Tema 2: Búsqueda en Juegos.

- Introducción.
- Estrategias básicas de búsqueda en juegos.
  - Estrategia exhaustiva: MiniMax.
  - Estrategia de poda: α-β.
  - Estrategia SSS*.
  - Estrategia de test previo: SCOUT.
- Técnicas avanzadas.
  - Técnica de bajada progresiva.
  - Poda heurística.
  - Continuación heurística.
- Coste computacional.

Introducción

- XVIII (1760), Wolfgang Kempelen: ajedrecista mecánico

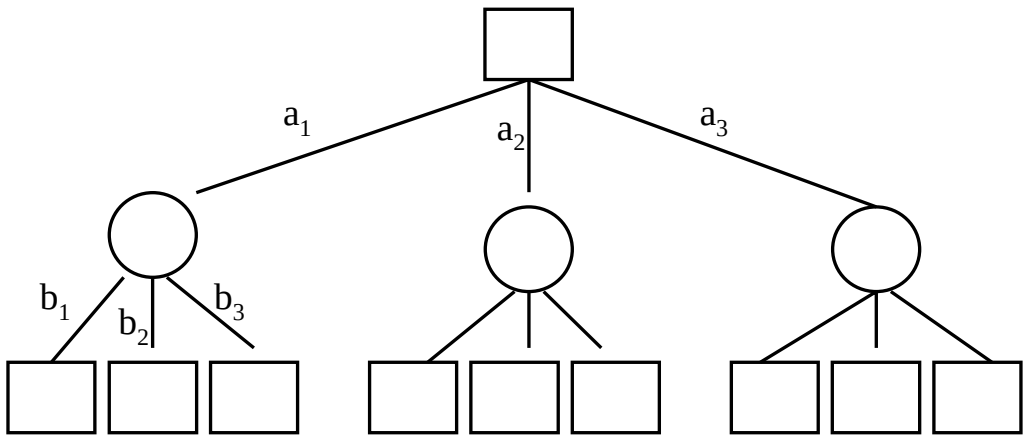
- XIX, Charles Babbage: máquina análitica, ajedrez
- Alan Turing, ajedrez
- 1963, Arthur Samuel, damas
- 1997, Deep Blue , ajedrez, gana al campeón mundial de ajedrez
- Ajedrez: en cada jugada existe una media de 35 movimientos posibles y una partida se puede resolver en unas 100 jugadas. La búsqueda exhaustiva no sirve.

Introducción.

- En este tema presentaremos un conjunto de estrategias y métodos para la búsqueda de solución en un determinado juego. Nos centraremos en un juego genérico.
- Los juegos a tratar serán juegos en los que participen dos jugadores.
  - Cada jugador está perfectamente informado de las reglas del juego y conoce en todo momento la configuración de cada jugada.
  - De estos dos jugadores cada uno intenta ganar para sí. Si decimos que un jugador gana el otro pierde y viceversa. Puede existir un estado de empate.
  - Etiquetamos a los jugadores con MAX y MIN. Por norma siempre empezará a jugar MAX y vamos a querer que gane dicho jugador.

Introducción.

- Cada estado del juego vendrá dado por una configuración de dicho juego. Denotaremos un estado con N.
- La generación de las distintas jugadas se va a representar mediante un árbol que llamaremos 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.
- El factor de ramificación se denota por B y es el número de posibles movimientos que se pueden realizar.

Introducción.

- Debido a la imposibilidad de generar todo el árbol de búsqueda lo que se suele hacer es generarlo hasta un determinado nivel de profundidad y en ese nivel aplicar alguna función de evaluación, la denotaremos como f(N).
  - Esta función nos puede devolver un valor numérico que nos diga cómo de bueno es un determinado estado.
  - A mayor valor devuelto por la función mejor será la jugada para MAX. De esta forma MAX *max*imizará esta función y MIN *min*imizará dicha dicha función.
  - En algunos casos la función nos puede devolver valores como PIERDE, GANA o EMPATA, siempre referidos a MAX.
  - **El objetivo del análisis del árbol es encontrar qué valor asignamos al nodo raíz (inicio de la jugada). A este valor se le denomina valor minimax**.

Estrategias básicas de búsqueda en juegos.

- Estrategias básicas de búsqueda en juegos.
  - Estrategia exhaustiva: MiniMax.
  - Estrategia de poda: α-β.
  - Estrategia SSS*.
  - Estrategia de test previo: SCOUT.

Estrategia exhaustiva: MiniMax (1).

- Esta estrategia genera, en un principio, todos los nodos del árbol hasta la profundidad deseada. Evalúa cada nodo hoja del árbol generado y asigna un valor al nodo raíz dependiendo de la aplicación del algoritmo descrito más abajo.
- Básicamente este algoritmo consiste en, 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 exhaustiva: MiniMax (2).

Algoritmo MINIMAX . $V(N)$
Entrada: Nodo N
Salida: Valor de dicho nodo.

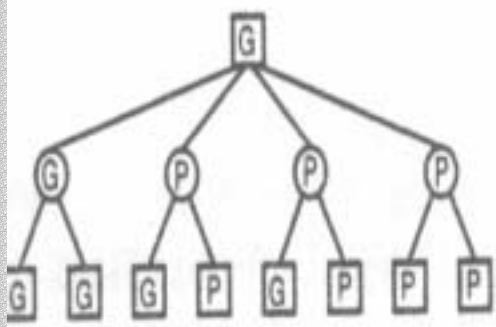
Si N es nodo hoja **entonces** devolver $f(N)$.
sino
  Generar todos los sucesores de N : $N_1, N_2, \ldots, N_b$.
  Evaluar dichos sucesores de izquierda a derecha: $V(N_1), V(N_2), \ldots, V(N_b)$.
  Si N es MAX **entonces** devolver $\max[V(N_1), V(N_2), \ldots, V(N_b)$ **FinSi.**
  Si N es MIN **entonces** devolver $\min[V(N_1), V(N_2), \ldots, V(N_b)$ **FinSi.**
FinSi

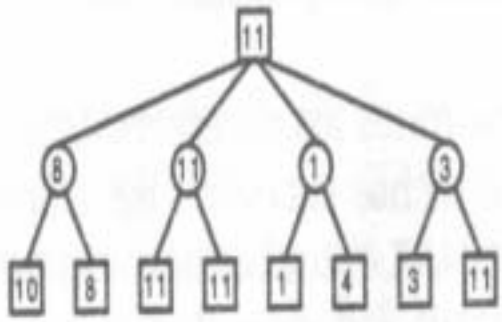
- Este algoritmo evalúa todos los nodos terminales.
- Sin embargo, cuando la función de evaluación sólo puede devolver dos valores, f(N) ∈ {GANA, PIERDE}, se puede redefinir el algoritmo para que, en el mejor de los casos, no se exploren todos los nodos.
- Esta redefinición viene dada debido a que si, por ejemplo, nos encontramos en un nodo MAX, en el momento que evalúe un nodo hijo a GANA no es necesario evaluar el resto. Lo mismo pasa con un nodo MIN, tan pronto evalúe un nodo hijo a PIERDE no debe continuar evaluando más nodos.

Estrategia exhaustiva: MiniMax (3).

Algoritmo MINIMAX **modificado.** $V(N)$
Entrada: Nodo N
Salida: Valor minimax de dicho nodo.

Si N es nodo hoja **entonces** devolver $f(N)$.
sino
Generar todos los sucesores de $N: N_1, N_2, \ldots, N_b$.
Si N es MAX **entonces** ir evaluando los hijos y devolver GANA tan pronto como un nodo hijo se evalúe a GANA. Devolver PIERDE si todos los hijos se evalúan a PIERDE **FinSi.**
Si N es MIN **entonces** ir evaluando los hijos y devolver PIERDE tan pronto como un nodo hijo se evalúe a PIERDE . Devolver GANA si todos los hijos se evalúan a GANA **FinSi.**
FinSi





Estrategia de poda: α-β. (1)

- Esta estrategia también intenta encontrar el valor minimax del nodo raíz del árbol.
- La filosofía de esta estrategia es determinar un valor y no evaluar aquellos nodos o ramas que no puedan cambiar dicho valor. Estos valores se denominan límite α y límite β. Estos se ajustan dinámicamente y se transmiten de los nodos superiores a los inferiores de la siguiente manera:
  - *Límite* α. El límite de corte para un nodo MIN es un límite inferior llamado α, que es igual al valor actual más alto de todos los predecesores MAX de dicho nodo. Terminaremos la exploración del nodo tan pronto el valor del nodo (β) sea menor o igual a α.
  - *Límite* β. El límite de corte para un nodo MAX es un límite superior llamado β, que es igual al valor actual más pequeño de todos los predecesores MIN de dicho nodo. Terminaremos la exploración del nodo tan pronto el valor del nodo (α) sea mayor o igual a β.

Estrategia de poda: α-β. (1)

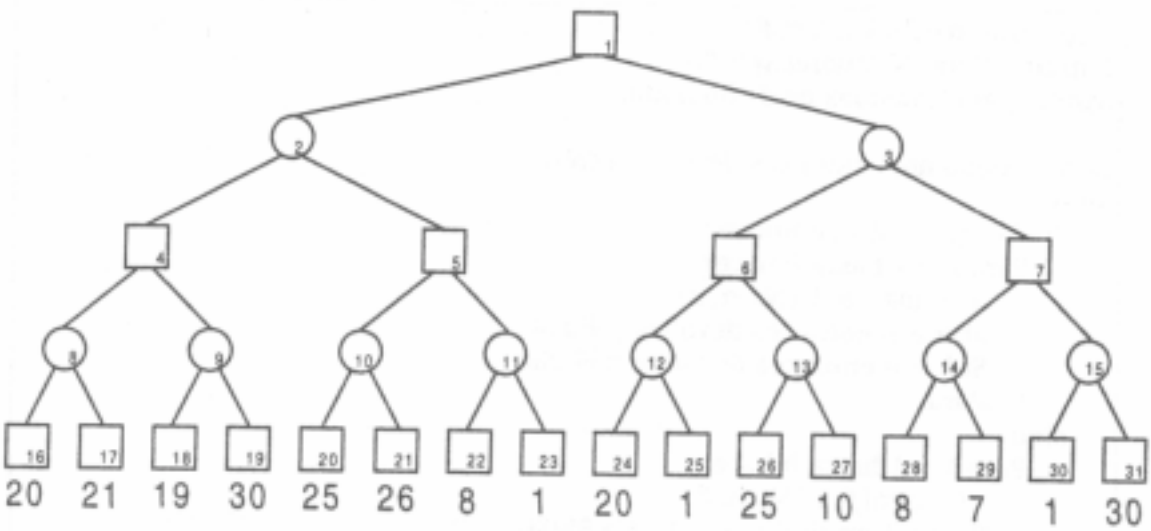
```
Algoritmo α−β . V(N, α, β)
Entrada: Nodo N, valores α y β.
Salida: Valor minimax de dicho nodo.

Si N es nodo hoja entonces devolver f(N).
sino
   Si N es nodo MAX entonces
      Para k = 1 hasta b hacer
         α = max[α, V(N_k, α, β)]
         Si α ≥ β entonces devolver β FinSi.
         Si k = b entonces devolver α FinSi.
      FinPara.
   sino
      Para k = 1 hasta b hacer
         β = min[β, V(N_k, α, β)]
         Si α ≥ β entonces devolver α FinSi.
         Si k = b entonces devolver β FinSi.
      FinPara.
   FinSi
FinSi
```

Estrategia de poda: α-β. (2)

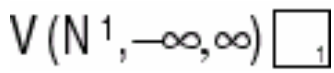
- El valor minimax de un nodo estará siempre acotado por estos valores, es decir, $\alpha \leq V(N) \leq \beta$. Al principio inicializamos $\alpha = -\infty$ y $\beta = \infty$ al no tener ninguna evidencia de estos valores. Al ir profundizando, cuando lleguemos a un nodo terminal estos valores irán cambiando. Si en algún momento $\alpha \geq \beta$, entonces se produce un corte.

- Ejemplo:



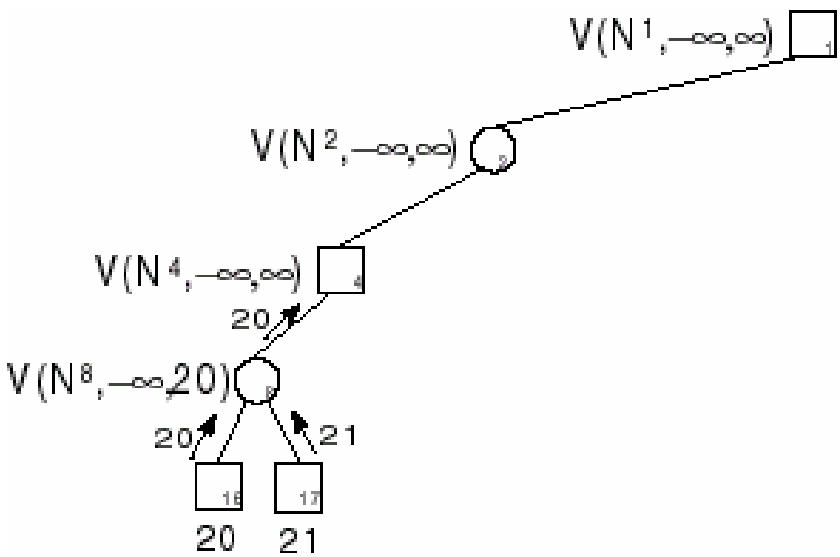
Estrategia de poda: α-β. (3).

- 1. El algoritmo empieza calculando el valor minimax del nodo raíz N^1 inicializando los valores α y β a -∞ y ∞ respectivamente.



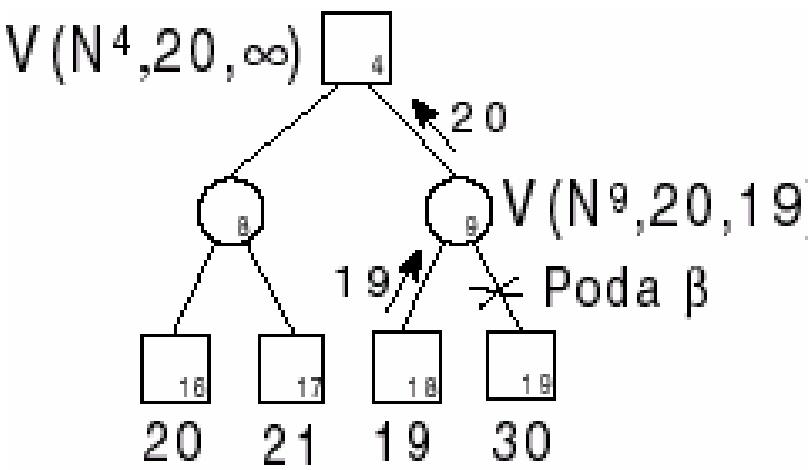
- 2. En una primera etapa desciende por la rama más a la izquierda del árbol hasta encontrar un nodo hoja, N^{16}, el cual devuelve el resultado de aplicar la función de evaluación a dicho nodo. En este caso devuelve 20.

- 3. Al retornar nos encontramos en el nodo N^8 que actualiza su valor β a 20. Volvería a calcular el mínimo entre este valor y el valor que le devuelve el siguiente hijo. Al final devuelve 20.



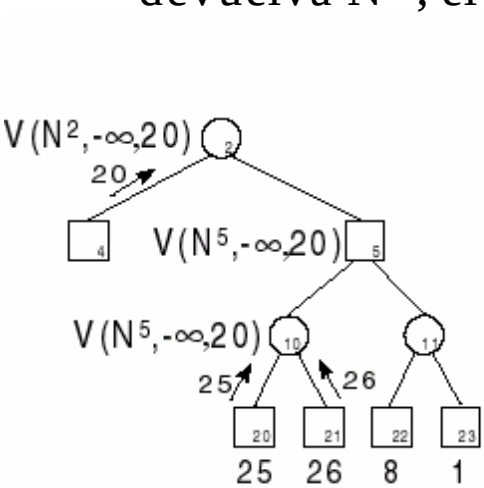
Estrategia de poda: α-β. (4).

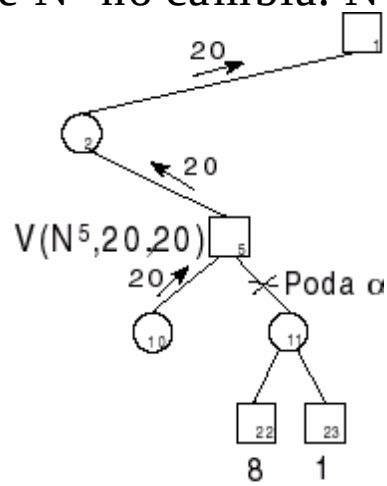
- 4. Retornamos al nodo N^4 el cual actualiza su valor α a 20. Calcula entonces el máximo entre este valor y el devuelto por su siguiente hijo.
- 5. Pasamos al siguiente nodo N^9. Este primero calcula el valor de su hijo más a la izquierda, N^{18}, y cambia el valor de β a 19. En este punto se produce una poda β: el nodo N^9 ya tiene un sucesor que le ha devuelto el valor 19. Este nodo N^9 devolverá ese valor u otro más pequeño, dado que su función es minimizar. Por otro lado N^4 ya tiene un hijo que le devuelve 20, y como N^9 le devolverá como máximo 19, N^4 elegirá 20, independientemente del valor que devuelva N^{19}, por lo que éste no se evalúa.



Estrategia de poda: α-β. (5).

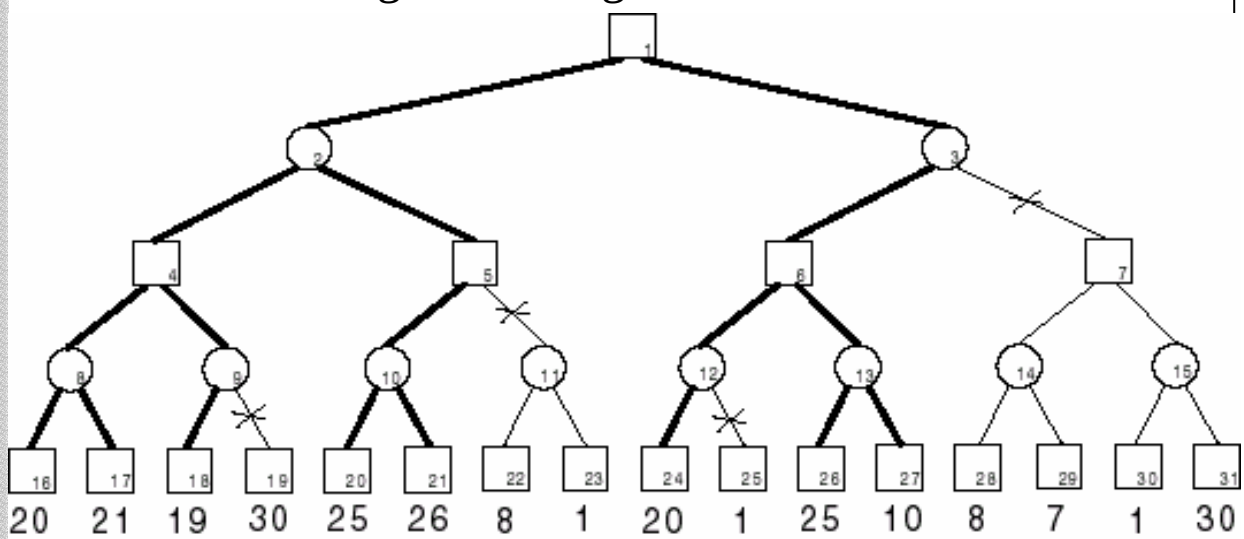
- 6. N^4 devuelve 20 y N^2 cambia su valor β a 20. Calcula el valor del siguiente hijo, N^5. Este a su vez calcula el valor de N^{10}.
- 7. En N^{10} no se produce ninguna poda, ya que el límite α es -∞, por lo que se calculan los valores de N^{20} y N^{21}, devolviendo el mínimo entre ellos y su valor α, 20.
- 8. Situados en N^5 hemos recibido el valor 20 que actualiza el valor de α, por lo que se produce una poda α y el nodo N^{11} no es evaluado. Similarmente a la poda β el nodo N^{11} no se evalúa debido a que N^5 ya tiene asignado el valor 20 y este nodo es MAX por lo que tomará como mínimo dicho valor. Sin embargo N^2 tomará como máximo 20, por lo que devuelva el valor que devuelva N^{11}, el valor de N^2 no cambia. N^2





Estrategia de poda: α-β. (6).

- 9. Ahora se actualiza el valor de N^1 y pasamos a la siguiente rama, en la cual se producen varias podas al aplicar el algoritmo, quedando al final una configuración como la mostrada en la figura donde las cruces indican podas y las aristas más gruesas los caminos de evaluación que ha seguido el algoritmo:



» Hemos reducido el número de evaluaciones de nodos terminales frente al algoritmo exhaustivo MiniMax .

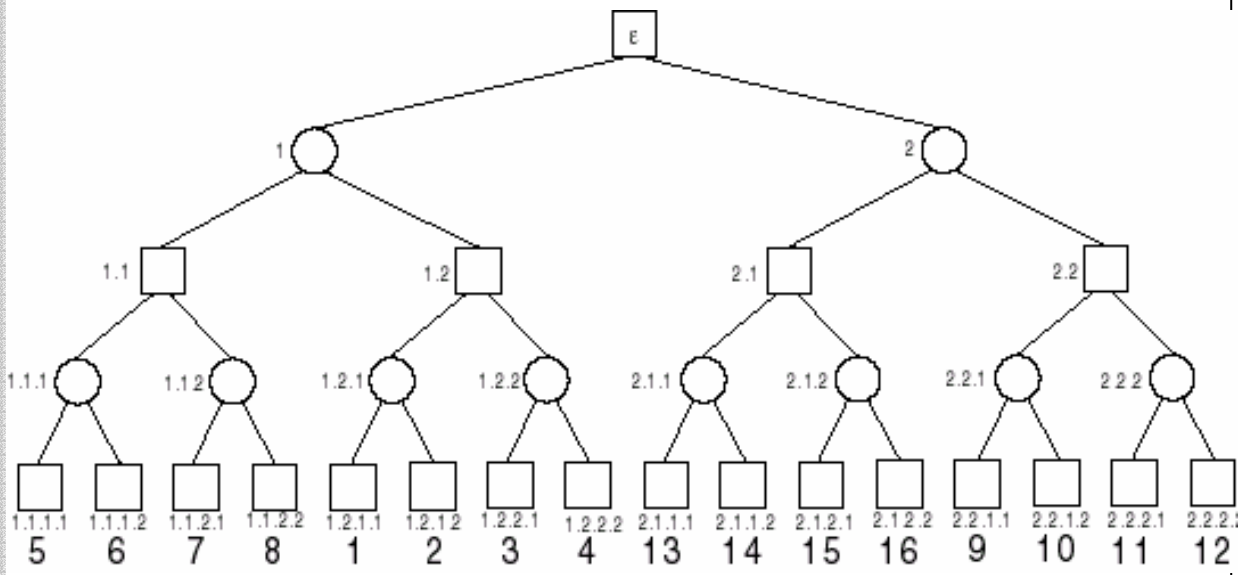
» Sin embargo la eficacia del algoritmo depende de la ordenación de los nodos terminales. Una buena ordenación de estos nodos terminales puede suponer el que sólo se evalúen la mitad de los nodos que se evaluarían si los nodos estuvieran mal ordenados.

Estrategia SSS*. (1)

- Esta estrategia supone una mejora al algoritmo α-β debido a que soluciona el problema de la ordenación de los nodos terminales. Sin embargo, la estructura utilizada para manejar los nodos implica un aumento de la complejidad espacial que es la desventaja de este algoritmo.
- SSS* utiliza subárboles del árbol de juego (normalmente ramas de dicho árbol) y realiza un refinamiento de esos subárboles, los cuales tienen asignados el límite inferior más alto de sus constituyentes. Es decir, el valor asignado al subárbol siempre será mayor o igual que la mejor solución encontrada dentro de ese subárbol.
- Suponemos que empieza jugando MAX por lo que buscamos el valor más alto posible entre todos los valores asignados a los nodos terminales. Este valor asignado al subárbol sirve para realizar la ordenación y evitar la dependencia a la ordenación que sufría el algoritmo α-β , utilizando una estrategia similar al algoritmo A*.

Estrategia SSS*. (2)

- SSS* utiliza la notación decimal de Dewey para representar los nodos en el árbol.



- Notar que el nodo raíz se etiqueta con la secuencia vacía ε. Esta notación se construye de tal manera que si tenemos un nodo etiquetado N.n estamos situados en el hijo n, contando desde la izquierda, del nodo N.
  - El algoritmo hace también uso de un estado definido como una tripleta (N, s, h) donde N indica un nodo del árbol, s ∈ {VIVO, SOLUCIONADO} es el estado de la solución de N y h ∈ [-∞,∞] es el valor del estado.
  - Un nodo etiquetado con VIVO indica que aún se puede seguir generando sucesores del nodo,
  - SOLUCIONADO indica que todos los sucesores ya han sido generados.

Estrategia SSS*. (3)

- Por último el algoritmo utiliza una lista de estados LISTA donde almacena los estados en orden decreciente con respecto a h, de tal forma que un estado de la lista tiene un valor de h igual o mayor al resto de estados por detrás de él.

Algoritmo SSS^* . $V(N)$
Entrada: Nodo N.
Salida: Valor minimax de dicho nodo.

Insertar en $LISTA$ el estado inicial $(N = \epsilon, s = VIVO, h = +\infty)$
Hacer siempre
  Seleccionar y eliminar el primer estado de $LISTA$: $p = (N, s, h)$.
  Si $N = \epsilon$ **y** $s = SOLUCIONADO$ **entonces** devolver h **FinSi**
  Expandir el estado p de la siguiente forma:
  Si $s = VIVO$ **entonces**
    Si N no es terminal **entonces**
      Si N es MAX **entonces**
        Para $n = b$ **hasta** 1 **hacer**
          Insertar el estado $(N.n, s, h)$ en la cabeza de $LISTA$.
        FinPara
      sino /*N es MIN*/
        Insertar el estado $(N.1, s, h)$ en la cabeza de $LISTA$.
      FinSi
    sino /*N es terminal*/
      Insertar el estado $(N, SOLUCIONADO, \min(h, f(N)))$ en $LISTA$, delante de todos los estados con menor h que este estado.
    FinSi
  sino /*$s = SOLUCIONADO$*/
    Si N es MAX **entonces**
      Sea $N = M.n$
      Si $n \neq b$ **entonces**
        Insertar $(M.n + 1, VIVO, h)$ en la cabeza de $LISTA$.
      sino /*$n = b$*/
        Insertar $(M, SOLUCIONADO, h)$ en la cabeza de $LISTA$.
      FinSi
    sino /*NesMIN*/
      Sea $N = M.n$
      Insertar $(M, SOLUCIONADO, h)$ en la cabeza de $LISTA$.
      Eliminar de $LISTA$ todos aquellos estados sucesores de M.
    FinSi
  FinSi
FinHacer

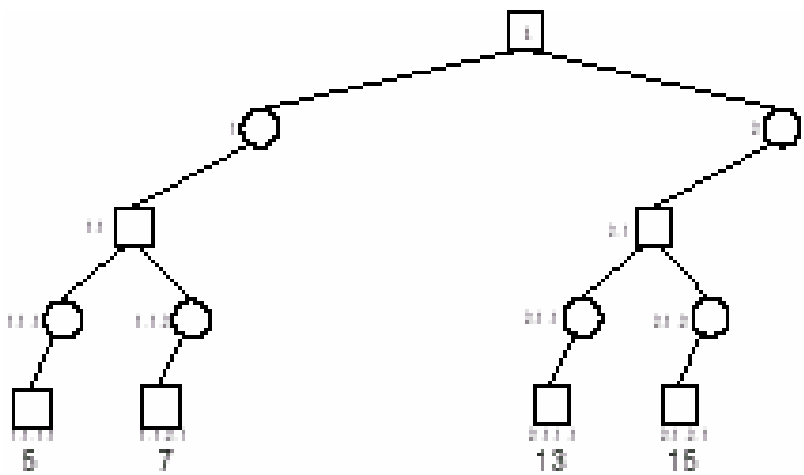
Estrategia SSS*. (4)

- Ejecución del algoritmo SSS*

Iter	LISTA			
1	(ε,V,∞)			
2	(1,V,∞),	(2,V,∞)		
3	(1.1,V,∞),	(2,V,∞)		
4	(1.1.1,V,∞),	(1.1.2,V,∞),	(2,V,∞)	
5	(1.1.1.1,V,∞),	(1.1.2,V,∞),	(2,V,∞)	
6	(1.1.2,V,∞),	(2,V,∞),	(1.1.1.1,S,5)	
7	(1.1.2.1,V,∞),	(2,V,∞),	(1.1.1.1,S,5)	
8	(2,V,∞),	(1.1.2.1,S,7),	(1.1.1.1,S,5)	
9	(2.1,V,∞),	(1.1.2.1,S,7),	(1.1.1.1,S,5)	
10	(2.1.1,V,∞),	(2.1.2,V,∞),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
11	(2.1.1.1,V,∞),	(2.1.2,V,∞),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
12	(2.1.2,V,∞),	(2.1.1.1,S,13),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
13	(2.1.2.1,V,∞),	(2.1.1.1,S,13),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
14	(2.1.2.1,S,15),	(2.1.1.1,S,13),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
15	(2.1.2.2,V,15),	(2.1.1.1,S,13),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
16	(2.1.2.2,S,15),	(2.1.1.1,S,13),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
17	(2.1.2,S,15),	(2.1.1.1,S,13),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
18	(2.1,S,15),	(1.1.2.1,S,7),	(1.1.1.1,S,5)	
19	(2.2,V,15),	(1.1.2.1,S,7),	(1.1.1.1,S,5)	
20	(2.2.1,V,15),	(2.2.2,V,15),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
21	(2.2.1.1,V,15),	(2.2.2,V,15),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
22	(2.2.2,V,15),	(2.2.1.1,S,9),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
23	(2.2.2.1,V,15),	(2.2.1.1,S,9),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
24	(2.2.2.1,S,11),	(2.2.1.1,S,9),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
25	(2.2.2.2,V,11),	(2.2.1.1,S,9),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
26	(2.2.2.2,S,11),	(2.2.1.1,S,9),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
27	(2.2.2,S,11),	(2.2.1.1,S,9),	(1.1.2.1,S,7),	(1.1.1.1,S,5)
28	(2.2,S,11),	(1.1.2.1,S,7),	(1.1.1.1,S,5)	
29	(2,S,11),	(1.1.2.1,S,7),	(1.1.1.1,S,5)	
30	(ε,S,11)			

Estrategia SSS*. (5)

- Si aplicamos el algoritmo α-β a este árbol no se consigue realizar ninguna poda mientras que con el algoritmo SSS* evaluamos únicamente la mitad de los nodos. El algoritmo funciona intuitivamente en dos etapas. En una primera etapa se generan un conjunto de ramas, expandiendo el primer hijo si se trata de un nodo MIN y todos los hijos si se trata de uno MAX. Como al principio el valor de cada estado es ∞, se expandirán todas las posibles ramas, que en este caso son cuatro. Se corresponde desde la primera iteración hasta la 14.



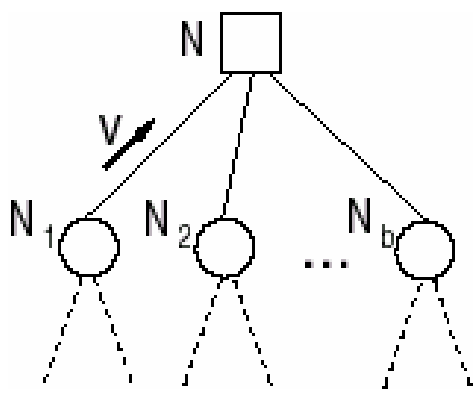
Estrategia SSS*. (6)

- A partir de entonces se van seleccionando estados según su valor h, por lo que se evita la dependencia a la ordenación que sufría el algoritmo α-β . Básicamente el algoritmo va seleccionando los estados. Si no está solucionado se siguen generando sucesores hasta que se solucionen. Si el estado está solucionado se soluciona el padre, caso de un nodo MIN o que sea el último hermano de un MAX, o se genera el siguiente hermano para solucionar, caso de un MAX que no es el último hermano.

Estrategia de test previo: SCOUT. (1)

- El algoritmo SCOUT se propone partiendo de la base de que disponemos de un método rápido para comprobar desigualdades. Es decir, disponemos de una función que permite comprobar si el valor minimax de un determinado nodo es menor o mayor que un determinado valor.

  - » Veamos esto con un ejemplo. Tenemos el árbol parcial mostrado en la figura.Suponemos que el nodo N ha calculado el valor de su hijo N_1 y es igual a v. El algoritmo α-β pasaría a explorar la rama N_2. Sin embargo, si pudiéramos saber que el valor minimax de N_2 es menor o igual a V no exploraríamos esa rama. Aquí es donde SCOUT tiene su potencia: no explora ramas que no le lleven a mejor solución.



Estrategia de test previo: SCOUT. (2)

- SCOUT utiliza dos rutinas: La primera de ella es la que resuelve la desigualdad y la llamaremos TEST. Recibe como parámetros el nodo a aplicar el test, el valor de comparación y la desigualdad a aplicar, que puede ser $>$ o $\geq$. Esta función devuelve verdadero o falso dependiendo si se cumple o no la desigualdad. En la figura se muestra esta rutina. Esta rutina se define para la desigualdad $>$. De forma análoga se define para $\geq$.
- La segunda rutina se denomina EVAL y es la que calcula el valor minimax de un determinado nodo. Hace uso de la anterior rutina para comprobar si debe explorar una determinada rama.

Estrategia de test previo: SCOUT. (3)

Algoritmo EVAL. EVAL(N)
Entrada: Nodo N
Salida: Valor minimax de dicho nodo.

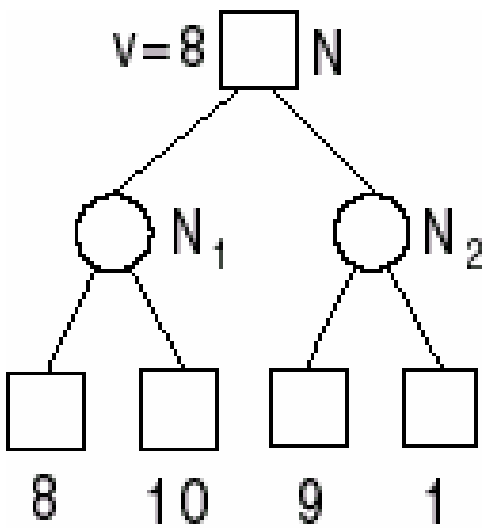
```
Generar los hijos de N:N1, N2, Nb
Si N es nodo terminal
        v=f(N)
sino
        v=EVAL(N1)
        Para i=2 hasta b hacer
                Si N es MAX entonces
                        Si TEST(Ni,v,>) entonces v=EVAL(Ni) FinSi
                sino /*N es MIN*/
                        Si NO(TEST(Ni,v,≥)) entonces v=EVAL(Ni) FinSi
                FinSi
        FinPara
FinSi
Devolver v.
```

Algoritmo TEST. TEST(N,v,>)
Entrada: Nodo N, valor v, operando >.
Salida: Verdadero o falso.

```
Si N es nodo terminal entonces
        Si f(N)>v entonces devolver CIERTO
        sino devolver FALSO
        FinSi
sino
        Para i=1 hasta b hacer
                Si N es MAX entonces
                        Si TEST(Ni,v,>) entonces devolver CIERTO FinSi
                Sino /*N es MIN*/
                        Si NO(TEST(Ni,v,>)) entonces devolver FALSO FinSi
                FinSi
        FinPara
/* Ningún hijo cumple la condición */
Si N es MAX entonces devolver FALSO sino devolver CIERTO FinSi
FinSi
```

Estrategia de test previo: SCOUT. (4)

- En la figura se puede observar un ejemplo de aplicación del algoritmo. Aplicamos EVAL sobre el nodo N y primero obtendría el valor de su hijo N_1. Para ello aplicaría a otra vez el algoritmo sobre su hijo y al final devuelve 8. Ahora, situados en N aplicaríamos TEST sobre el hijo N_2 y éste devolvería FALSO, por lo que esa rama no se explora. El algoritmo α-β sí que explora esa rama y no realiza poda ninguna. SCOUT utiliza menos memoria que SSS^* .



Técnicas avanzadas.

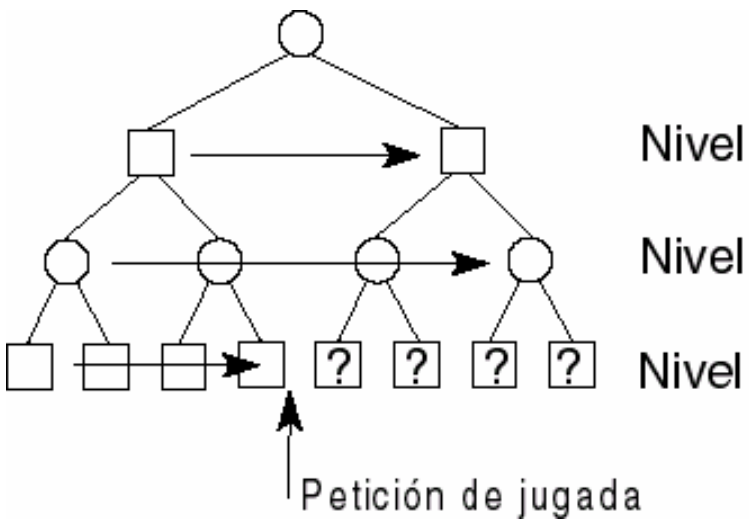
- Uso de movimientos de libro
- Técnica de bajada progresiva.
- Poda heurística.
- Continuación heurística.

Uso de movimientos de libro.

- Juegos complicados: imposible seleccionar un movimiento consultando la configuración actual del juego en un catálogo y extrayendo el movimiento correcto. El catálogo sería inmenso, ¿cómo construirlo?
- Es un enfoque razonable en para algunas partes de ciertos juegos. En ajedrez, por ejemplo, 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

Técnica de bajada progresiva. (1)

- Si el juego a evaluar impone unas restricciones de tiempo, los algoritmos presentados anteriormente pueden no ser los más adecuados para dicha evaluación.
- Para solucionar este problema se aplica la técnica de bajada progresiva que consiste en ir recorriendo los nodos por niveles y cuando llega la petición de jugada devolveremos la solución del último nivel completo.



Técnica de bajada progresiva. (2)

- Se puede pensar que se producen excesivas evaluaciones de nodos, comparado con los algoritmos anteriormente descrito. Vamos a demostrar que esto no es cierto.
  - El número de nodos terminales de un árbol con un factor de ramificación de B y una profundidad de d es igual a B^d nodos. Por otro lado el número de nodos no terminales en dicho árbol se puede calcular con la serie:

    $$B^0+B^1+\ldots+B^{d-1} = (B^d-1)/(B-1)$$

  - Si dividimos el número de nodos terminales entre el número de nodos no terminales tenemos:

    $$B^d(B-1)/(B^d-1) \approx B-1$$

  - que indica que el número de nodos terminales es B-1 veces mayor que el número de nodos no terminales, por lo que no se produce sobreevaluación de nodos.

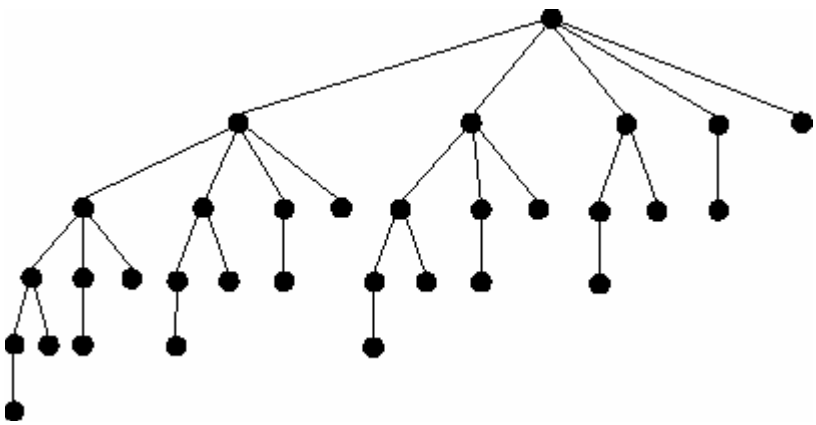
Poda heurística. (1)

- El objetivo de esta técnica consiste en reducir el factor de ramificación B desarrollando únicamente los mejores movimientos de cada nivel.
- Hace uso de una función de evaluación g(N) adicional de bajo coste. Se suele utilizar una versión simplificada de la función de evaluación aplicada a los nodos terminales. Esta función g(N) se utiliza para realizar una ordenación en cada nivel, de mayor a menor, de tal manera que, una vez ordenados, el primer nodo de un nivel es el que mayor valor de g(N) ha obtenido.
- El factor de ramificación de cada nodo es un determinado nivel viene dado por el factor del padre menos el número de orden con respecto a sus hermanos:

$$Factor(Nodo) = Factor(Padre(Nodo)) - Rango(Nodo)$$

Poda heurística. (2)

- La figura muestra un ejemplo de un árbol generado según esta técnica. Los nodos más a la izquierda tienen mayor valor de g(N).



Continuación heurística.

- Intenta limitar en anchura .
- Esta técnica intenta también evitar el efecto horizonte. Este efecto es el provocado por la limitación en profundidad: sólo podemos tener conocimiento hasta la profundidad seleccionada. Imaginemos que descendemos hasta una profundidad d y el algoritmo aplicado nos dice que el mejor nodo hasta ese nivel es N_k. Sin embargo si descendiéramos un par de niveles más es posible que esa jugada indique una pérdida de partida, pero eso no lo podemos saber hasta llegar a ese nivel, el horizonte no nos deja ver más allá.
- La técnica de continuación heurística consiste en desarrollar en anchura hasta un determinado nivel y a continuación seleccionar un subconjunto de nodos terminales para desarrollar búsquedas más profundas a partir de ellos. La selección de dichos nodos puede venir dada por un conjunto de heurísticas directamente relacionadas con el juego.

Coste computacional.

- Depende en gran medida del número de nodos terminales examinados, es decir, del número de evaluaciones realizadas. Como vimos con el algoritmo α-β , el número de evaluaciones depende a su vez de la ordenación de los nodos terminales.
- MiniMax: Examina todos los nodos terminales con lo cual tenemos una complejidad de B^d.
- α-β : En el mejor de los casos la estrategia examina

  $2B^{d/2}-1$ si d es par

  $2B^{(d+1)/2}+B^{(d-1)/2}-1$ si d es impar
- SSS* : no tenemos una expresión analítica. La complejidad promedio de SSS* indica una cierta mejora con respecto a α-β , derivada de la independencia a la ordenación de este algoritmo.
- SCOUT: no disponemos de una expresión analítica. Sin embargo comparado de forma experimental presenta mejoras en algunos casos frente a α-β y SSS* , pero en otros casos estos dos últimos algoritmos presentan mejores respuestas que SCOUT.