Capítulo 3

Juegos

3.1 Introducción

La habilidad de jugar es considerada como una distinción de inteligencia.

Características:

- Fácil de crear situaciones complicadas con reglas sencillas.
- Se pueden probar contra humanos en donde existen escalas.
- Son adictivos.

Casi todos: 2 jugadores + *información perfecta*, aunque existen otros, por ejemplo: barajas, backgammon, etc.

Con información perfecta: las reglas del juego determinan los posibles movimientos.

A diferencia de búsqueda, el oponente introduce incertidumbre porque no sabemos que va a tirar, lo cuál se asemeja más a problemas reales.

En un juego tenemos:

- Posición inicial.
- Conjunto de operadores (definen movidas legales).
- Estado terminal.
- Función de utilidad, e.g., gana, pierde, empata (pero puede haber más, por ejemplo en backgammon).

Los árboles de juegos tienen correspondencia con árboles AND–OR (mis tiradas son OR, las del oponente son AND).

Se require de una estrategia que garantice llegar a estados terminales ganadores independientemente de lo que haga el oponente.

Los juegos, y el ajedrez en particular, han sido objeto de estudio por muchos años por los investigadores de Inteligencia Artificial.

3.2 MiniMax

Shanon lo sugirió originalmente (50), basado en teoría de juegos de von Neumann y O. Morgenstern, aunque Turing presentó el primer programa (51).

Idea: Maximizar mis tiradas considerando que el oponente va a minimizar.

Para decidir qué jugada hacer, el árbol se divide por niveles:

- *Max*: el primer jugador (nivel) y todas las posiciones (niveles) donde juega.
- *Min*: el oponente y todas las posiciones en donde juega

Las hojas se etiquetan con *gana, pierde*, *empata* desde el punto de vista de *max*.

Si podemos etiquetar todas las hojas, podemos etiquetar todo el árbol.

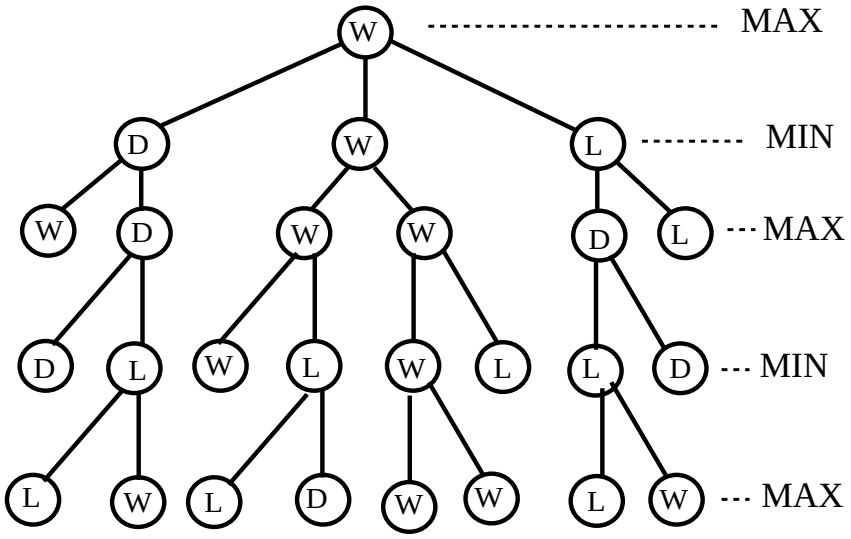

Figura 3.1: Etiquetamiento de un árbol usando *minimax*.

La siguiente función calcula lo mejor que *max* puede esperar desde la posición J si juega de manera óptima contra un oponente perfecto.

Si J es nodo *max* no–terminal:

$$etiq(J) = \begin{cases} gana & \text{si algún sucesor de } J \text{ es } gana \\ pierde & \text{si todos los sucesores son } pierde \\ empata & \text{si alguno } empata \text{ y ninguno } gana \end{cases}$$

Si J es nodo *min* no–terminal:

$$etiq(J) = \begin{cases} gana & \text{si todos los sucesores de } J \text{ son } gana \\ pierde & \text{si algún sucesor de } J \text{ es } pierde \\ empata & \text{si alguno } empata \text{ y ninguno } pierde \end{cases}$$

Una estrategia para un jugador es un árbol considerando un camino para cada nodo del jugador y todos los posibles para el oponente.

La intersección de las dos estrategias define un juego.

Desde el punto de vista de *max*:

- *min* tira hacia la estrategia menos favorable: $min(estrategias)$
- *max* trata de maximiar ésto: $max(min(estrategias))$

Si cambiamos los roles (ahora tira *min*): $min(max(estrategias))$.

Minimax propaga asumiendo que las estimaciones son correctas. Si la estimación es burda, la propagación también va a ser burda.

Estrategia alternativa:

Negmax: igual a *minimax*, pero la función de evaluación es simétrica, por lo que:

$$etiq(J) = \begin{cases} gana & \text{si algún sucesor de } J \text{ es } pierde \\ pierde & \text{si todos los sucesores de } J \text{ son } gana \\ empata & \text{cualquier otra situación} \end{cases}$$

$$status(J) = max_{J'}\{-status(J') \mid J' \text{ es hijo de } J\}$$

Lo que nos interesa son estrategias de juegos ganadoras sin importar lo que haga el oponente (*min*).

3.2.1 Tamaño de Búsqueda

Una de las motivaciones principales de investigación en juegos es que son difíciles de resolver.

Ajedrez: *branching factor* ≈ 35, número de jugadas por jugador ≈ 50, por lo que hay $\approx 35^{100}$ o 10^{120} hojas o nodos terminales (aunque *sólo* hay 10^{40} posiciones diferentes legales).

En en caso de Damas Chinas es: 10^{40}

Como no se puede evaluar/explorar todo el árbol, exploramos hasta cierta profundidad y usamos heurísticas (una función de evaluación *estática*).

Suposición: evaluando después de cierta exploración es mejor que evaluar sin exploración, bajo el supuesto que el mérito de una situación se clarifica al ir explorando.

La calidad del programa depende fundamentalmente de la función de evaluación. Tiene que ser (i) congruente con la función de utilidad de los nodos terminales, (ii) rápida y (iii) reflejar las posibilidades actuales de ganar.

Una valor de la función de evaluación en realidad cubre muchas posibles posiciones.

Si fuera perfecta, no tendríamos que hacer búsqueda.

El algoritmo *minimax* usa dos heurísticas: (i) cálculo de la función de evaluación y (ii) propagar valores hacia atrás asumiendo que los valores en los nodos de frontera son correctos.

Si las estimaciones son burdas, también son los resultados.

Muchas de las funciones de evaluación asumen independencia entre los factores y las expresan como funciones lineales con pesos ($w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$).

Para ésto, hay que encontrar los pesos y decidir qué factores a considerar.

Algunas estrategias de evaluación (tomadas de Deep–Blue antes Deep–Thought):

- Material: peón = 1, caballos y alfiles = 3, torre = 5, y reina = 9. Puede variar dependiendo de la situación (e.g., el mantener un par de alfiles al final del juego).
- Posición: Al principio se pensó que el control del centro era lo primordial. Aunque si es muy importante no es lo único. Una forma fácil de imaginarse "posición" es contanto el número de cuadros que se pueden atacar en forma segura. Entre más cuadros se tengan, se tiene más control.
- Defensa: Los aspectos defensivos de la posición tienen que ver con la

Tabla 3.1: Algoritmo Minimax.

Para determinar el valor *minimax* de J: $V(J)$

si J es terminal, $V(J) \leftarrow ev(J)$
sino genera los sucesores de $J : J_1, J_2, \ldots, J_n$
evalua $V(J_1), V(J_2), \ldots, V(J_n)$ de izquierda a derecha
si J es nodo *max*, $V(J) \leftarrow max[V(J_1), \ldots, V(J_n)]$
si J es nodo *min*, $V(J) \leftarrow min[V(J_1), \ldots, V(J_n)]$

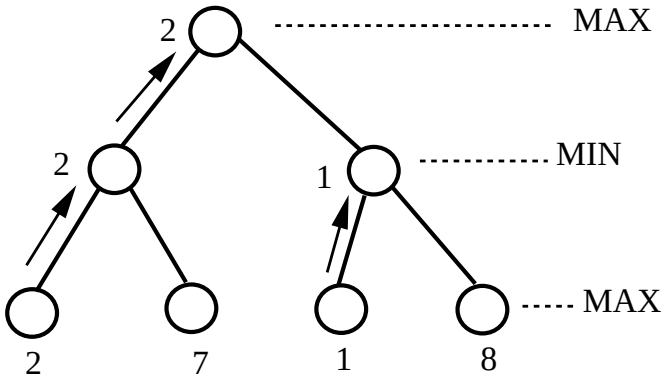

Figura 3.2: Ejemplo sencillo de la estrategia *minimax*.

seguridad del rey.

- Tempo: define la "carrera" por el control del tablero.

3.2.2 Algoritmo Minimax

La mayoría de los programas usan variantes de *minimax*: (i) generar árbol hasta cierta profundidad y (ii) evaluar posiciones en nodos de frontera.

El esfuerzo (dada una función de evaluación) es proporcional al número de nodos en la frontera que son evaluados.

Tabla 3.2: Algoritmo Minimax con *backtracking*.

Algoritmo *minimax* (*backtracking*)
si J es terminal, $V(J) \leftarrow ev(J)$, *sino*
for $k = 1, \ldots, n$ *do*
    generar J_k (el k–esimo sucesor de J)
    evalua $V(J_k)$
    si $k = 1, VA(J) \leftarrow V(J_1)$ *sino*
    para $K \geq 2$
        si J es nodo *max*, $VA(J) \leftarrow max[VA(J), V(J_k)]$
        si J es nodo *min*, $VA(J) \leftarrow min[VA(J), V(J_k)]$
regresa $V(J) \leftarrow VA(J)$

3.2.3 Alternativas

1. No tenemos que generar todos los sucesores y guardar sus valores hasta que todos sean evaluados. Podemos usar un algoritmo *depth–first* (ver tabla 3.2).

2. No es necesario que evaluemos todos los nodos terminales (ver tabla 3.3).

Las mejoras dependen del orden en que los sucesores son evaluados.

Una de las razones de la eficiencia del último algoritmo es que se da cuenta que el estatus de algunos nodos no afecta la evaluación del nodo raíz.

3.3 Alpha–Beta

A McCarthy (56) se le considera el responsable de ver la utilidad de *alpha–beta*.

Parecido a *dynamic–programming* en el sentido de que elimina caminos que no van a producir mejores resultados.

Es el más usado en juegos.

Tabla 3.3: Algoritmo Minimax con *pierde/gana*.

Algortimo mejorado (ilustrado con *gana–pierde*)
si J es terminal, regresa, *gana/pierde*
empieza a etiquetar los sucesores de J (de izq. a der.)
si J es *max*, regresa *gana* en cuanto se encuentre un
sucesor con *gana* o regresa *pierde* si todos
los sucesores de J son *pierde*
si J es *min*, regresa *pierde* en cuanto se encuentre un
sucesor con *pierde* o regresa *gana* si todos
los sucesores de J son *gana*

Es como un *minimax* haciendo backtracking, pero ...

> si al actualizar el valor *minimax* de un nodo, éste cruza cierto límite, entonces no hace falta hacer más exploración abajo de ese nodo; su valor (VA) se puede transmitir a su padre como si todos sus hijos hubieran sido evaluados.

Los límites o cortes son ajustados dinámicamente.

Límite–alpha: el límite para un nodo J MIN es un límite inferior llamado *alpha*, igual al *valor más alto de todos los ancestros* MAX *de* J. La exploración de J termina en cuanto el valor actual VA es igual o menor a *alpha*.

Límite–beta: el límite para un nodo J MAX es un límite superior llamado *beta*, igual al *valor más pequeño de todos los ancestros* MIN *de* J. La exploración de J termina en cuanto el valor actual VA es igual o mayor a *beta*.

α representa el mejor valor que hemos encontrado hasta ahora para MAX y β el mejor valor (más bajo) para MIN.

Idea (ver tabla 3.4): 2 parámetros: α y β:

- Regresa el valor $V(J)$ si está entre α y β.
- Regresa α si $V(J) \leq \alpha$.

Tabla 3.4: Algoritmo Alfa-Beta.

set $\alpha = -\infty$
set $\beta = \infty$
si J es nodo terminal, *entonces* $V(J) \leftarrow eval(J)$
sino sean $J_1, J_2, \ldots, J_n$ los sucesores de J
    set $k = 1$
    si J es max
        set $\alpha \leftarrow max[\alpha, V(J; \alpha, \beta)]$
        si $\alpha \geq \beta$ regresa β, *sino* continua
        si $k = n$ regresa α,
        sino set $k \leftarrow k + 1$ y continua
    si J es min
        set $\beta \leftarrow min[\beta, V(J; \alpha, \beta)]$
        si $\beta \leq \alpha$ regresa α, *sino* continua
        si $k = n$ regresa β,
        sino set $k \leftarrow k + 1$ y continua

- Regresa β si $V(J) \geq \beta$.

3.3.1 Algoritmo Alfa-Beta

Se puede hacer una definición más concisa usando *neg–max*.

La eficiencia de *alpha–beta* depende del orden de los valores de los nodos terminales.

Si construimos un programa que pueda evaluar 1,000 posiciones por segundo, con 150 segundos por movida, podríamos ver 150,000 posiciones. Con un factor de arborescencia de 35 nuestro programa podría ver solo 3 o 4 tiradas (*ply*) adelante y jugaría a nivel de novato.

Si pudieramos ordenar los nodos terminales tendríamos que examinar $O(b^{d/2})$, por lo que el factor de arborescencia efectivo es de $\sqrt{b}$ en lugar de b, en ajedrez sería 6 en lugar de 35, por lo que ahora podríamos buscar hasta profundidad

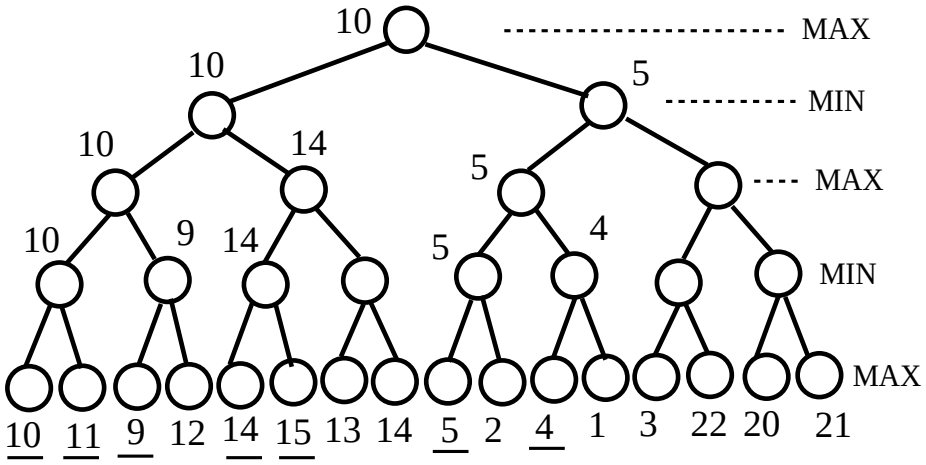

Figura 3.3: Ejemplo sencillo de la estrategia *Alfa-Beta*.

8.

A veces se usan ordenamientos sencillos, como considerar primero las capturas, luego las amenazas, luego movimientos hacia adelante y finalmente movimientos hacia atrás.

Otra estrategia es usar una búsqueda de profundidad iterativa y usar los últimos valores para decidir qué explorar en la siguiente iteración.

En promedio $\alpha - \beta$ permite explorar un 33% mas que un simple *minimax*.

3.3.2 Análisis

$\alpha - \beta$ poda el árbol al darse cuenta que los nodos podados (independientemente de sus valores) no pueden influenciar en el valor del nodo raíz.

Si definimos $F_J(x)$ como una función de influencia del nodo raíz con respecto a un nodo J, la influencia debe de transmitirse a través del camino que los conecta.

La función de influencia se puede calcular como varias funciones de influencia

consecutivas.

Lo importante es que las funciones son cerradas bajo composición por lo que se pueden escribir en términos de dos funciones nuevas:

$A(J) = max[\alpha(K), \alpha(M), \ldots]$ y $B(J) = min[\beta(L), \beta(N), \ldots]$

Estas funciones las podemos usar para evaluar los nodos explorados por $\alpha-\beta$:

Para cualquier nodo J, forma el camino desde la raíz hasta J y define la siguientes cantidades:

- A(J) = el valor minimax más alto entre todos los descendientes izquierdos de los ancestros MAX de J.
- B(J) = el valor minimax más pequeño de todos los descendientes izquierdos de los ancestros de J.

J es generado por $\alpha - \beta$ sii: $A(J) < B(J)$

3.4 SSS* (Stockman 79)

Explora caminos tipo *best–first* (AO*).

Es superior a *alpha–beta* en el sentido que:

- No explora nodos que *alpha–beta* no explora.
- Puede no explorar nodos que *alpha–beta* si explora.

Idea: tomar el árbol de búsqueda globalmente, se puede ver como una versión de AO^* y por lo mismo trata de encontrar la solución óptima.

Desventaja: tiene que guardar mucha información para considerar siempre el mejor camino a explorar.

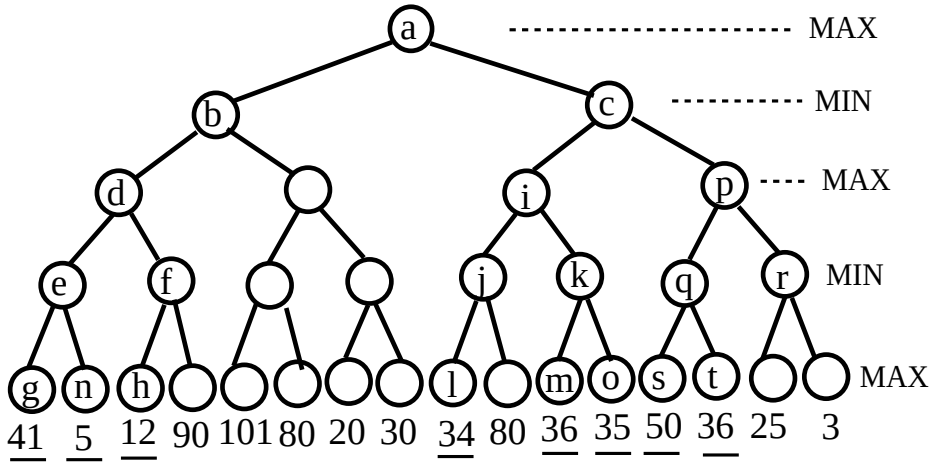

Figura 3.4: Ejemplo de la estrategia *SSS**.

La figure 3.4 muestra un ejemplo de SSS* donde las letras dentro de los nodos indican el orden en que se evalúan los nodos y los números subrayados muestran los nodos terminales que son evaluados.

Se han propuesto varias pequeñas variantes de SSS* como InterSSS*, RecSSS* y Mem*, las cuales hacen un uso más eficiente de memoria.

3.5 SCOUT (Pearl 80)

La idea básica es que mucho esfuerzo computacional se puede evitar si se tiene una forma rápida de probar desigualdades.

Idea: probar si la desigualdad $V(J) > v$ es verdadero y evaluar J sólo si es el caso, donde v es algún valor escogido para la prueba (e.g., si $ev(t) > v$ entonces se declara ganadora la posición).

Eval(J) calcula el valor minimax de posiciones MAX evaluando el sucesor de la izquieda y explora los sucesores siguientes de izquierda a derecha para determinar (llamando a Prueba) si se cumple que $V(J_k) > V(J_1)$

Tabla 3.5: Procedimiento de Prueba de SCOUT.

Procedimiento Prueba
Si J es nodo terminal, regresa **T** si $ev(J) > v$, si no regresa **F**
De izquierda a derecha:
Si J es MAX regresa **T** en cuanto un sucesor sea mayor que v
y regresa **F** si todos son menores a v
Si J es MIN regresa **F** en cuanto un sucesor sea menor que v
y regresa **T** si todos son mayores a v

Si la desigualdad se cumple, se evalua el valor exactamente y se continua.

3.6 Estrategias de Juego

Progressive/iterative deepening: explorar por niveles.

Si: $b =$ *branching factor* y $d =$ profundidad: En general el número de nodos evaluados al final es: b^d.

El número de nodos en el resto del árbol es:

$$\sum_{i=0}^{d-1} b^i = \frac{b^d - 1}{b - 1}$$

La razón entre ellos es:

$$b^d \frac{b-1}{b^d - 1} \approx b - 1$$

i.e., con profundidad de 16 el costo total es $\frac{1}{15}$ del costo normal lo cual es bastante bajo.

Heuristic Pruning: ordenar los nodos con respecto a su función de evaluación y variar la profundidad de exploración (mayor/menor). Sin embargo con sólo esta estrategia no habría sacrificios de damas en ajedrez.

Heuristic Continuation: continuar caminos que por heurísticas se consideran importantes (e.g., el rey peligra, se va a perder una pieza, un peón puede coronar, etc.), para tratar de evitar el *efecto horizonte* (no siempre se puede).

Quiescence Search: En principio, no se debería de aplicar la función de evaluación en posiciones en donde no se esperan cambios fuertes en el valor de la función de evaluación.

A esto, se le llama *quiescence search* y a veces se restringe sólo a algunos tipos particulares de movidas (e.g., capturas).

Todas las metodologías asumen que la decisión de la jugada mejora con la profundidad de la búsqueda.

Se ha mostrado que funciona bien en juegos y ésto se cuestiona poco. De hecho se cree que el nivel extra de exploración en ajedrez mejora en 200 el puntaje de la máquina.

Sin embargo, se ha mostrado que el nivel discriminatorio decrece apreciablemente en cada nivel.

Si la función de evaluación se mantiene igual en todos los niveles de juego, entre más buscamos a profundidad, peor es nuestra evaluación.

Porqué no ocurre en los juegos?

> Los juegos comunes no son uniformes, están llenos de "trampas" que son detectadas al buscar a profundidad.

Sin embargo, no se debe de perder de vista la detereoridad del algoritmo.

Algo muy usado en juegos (además de guardar aperturas y finales de juego) son las *transposition tables*. Esto es, guardar información de nodos recorridos para evitar recorrerlos otra vez.

Existen muchas otras estrategias: En B* (Berlinger 79), cada nodo tiene una evaluación pesimista y una optimista. Un nodo deja de ser explorado cuando su estimación optimista es más baja que todas las pesimistas de sus rutas alternas.

Líneas de investigación: usar *meta*–razonamiento y combinar planeación con búsqueda.

3.7 Juegos con eventos externos

Juegos, como backgammon, incluyen factores externos no predecibles (e.g., el resultado de tirar un dado).

Para este tipo de juego, se incluyen nodos intermedios (*chance nodes*) que contemplan los posibles resultados de los eventos externos con sus probabilidades.

Para propagar valores, de los nodos terminales se toman sus valores y al llegar a los nodos "aleatorios" se calcula su valor esperado sumando todas las posibilidades multiplicadas por su probabilidad para obtener un valor.

3.8 Hombres vs. Máquinas

Temores:

- Juegos aburridos
- Jugadas sin sentido para el hombre

Estado del arte:

- Ajedrez: Kasparov es derrotado en condiciones de torneo por Deep Blue (1997).

  Deep Blue (97): 32 procesadores RS/6000 (Power Two Super Chip (P2SC)), cada uno con 8 procesadores VLSI de ajedrez, para un total de 256 procesadores. Analiza alrededor de 100–200 mil millones de posiciones cada 3 minutos y alrededor de 200,000,000 posiciones por segundo. Kasparov analiza aproximadamente 3.

- Damas Chinas: Chinook es declarado el campeón mundial cuando Tinsley se retira por razones de salud (1994).
- Backgammon: TD-gammon es considerado como uno de los 3 mejores jugadores del mundo (1995). Este sistema usa aprendizaje por refuerzo, el cual veremos más adelante en el curso.
- Othello: Logistello considerado muy superior a los humanos (1994).
- Go: las máquinas no están ni cerca (factor arborescencia ≈ 360).
- Go–Moku: Se sabe que el que empieza gana (1996).