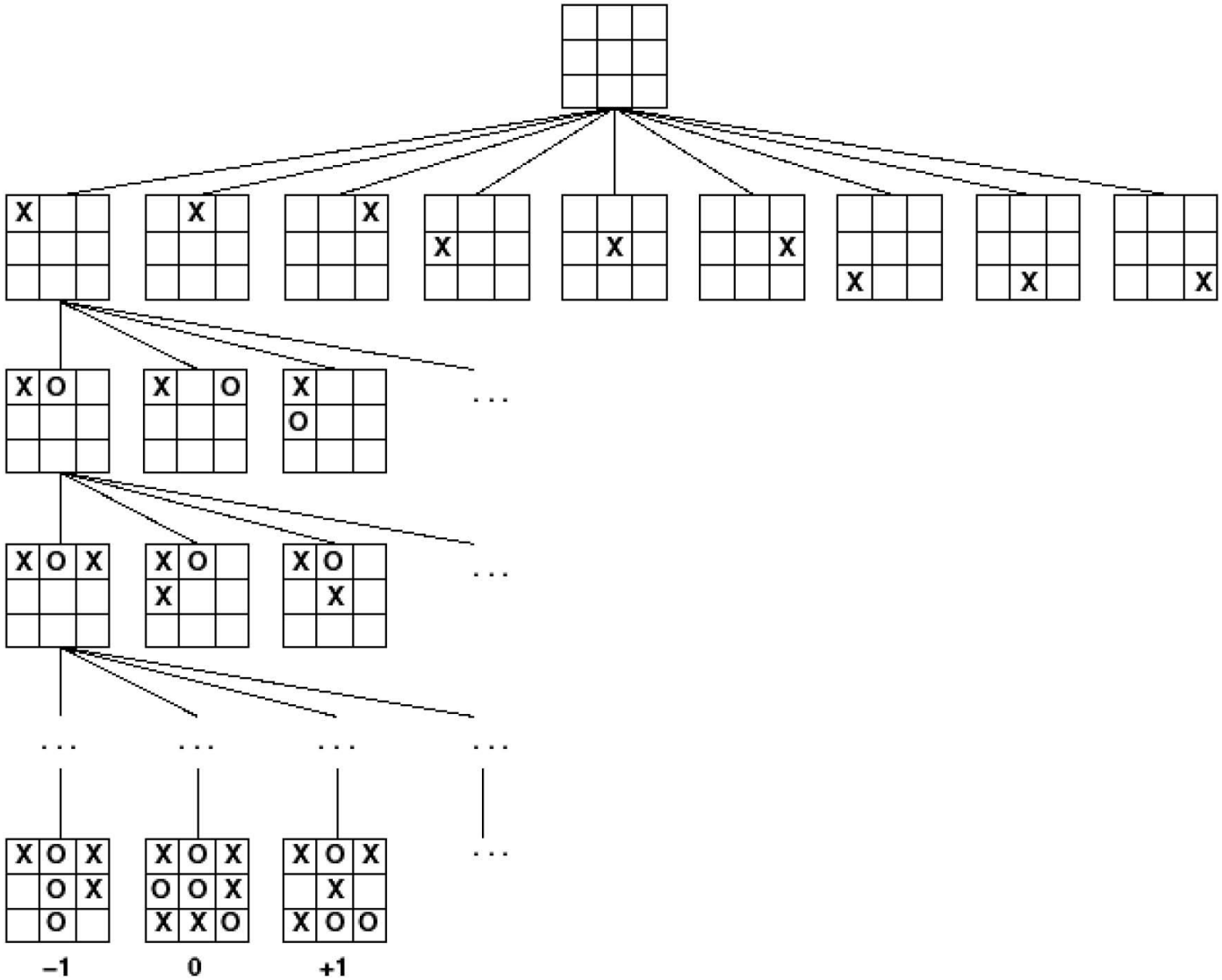
- Estado Inicial
  Tablero con las piezas ubicadas en la posición inicial.
  Jugador que comienza: el que juega con piezas blancas.
- Función sucesor
  Dado un tablero legal y un jugador por jugar, se determinan todos los movimientos legales que ese jugador puede hacer.
- Test de terminación
  Determinamos si es jaque mate o tablas.
- Función utilidad:
  Pueden ser:
  - +1, -1, 0

Formulación como un Problema de Búsqueda

Tic-Tac-Toe

- Estado Inicial
  Tablero 3×3 vacío.
- Función sucesor
  Inicialmente juega uno de los jugadores con X y puede ubicarla en cualquier cuadro. Dado un tablero legal y un jugador por jugar, el tablero resultante es aquel en el que el jugador puso una ficha suya en un cuadrado vacío.
- Test de terminación
  Hay tres fichas iguales en línea o bien el tablero no contiene cuadros vacíos.
- Función utilidad:
  - Gana:+1
  - Pierde: -1
  - Empatan: 0

Espacio de búsqueda



Algoritmo Minimax

Dos jugadores MAX ($\triangle$) y MIN($\triangledown$).

Idea

El jugador MAX juega a la posición de máximo valor, mientras que MIN prefiere el estado de menor valor. Recordar que los valores están dados para MAX, así un valor malo para MAX es bueno para MIN.

Algoritmo Minimax

Idea

El jugador MAX juega a la posición de máximo valor, mientras que MIN prefiere el estado de menor valor. Recordar que los valores están dados para MAX, así un valor malo para MAX es bueno para MIN.

Minimax-Value(n)=

$$\begin{cases} \mathit{Utilidad}(n) & \text{si } n \text{ es un estado terminal} \\ \max_{s \in \mathit{Sucesores}(n)} \text{Minimax-Value(s)} & \text{si } n \text{ es un nodo } \mathit{MAX} \\ \min_{s \in \mathit{Sucesores}(n)} \text{Minimax-Value(s)} & \text{si } n \text{ es un nodo } \mathit{MIN} \end{cases}$$

Algoritmo Minimax

Idea

El jugador MAX juega a la posición de máximo valor, mientras que MIN prefiere el estado de menor valor. Recordar que los valores están dados para MAX, así un valor malo para MAX es bueno para MIN.

Minimax-Value(n)=

$$\begin{cases} \mathit{Utilidad}(n) & \text{si } n \text{ es un estado terminal} \\ \mathit{max}_{s \in \mathit{Sucesores}(n)} \text{Minimax-Value(s)} & \text{si } n \text{ es un nodo } \mathit{MAX} \\ \mathit{min}_{s \in \mathit{Sucesores}(n)} \text{Minimax-Value(s)} & \text{si } n \text{ es un nodo } \mathit{MIN} \end{cases}$$

Algoritmo Minimax

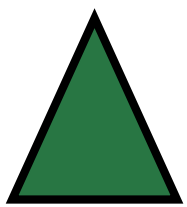
- El recorrido del árbol de búsqueda es Depth-First.
- Calcular el valor de minimax de cada nodo recursivamente

Nodo Max

Nodo Min

Algoritmo Minimax

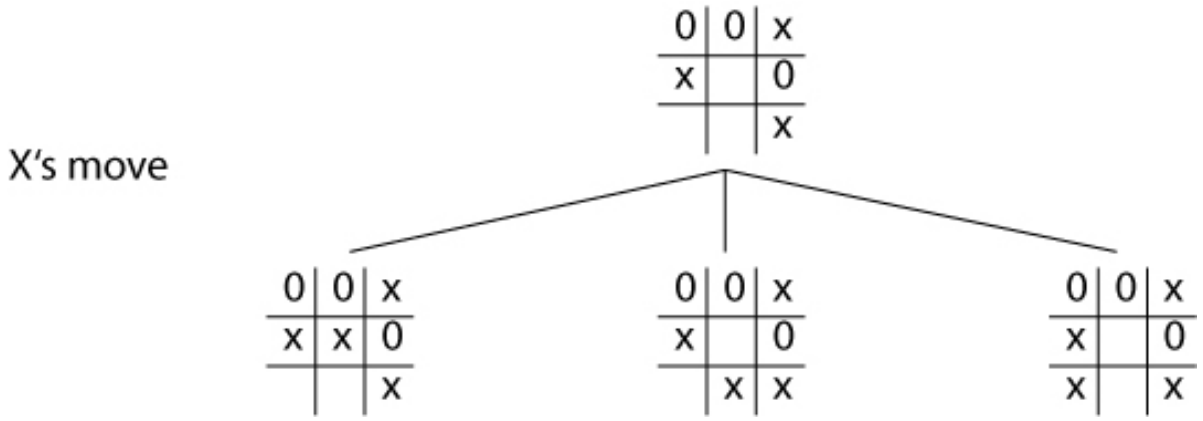
- El recorrido del árbol de búsqueda es Depth-First.
- Calcular el valor de minimax de cada nodo recursivamente



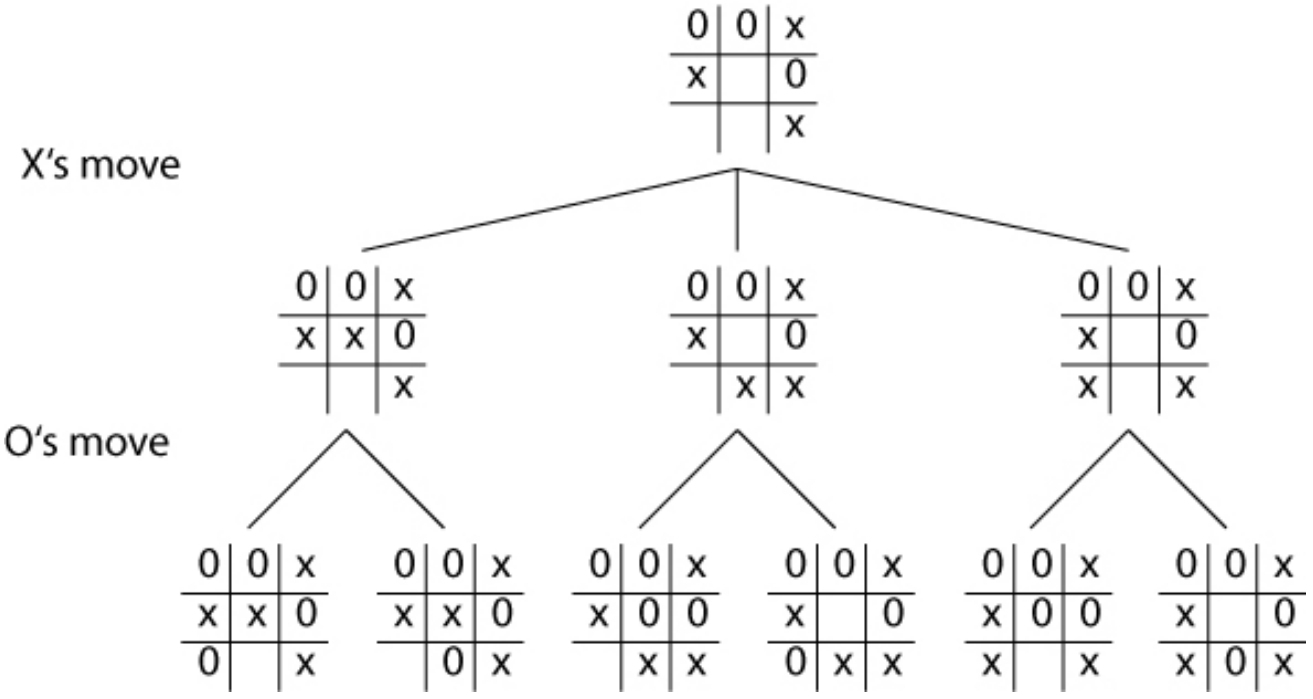
Nodo Max

Nodo Min

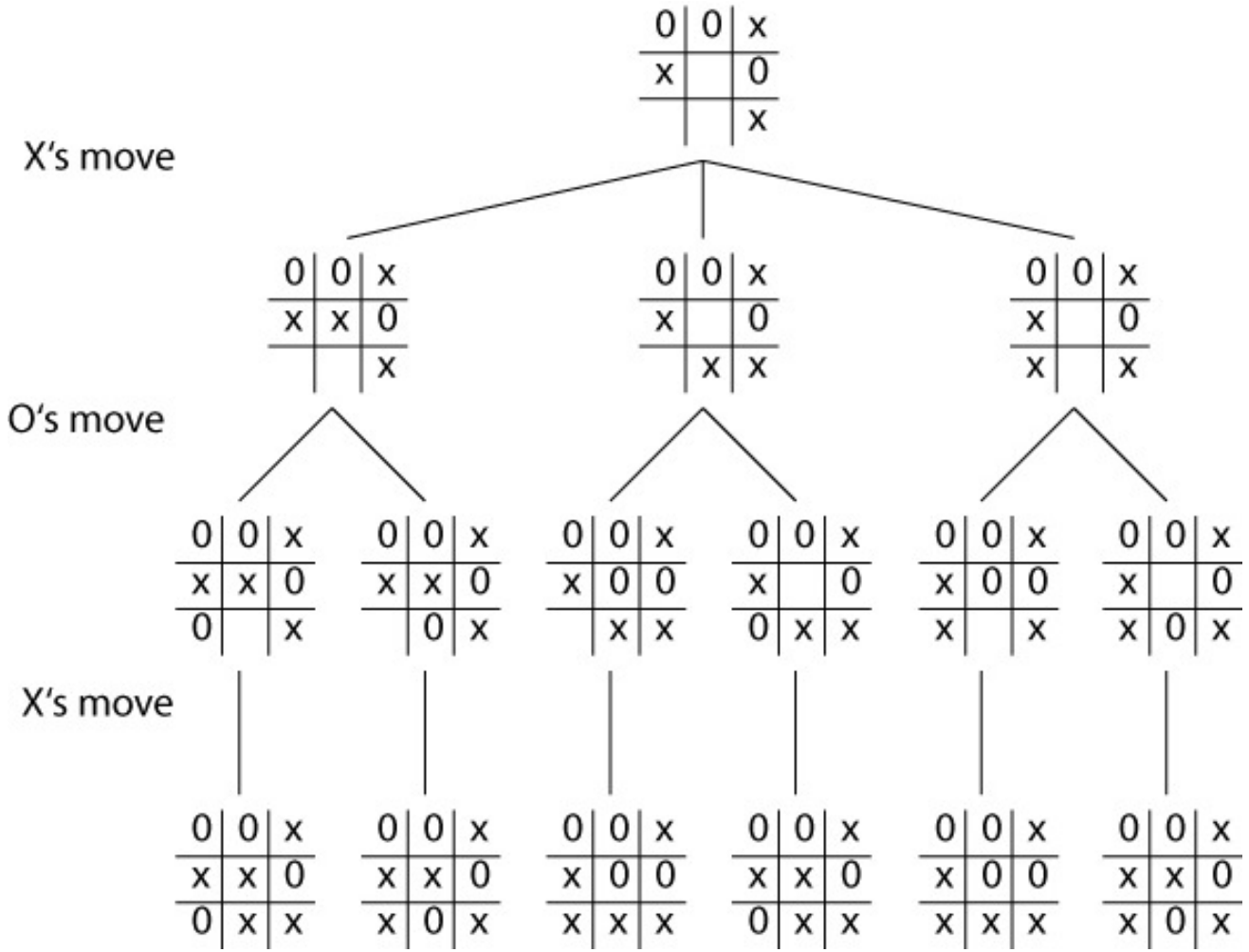
Espacio de búsqueda



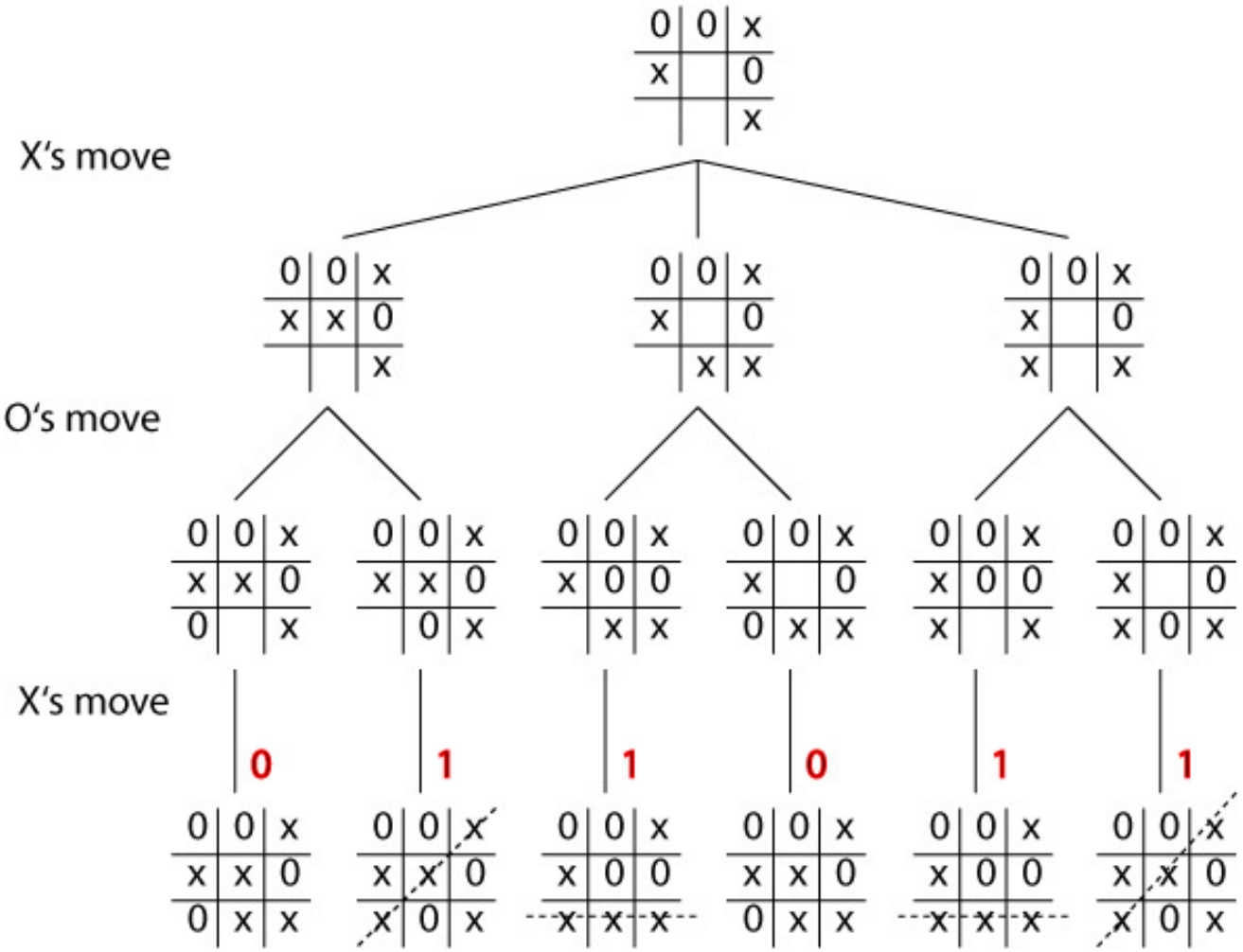
Espacio de búsqueda



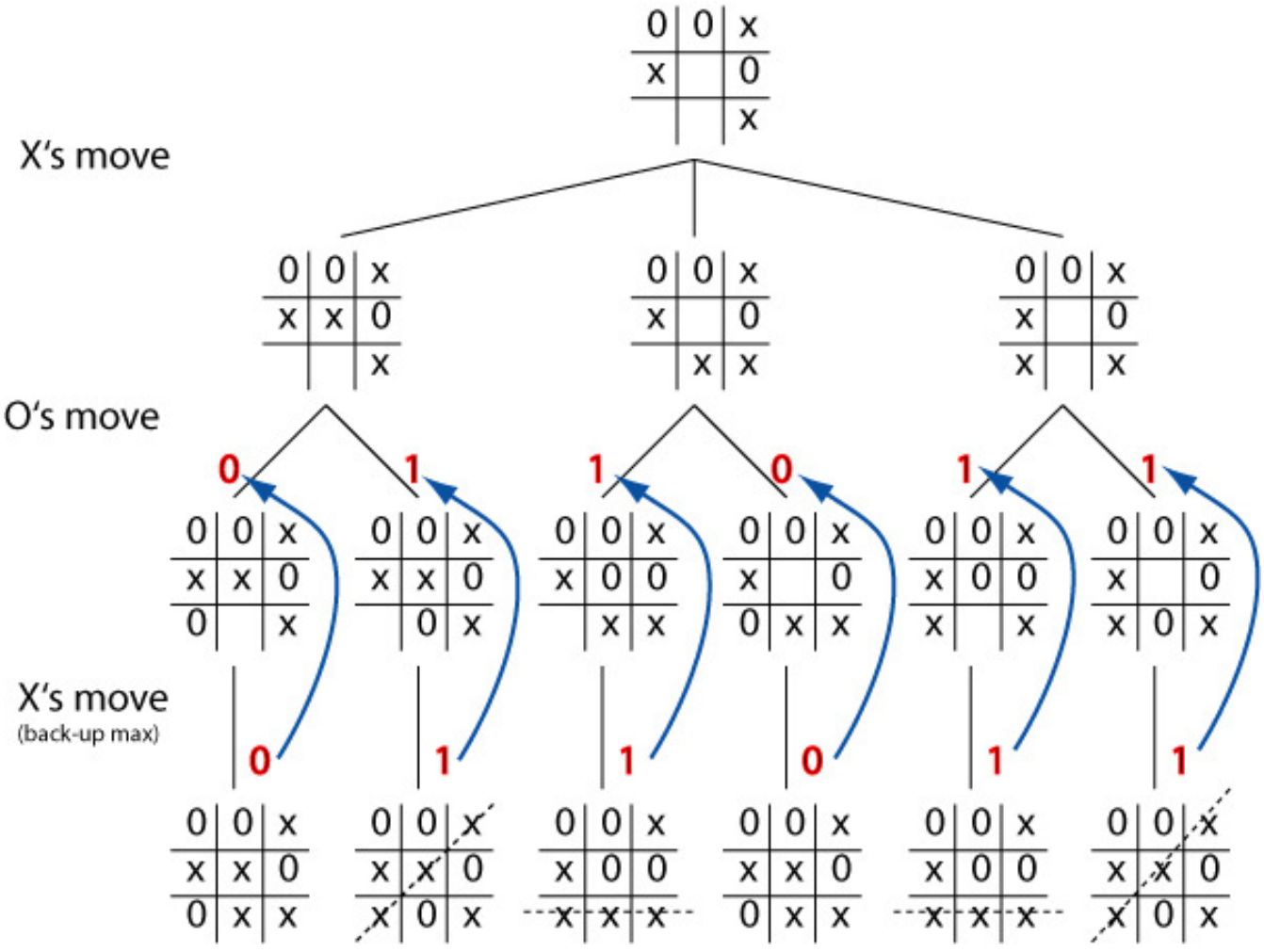
Espacio de búsqueda



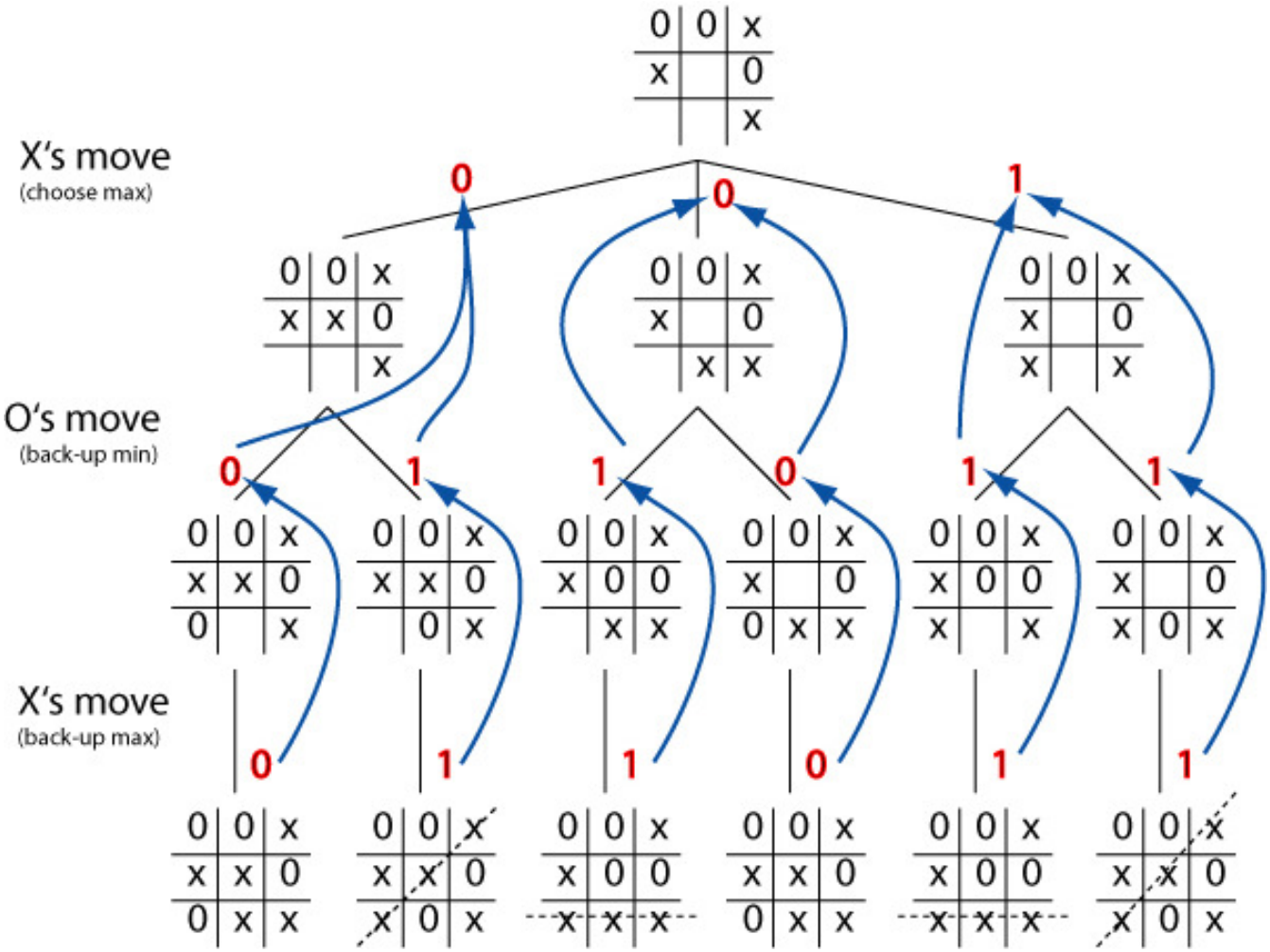
Espacio de búsqueda



Espacio de búsqueda



Espacio de búsqueda



Propiedades de Minimax

Asumimos una profundidad máxima del árbol de m y que el juego tiene b movimientos legales en cada nodo (ramificación). Recordemos que el recorrido es Depth-First.

- ¿Completo? Solo si el árbol es finito.
- ¿Complejidad Temporal? $O(b^m)$.
- ¿Complejidad Espacial? $O(bm)$.
- ¿Óptimo? Si, contra un oponente óptimo.¿De lo contrario?

Propiedades de Minimax

Asumimos una profundidad máxima del árbol de m y que el juego tiene b movimientos legales en cada nodo (ramificación). Recordemos que el recorrido es Depth-First.

- ¿Completo? Solo si el árbol es finito.
- ¿Complejidad Temporal? $O(b^m)$.
- ¿Complejidad Espacial? $O(bm)$.
- ¿Óptimo? Si, contra un oponente óptimo.¿De lo contrario?

Propiedades de Minimax

Asumimos una profundidad máxima del árbol de m y que el juego tiene b movimientos legales en cada nodo (ramificación). Recordemos que el recorrido es Depth-First.

- ¿Completo? Solo si el árbol es finito.
- ¿Complejidad Temporal? $O(b^m)$.
- ¿Complejidad Espacial? $O(bm)$.
- ¿Óptimo? Si, contra un oponente óptimo.¿De lo contrario?

Propiedades de Minimax

Asumimos una profundidad máxima del árbol de m y que el juego tiene b movimientos legales en cada nodo (ramificación). Recordemos que el recorrido es Depth-First.

- ¿Completo? Solo si el árbol es finito.
- ¿Complejidad Temporal? $O(b^m)$.
- ¿Complejidad Espacial? $O(bm)$.
- ¿Óptimo? Si, contra un oponente óptimo.¿De lo contrario?

Propiedades de Minimax

Asumimos una profundidad máxima del árbol de m y que el juego tiene b movimientos legales en cada nodo (ramificación). Recordemos que el recorrido es Depth-First.

- ¿Completo? Solo si el árbol es finito.
- ¿Complejidad Temporal? $O(b^m)$.
- ¿Complejidad Espacial? $O(bm)$.
- ¿Óptimo? Si, contra un oponente óptimo.¿De lo contrario?

Minimax

Para el ajedrez tenemos una profundidad $m \approx 100$ y una ramificación $b \approx 35$.

El tiempo estimado es 35^{100}.
¡HMMMMMMMMMMMM! :(
Solución exacta NO viable completamente con Minimax.

Pensemos...
¿Será necesario explorar todos los caminos?

Minimax

Para el ajedrez tenemos una profundidad $m \approx 100$ y una ramificación $b \approx 35$.

El tiempo estimado es 35^{100}.
¡HMMMMMMMMMMMM! :(
Solución exacta NO viable completamente con Minimax.

Pensemos...
¿Será necesario explorar todos los caminos?

Minimax

Para el ajedrez tenemos una profundidad $m \approx 100$ y una ramificación $b \approx 35$.

El tiempo estimado es 35^{100}.
¡HMMMMMMMMMMMM! :(
Solución exacta NO viable completamente con Minimax.

Pensemos...
¿Será necesario explorar todos los caminos?

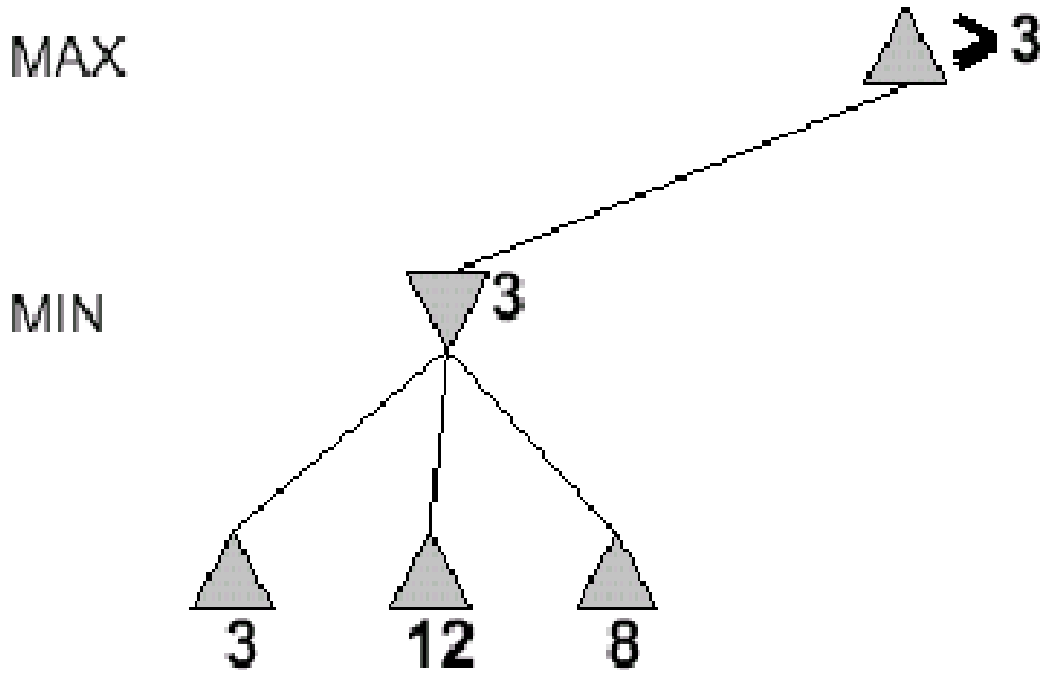
Minimax

Para el ajedrez tenemos una profundidad $m \approx 100$ y una ramificación $b \approx 35$.

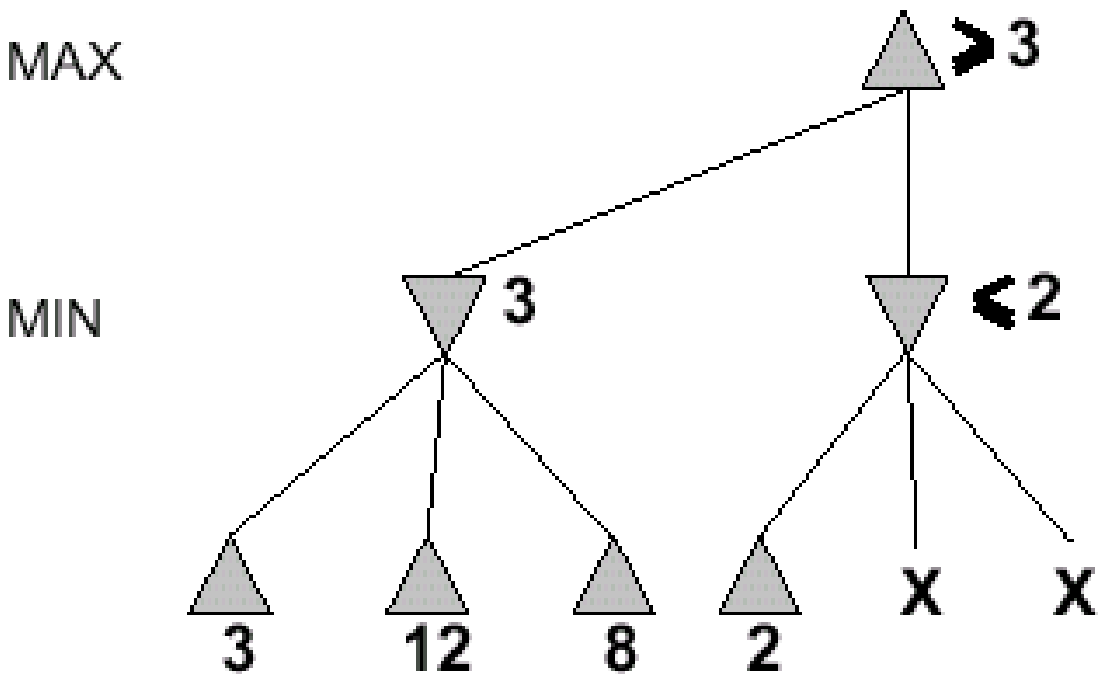
El tiempo estimado es 35^{100}.
¡HMMMMMMMMMMMM! :(
Solución exacta NO viable completamente con Minimax.

Pensemos...
¿Será necesario explorar todos los caminos?

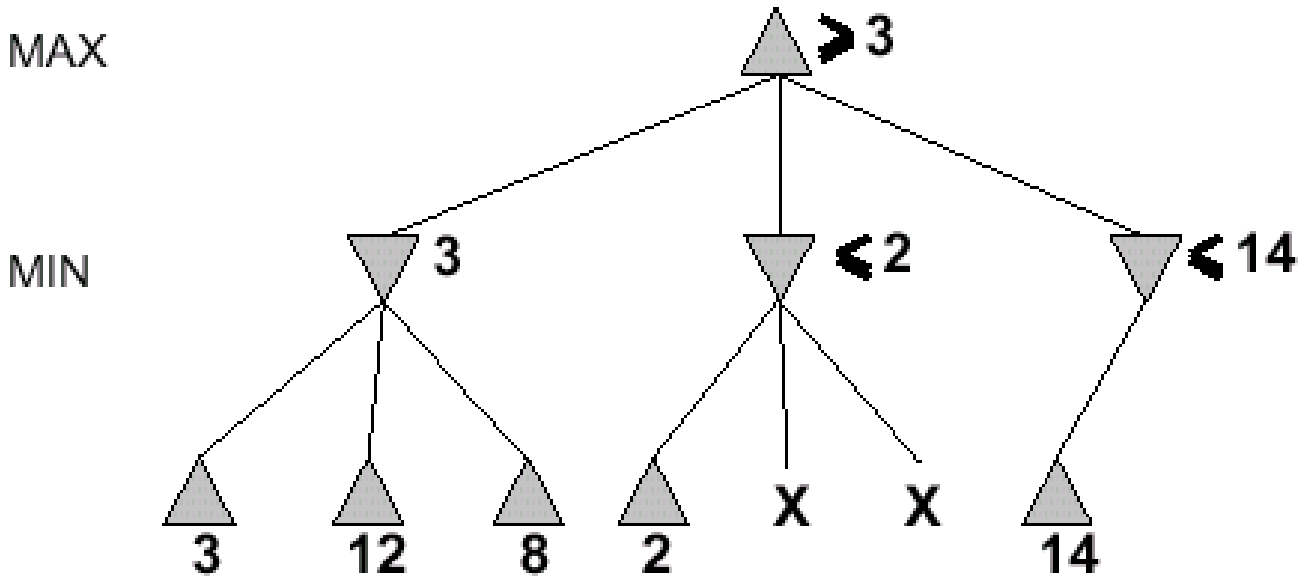
Poda Alpha-Beta



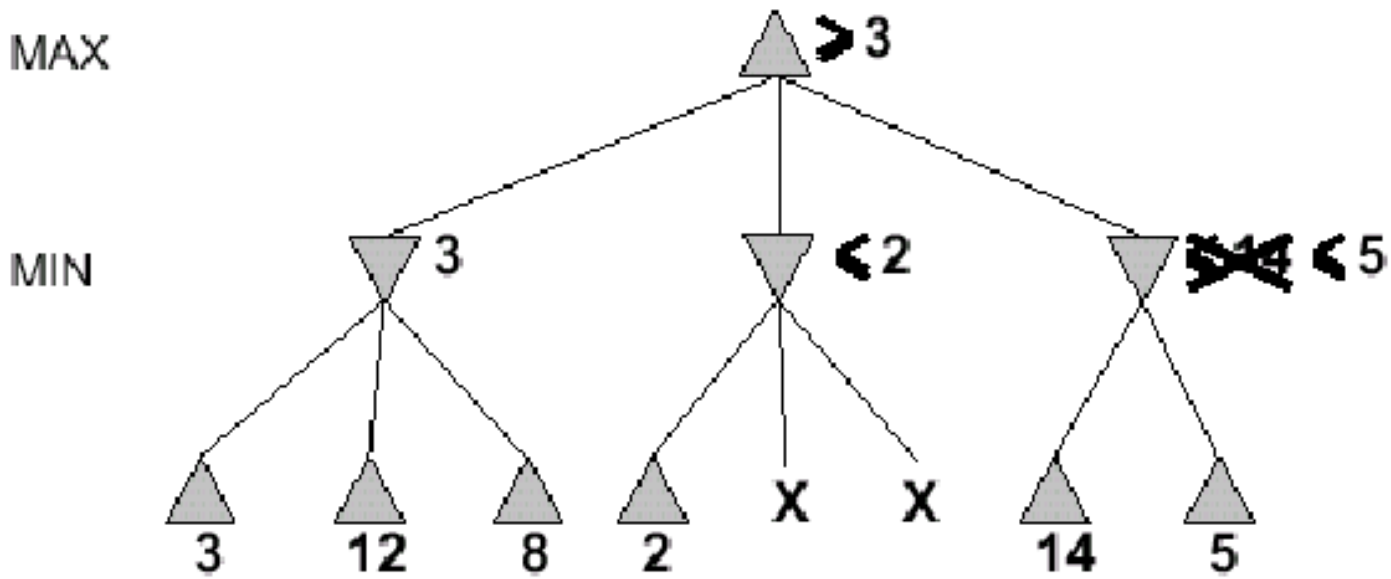
Poda Alpha-Beta



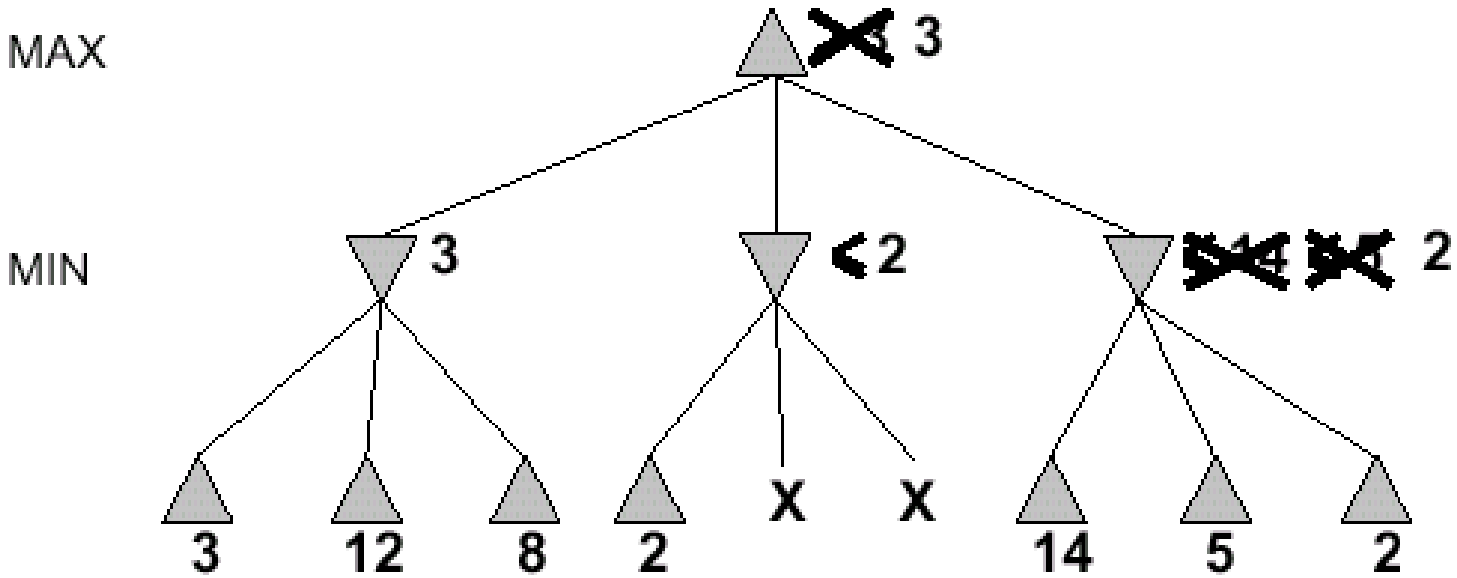
Poda Alpha-Beta



Poda Alpha-Beta



Poda Alpha-Beta



Alpha-Beta

- La poda NO afecta el resultado final.
- Un buen orden de los movimientos mejora la efectividad de la poda.
- Con un “orden perfecto”, la complejidad en tiempo es $O(b^{m/2})$, a diferencia del Minimax $O(b^m)$.

Poda Alpha-Beta

α: es el valor de la mejor elección (valor más alto) que hemos encontrado hasta ahora para MAX. Inicialmente en $-\infty$.

Si estamos analizando un nodo MIN y su valor $V_{Min} \leq \alpha$, PODA, ya que MAX no lo tendrá en cuenta.

β: es el valor de la mejor elección (valor más bajo) que hemos encontrado hasta ahora para MIN. Inicialmente en $+\infty$.

Si estamos analizando un nodo MAX y valor $V_{Max} \geq \beta$, PODA, ya que MIN no lo tendrá en cuenta.

Algoritmo $\alpha - \beta$

```
function ALPHA-BETA-DECISION(state) returns an action
return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

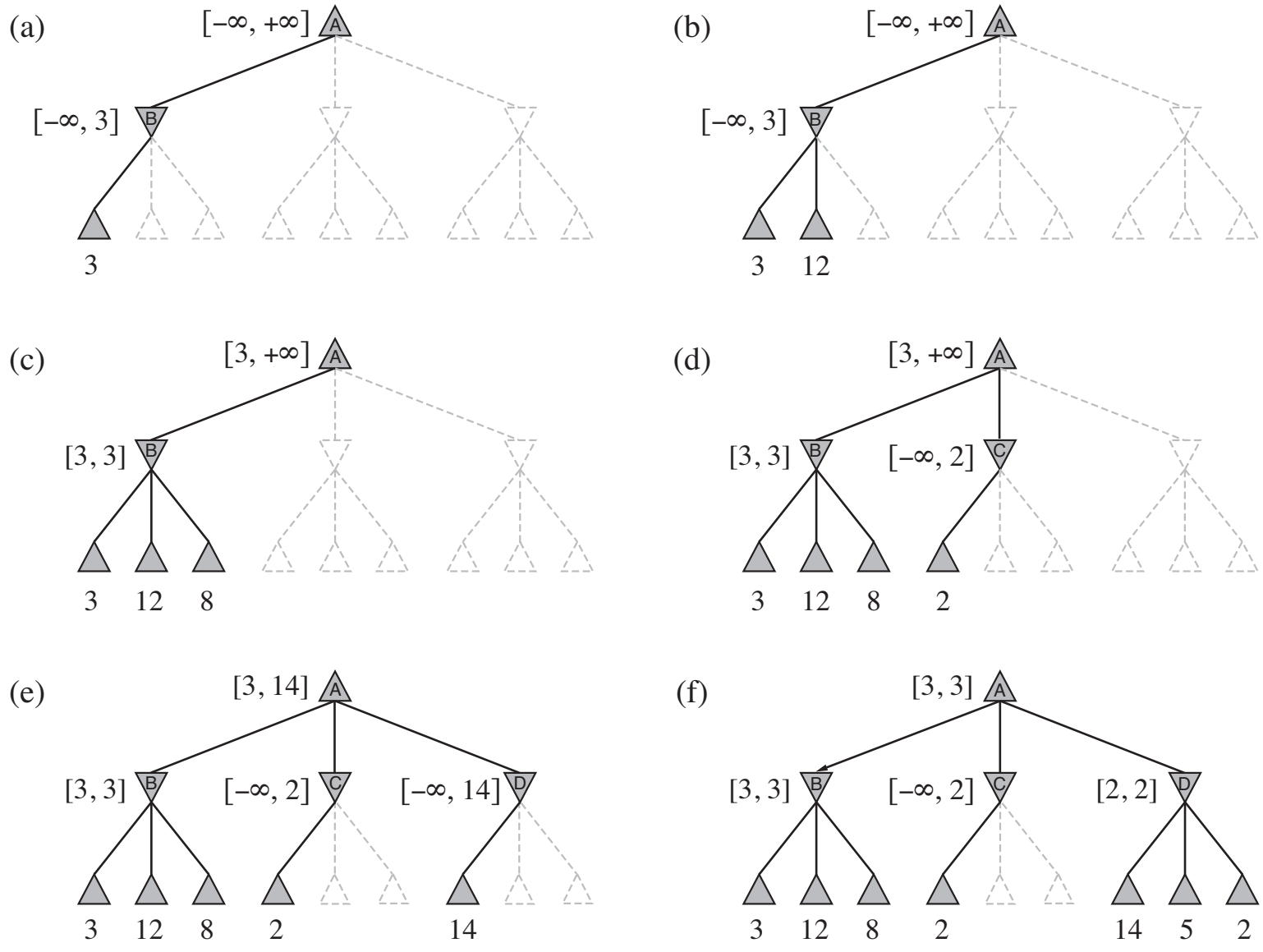
```
function MAX-VALUE(state, α, β) returns a utility value
entradas: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)

v ← −∞
for a, s in SUCCESSORS(state) do
    v ← MAX(v,MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
entradas: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) thenreturnUTILITY(state)

v ← +∞
for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
```

Ejemplo


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

Efectividad de la poda $\alpha - \beta$

- Alfa-Beta garantiza calcular el mismo valor para el nodo raíz que el calculado por el algoritmo Minimax.
- Peor caso: NO poda, el examen sobre los nodos hojas es del O(b^m)
- Mejor caso: el examen de los nodos hojas es del O($b^{d/2}$)
- Mejor caso es cuando la mejor jugada de cada jugador es la alternativa más a la izquierda.
- En Deep Blue, encontraron empíricamente que con la poda alfa-beta el factor de ramificación de la media en cada nodo es de 6 en vez de unos 35-40.

Toma de decisiones en tiempo real

- En un juego, el tiempo para “pensar” la siguiente movida es limitado.
- Las personas no deseamos esperar mucho tiempo para que nuestro contrincante realice su movida.

Solución

- Cortar el recorrido del espacio de juego antes de llegar a la hojas, convirtiéndo ese nodo interno en hoja y utilizar una función de evaluación (medida heurística de utilidad) del nodo.
- Implementar un test de corte (Cutoff test) que nos determina cuando usamos la función de evaluación.

Función de Evaluación

Valor estimado de la utilidad esperada del juego desde un nodo

La performance del juego dependerá de la calidad de la función de evaluación. Una función de evaluación imprecisa guiará al agente a una posición perdedora.

Características de diseño:

- Coincida con la función de utilidad en los nodos terminales.
- Su cálculo no sea costoso ni en tiempo ni espacio.
- Refleje de la forma más precisa las chances de ganar por ese camino.

Función de Evaluación

¿Cómo diseñamos una función de evaluación?

El diseño de la función depende de la identificación de características que distinguen a los estados.

Por ejemplo: en el ajedrez podría ser número de piezas blancas y negras, si tiene la reina blanca, si tiene la reina negra, etc.

Función de Evaluación

¿Cómo diseñamos una función de evaluación?

El diseño de la función depende de la identificación de características que distinguen a los estados.

Por ejemplo: en el ajedrez podría ser número de piezas blancas y negras, si tiene la reina blanca, si tiene la reina negra, etc.

Función de Evaluación

Valor

Esperado Podemos definir clases de equivalencia de estados. Cada clase tiene una probabilidad de ganar, perder, o empatar.

Ej. 60% gana, 30% pierde, 10% empata
Se usa para crear el valor esperado o peso promedio de la utilidad

$$(0.6 * 1) + (0.3 * -1) + (0.1 * 0) = 0.3$$

Material Le damos pesos a ciertas características.

$$E(s) = \sum_i w_i \cdot f_i(s)$$

Función de Evaluación

Valor

Esperado Podemos definir clases de equivalencia de estados. Cada clase tiene una probabilidad de ganar, perder, o empatar.

Ej. 60% gana, 30% pierde, 10% empata
Se usa para crear el valor esperado o peso promedio de la utilidad

$$(0.6 * 1) + (0.3 * -1) + (0.1 * 0) = 0.3$$

Material Le damos pesos a ciertas características.

$$E(s) = \sum_i w_i \cdot f_i(s)$$

Función de Evaluación

Valor

Esperado Podemos definir clases de equivalencia de estados. Cada clase tiene una probabilidad de ganar, perder, o empatar.

Ej. 60% gana, 30% pierde, 10% empata
Se usa para crear el valor esperado o peso promedio de la utilidad

$$(0.6 * 1) + (0.3 * -1) + (0.1 * 0) = 0.3$$

Material Le damos pesos a ciertas características.

$$E(s) = \sum_i w_i \cdot f_i(s)$$

Test de Corte

Enfoque Simple

Límite de profundidad para la búsqueda

```
Si CUTOFF-TEST(state, depth) entonces return
EVAL(state)
```

Problemas

- El corte podría ser aplicado a nodos que parecieran ser prometedores para un jugador A aunque en la siguiente jugada, quedara claramente mostrado que es perdedor para el jugador A.
- Puede detenerse antes que el tiempo disponible sea usado.

Se requiere de alguna mejora...

Test de Corte

Enfoque Simple

Límite de profundidad para la búsqueda

```
Si CUTOFF-TEST(state, depth) entonces return
EVAL(state)
```

Problemas

- El corte podría ser aplicado a nodos que parecieran ser prometedores para un jugador A aunque en la siguiente jugada, quedara claramente mostrado que es perdedor para el jugador A.
- Puede detenerse antes que el tiempo disponible sea usado.

Se requiere de alguna mejora...

Test de Corte

Problemas

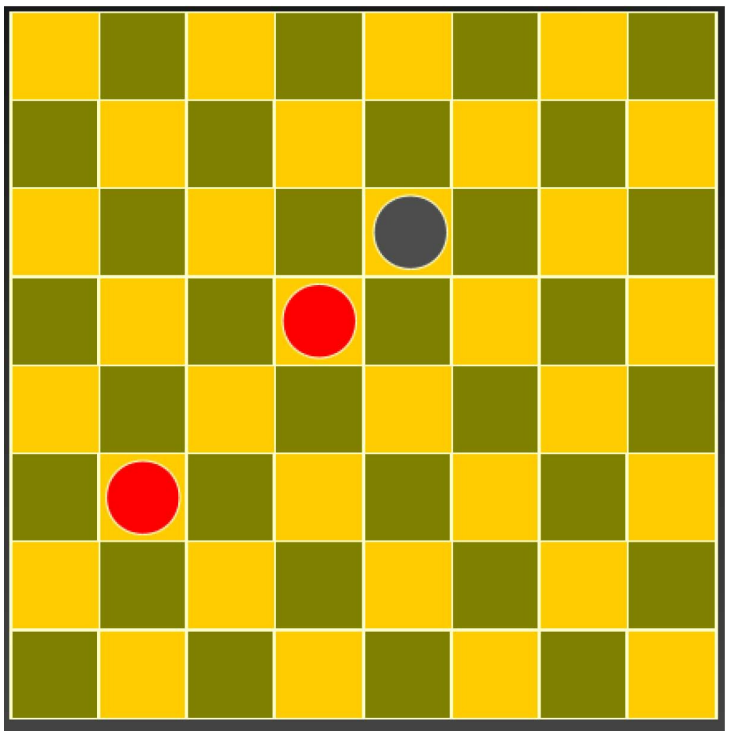
- El corte podría ser aplicado a nodos que parecieran ser prometedores para un jugador A aunque en la siguiente jugada, quedara claramente mostrado que es perdedor para el jugador A.
- Puede detenerse antes que el tiempo disponible sea usado.

Búsqueda en Reposo

El corte se aplica sólo a nodos en reposo, es decir a aquellos que tienen poca probabilidad de oscilar bruscamente, según la función de evaluación, en el futuro cercano.

Quiescence - Reposo

DAMA



Supongamos que la función de evaluación captura sólo el número de piezas de cada color.

Con esta función de evaluación: Rojo tiene ventaja.

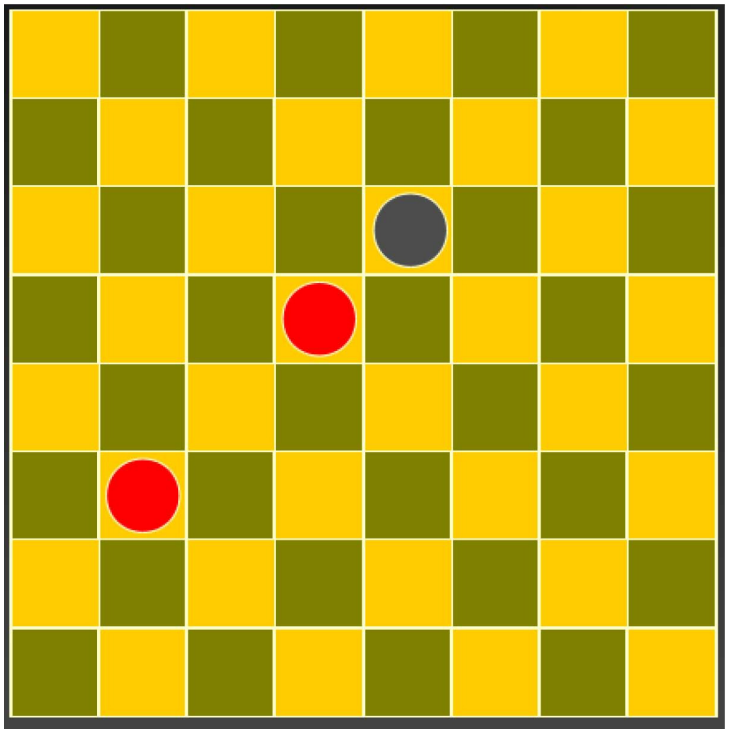
Sin embargo, ignora el hecho de que las negras puede saltar doble para ganar.

Solución: Sólo aplique el corte en posiciones en reposo. Esto es no hay grandes cambios entre un nodo y el siguiente.

En nuestro ejemplo, aplique el corte si las negras no tienen saltos permitidos que coman muchas piezas rojas.

Quiescence - Reposo

DAMA



Supongamos que la función de evaluación captura sólo el número de piezas de cada color.

Con esta función de evaluación: Rojo tiene ventaja.

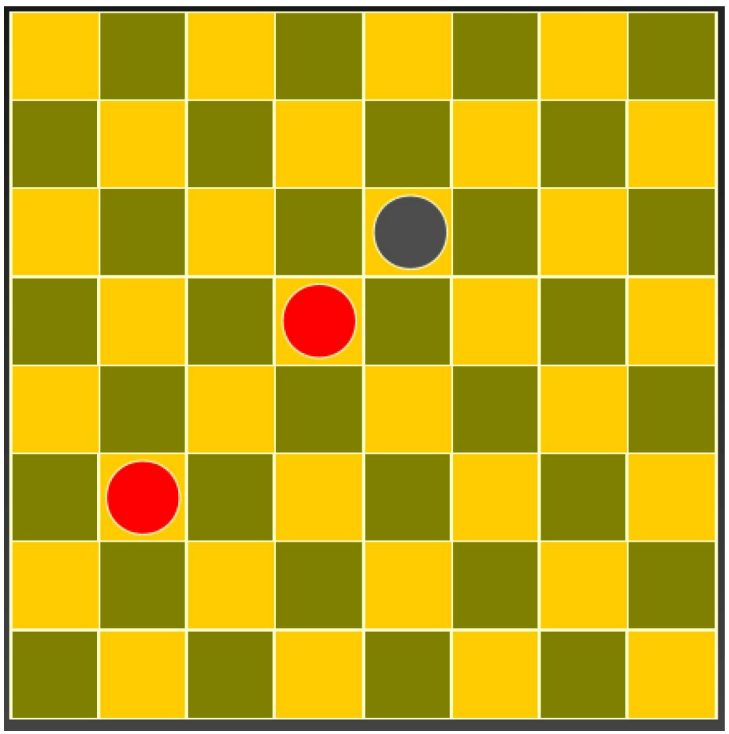
Sin embargo, ignora el hecho de que las negras puede saltar doble para ganar.

Solución: Sólo aplique el corte en posiciones en reposo. Esto es no hay grandes cambios entre un nodo y el siguiente.

En nuestro ejemplo, aplique el corte si las negras no tienen saltos permitidos que coman muchas piezas rojas.

Quiescence - Reposo

DAMA



Supongamos que la función de evaluación captura sólo el número de piezas de cada color.

Con esta función de evaluación: Rojo tiene ventaja.

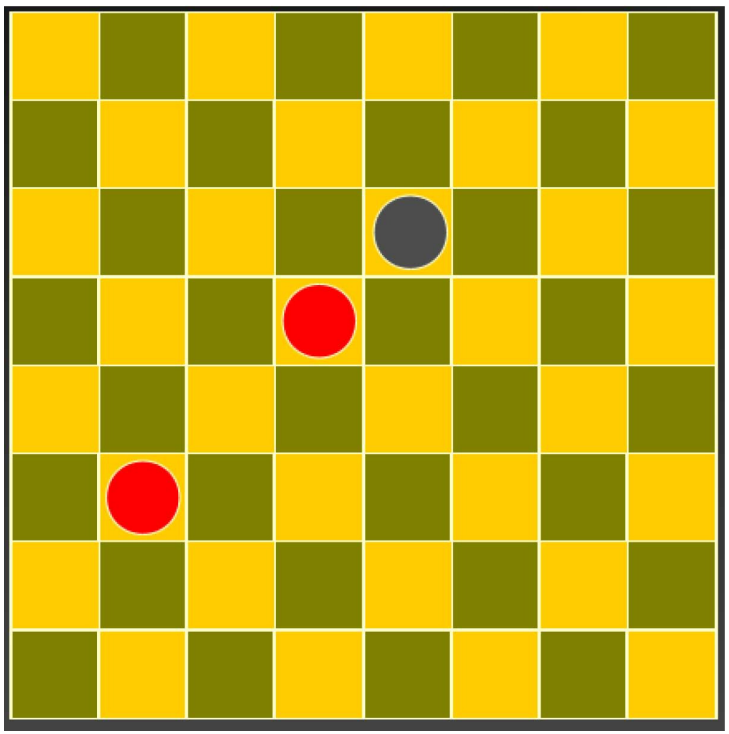
Sin embargo, ignora el hecho de que las negras puede saltar doble para ganar.

Solución: Sólo aplique el corte en posiciones en reposo. Esto es no hay grandes cambios entre un nodo y el siguiente.

En nuestro ejemplo, aplique el corte si las negras no tienen saltos permitidos que coman muchas piezas rojas.

Quiescence - Reposo

DAMA



Supongamos que la función de evaluación captura sólo el número de piezas de cada color.

Con esta función de evaluación: Rojo tiene ventaja.

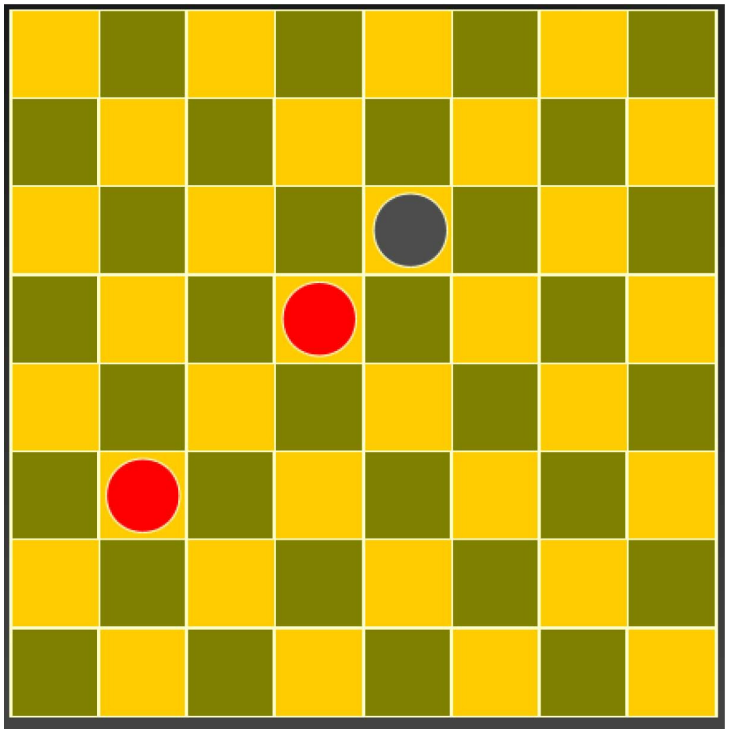
Sin embargo, ignora el hecho de que las negras puede saltar doble para ganar.

Solución: Sólo aplique el corte en posiciones en reposo. Esto es no hay grandes cambios entre un nodo y el siguiente.

En nuestro ejemplo, aplique el corte si las negras no tienen saltos permitidos que coman muchas piezas rojas.

Quiescence - Reposo

DAMA



Supongamos que la función de evaluación captura sólo el número de piezas de cada color.

Con esta función de evaluación: Rojo tiene ventaja.

Sin embargo, ignora el hecho de que las negras puede saltar doble para ganar.

Solución: Sólo aplique el corte en posiciones en reposo. Esto es no hay grandes cambios entre un nodo y el siguiente.

En nuestro ejemplo, aplique el corte si las negras no tienen saltos permitidos que coman muchas piezas rojas.

Efecto Horizonte

Idea

Ocurre cuando los movimientos del oponente causan un daño serio que es inevitable, pero que puede ser evitado temporalmente retrasado la táctica.

Poda hacia Adelante

Idea

Algunos nodos son podados inmediatamente sin ninguna consideración futura.

Estrategia: Búsqueda Dirigida

En cada ply considere los mejores n movimientos, en vez de considerar a todos los movimientos.

Esta estrategia es peligrosa, ya que no garantiza que el mejor movimiento no sea podado.

Juegos con azar

Consideramos a los juegos como estocásticos cuando:

- Contienen elementos que hacen a su comportamiento impredecible. Por ejemplo, los dados
- Información imperfecta: Cartas, dominó, etc.

¿Cómo representamos el árbol de un juego estocástico?

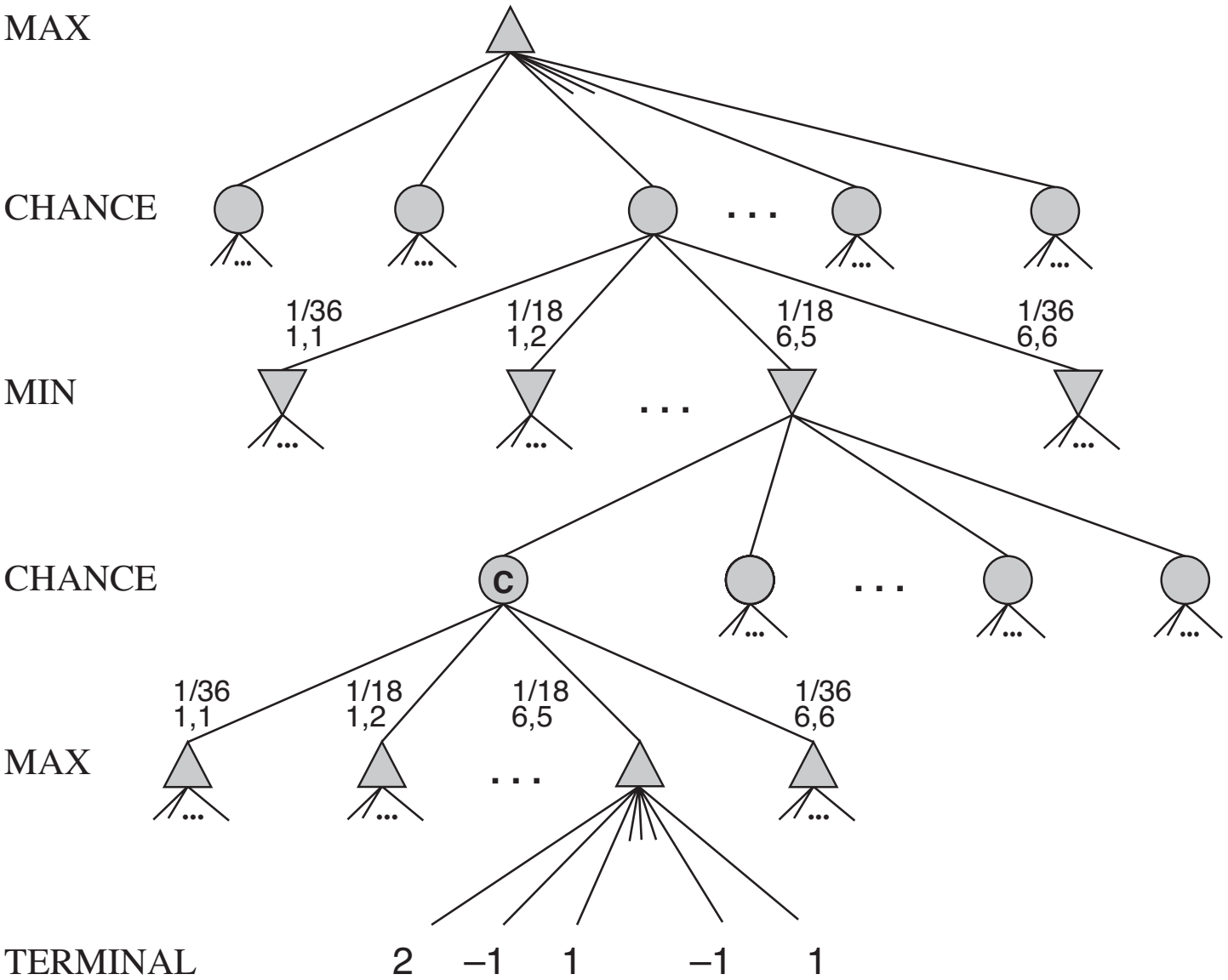
Juegos con azar

Consideramos a los juegos como estocásticos cuando:

- Contienen elementos que hacen a su comportamiento impredecible. Por ejemplo, los dados
- Información imperfecta: Cartas, dominó, etc.

¿Cómo representamos el árbol de un juego estocástico?

Nodos de azar



Valor del Expectminimax

Extiende la idea del minimax

$$Expectiminimax(s) = \begin{cases} Utility(s) & \text{si } Test - Terminal(s) \\ max_a Expectiminimax(Result(s,a)) & \text{si } Jugador(s) = Max \\ min_a Expectiminimax(Result(s,a)) & \text{si } Jugador(s) = Min \\ \sum_r P(s) \cdot Expectiminimax(Result(s,a) & \text{si } Jugador(s) = Chance \end{cases}$$

P(s) es la probabilidad de que un evento random ocurra (ej, dado) y Result(s,a) es el estado resultante desde s dado que ocurrió el elemento de azar r (por ejemplo, se arrojaron los dados y salió r).

Resumen

- Algunos juegos pueden ser representados como problemas de búsquedas
  - Arbol de juego
- Algoritmo Minimax
  - Asume que el oponente juega de manera óptima
  - Definiendo la función de utilidad
- Las podas pueden reducir el espacio de búsqueda drásticamente

Resumen

- Los juego en tiempo real requiere cutoffs
- Necesita definir una medida de utilidad heurística: función de evaluación
  - Función de evaluación puede ser derivada de simulaciones, análisis, experiencias
- Juegos con azar
  - Requiere nodo de azar en el árbol de juego
  - Valor de utilidad del estado estimada

Expectminimax

Complejidad temporal

- O($b^m n^m$), donde n is the número de eventos distintos
- Puede extenderse $\alpha - \beta$.


GAME OVER
INSERT COIN

Desafíos de la clase anterior

- Analice el problema de los misioneros y caníbales y formúlelo como un problema de búsqueda. ¿Qué algoritmo de los vistos se adapta mejor a este problema? Analice el espacio de búsqueda.
- Investigue sobre el problema de las 8 reinas. Formúlelo como un problema de búsqueda y determine qué algoritmo de búsqueda ciega se adapta mejor a este problema.
- Ejemplo de A^* con heurística sobreestimada. Muestre que A^* se comporta en forma errónea.
- Muestre un ejemplo en el que el problema de las 4-reinas entre en un risco o borde y que hill-climbing no resuelva en forma adecuada.
- Muestre un ejemplo de planicie y uno de risco para 8-puzzle.

¿Preguntas?