

Poda $\alpha - \beta$ El resultado que proporciona el algoritmo MINIMAX se denomina *valor minimax* del nodo raíz y permite obtener la decisión óptima para el horizonte de búsqueda fijado. Sin embargo, cuando el factor de ramificación (número de sucesores) es alto, como ocurre en el ajedrez, este procedimiento puede ser impracticable incluso para horizontes pequeños, puesto que obliga a un número de evaluaciones [cálculo de $e()$] exponencial con el valor de dicho horizonte. Afortunadamente, a menudo es posible calcular el valor minimax sin tener que evaluar todos los nodos hoja. La idea básica es que si, por ejemplo, asumimos un recorrido del árbol de izquierda a derecha (descenso en profundidad) puede darse el caso de que la información obtenida de la parte izquierda del árbol sea suficiente para calcular el valor minimax. O sea, que en algún momento de la búsqueda descubrimos que los nodos hoja que forman parte de un determinado sub-árbol no añaden información suficiente para cambiar la decisión y su evaluación puede obviarse, es decir, pueden ser *podados*.

Supongamos, por ejemplo, una situación en la que cada nodo del árbol (sea hoja o interior) tenga un hermano con peor evaluación a su derecha (véase la Figura 2.16a). Como el descenso se realiza en profundidad (se produce un barrido de nodos hoja de izquierda a derecha), entonces las mejores evaluaciones se encontrarán al principio. Esto puede dar lugar a situaciones en las que una vez conocemos el primer valor que llega, por ejemplo a un sub-árbol encabezado por un nodo MIN, sepamos que no es necesario seguir explorando dicho sub-árbol ya que este valor es menor que el primer valor

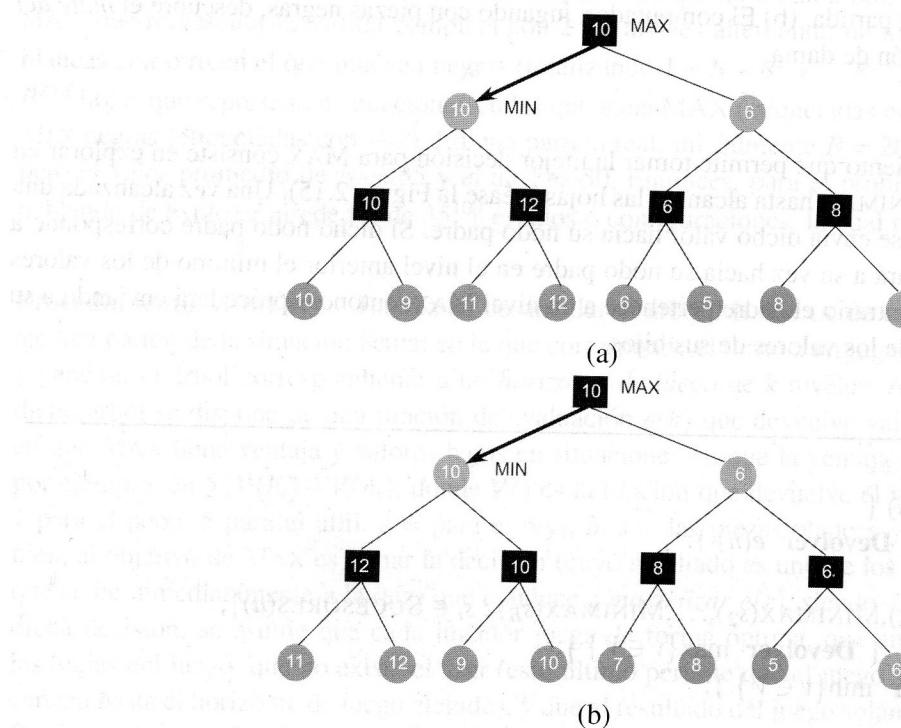


Figura 2.16. Mejor (a) y peor configuración (b) para $\alpha - \beta$.

que llega al nodo MAX padre de dicho nodo MIN. Esta situación da lugar a lo que se llama un *corte* β . La situación simétrica, cuando el sub-árbol se encabeza por un nodo MAX, se denomina corte α . Pues bien, cuando cada nodo tiene a su derecha un valor peor, el número de cortes totales es máximo, en consecuencia, también lo es el ahorro de evaluaciones de nodos hoja, de tal manera que el número de evaluaciones se reduce aproximadamente a $B^{l/2}$, lo cual quiere decir que podemos descender al doble de profundidad que con minimax en el mismo tiempo. Por el contrario, cuando cada nodo tiene a su derecha un valor peor (véase la Figura 2.16b) no se produce ningún corte y tenemos exactamente el comportamiento del algoritmo MINIMAX.

Algoritmo $\alpha - \beta$ El algoritmo que implementa las podas descritas anteriormente se basa en ajustar dinámicamente (véase la Figura 2.17) los llamados *límite- α* y *límite- β* de cada nodo. Para el nodo raíz, dichos límites valen respectivamente $-\infty$ y $+\infty$, ya que con toda seguridad el valor MINIMAX del nodo raíz estará comprendido entre estos límites. Dichos valores se propagan recursivamente hacia abajo y se actualizan. Los nodos MAX actualizan su límite α tan pronto como reciben un valor $v > \alpha$ y los nodos MIN actualizan su límite β tan pronto como reciben un valor $v < \beta$. La situación de poda se detecta en el momento en que un nodo satisface $\alpha \geq \beta$ y ello implica obviar la exploración del sub-árbol del siguiente nodo hijo, y devolver β si se trata de un nodo MAX o bien α si se trata de un nodo MIN. En caso de que todos los nodos hijos hayan sido explorados sin que se haya producido ninguna poda se devolverá α si se trata de un nodo MAX o bien β si es un nodo MIN. Así, si se da el caso de que no se produce ninguna poda el algoritmo se comporta de forma idéntica a MINIMAX.

```

Algoritmo  $\alpha - \beta(n, \alpha, \beta)$  {
  Si (Es_HOJA( $n$ )) Devolver  $e(n)$ ;
  Sino {
    Si (Es_MAX( $n$ )) {
      Para cada  $s \in$  SUCESORES( $n$ ) Hacer{
         $\alpha \leftarrow \max\{\alpha, \alpha - \beta(s, \alpha, \beta)\};$ 
        Si ( $\alpha \geq \beta$ ) { Devolver  $\beta$  };
        Si (Es_ULTIMO( $s$ )) { Devolver  $\alpha$  };
      }
    } Sino {
      Para cada  $s \in$  SUCESORES( $n$ ) Hacer{
         $\beta \leftarrow \min\{\beta, \alpha - \beta(s, \alpha, \beta)\};$ 
        Si ( $\alpha \geq \beta$ ) { Devolver  $\alpha$  };
        Si (Es_ULTIMO( $s$ )) { Devolver  $\beta$  };
      }
    }
  }
}

```

Figura 2.17. Algoritmo de poda $\alpha - \beta$.

Traza de $\alpha - \beta$ para el caso óptimo Veamos cómo funciona el algoritmo $\alpha - \beta$ para el árbol de la Figura 2.16a (caso óptimo). En primer lugar, se propaga desde la raíz el intervalo $(\alpha = -\infty, \beta = +\infty)$ hasta que se alcanza la primera hoja ($e = 10$) (véase la Figura 2.18a). Este valor se devuelve y se actualiza el α del nodo padre. Como no se produce corte se recibe el valor del segundo hijo ($e = 9$) que no mejora el valor α del padre.

Entonces, el padre MAX envía hacia arriba su valor $\alpha = 10$, que acaba actualizando el valor β del nodo padre (véase la Figura 2.18b). Entonces, este nodo MAX propaga hacia su hijo más a la izquierda el par $(-\infty, 10)$. Pero cuando este hijo recibe el valor de la hoja con $e = 12$ este nodo MAX se da cuenta de que no es necesario consultar el valor de su otro hijo: si este hijo aporta un valor $k > 12$, entonces cuando el nodo MAX lo propague hacia arriba, este valor nunca podrá modificar la mejor

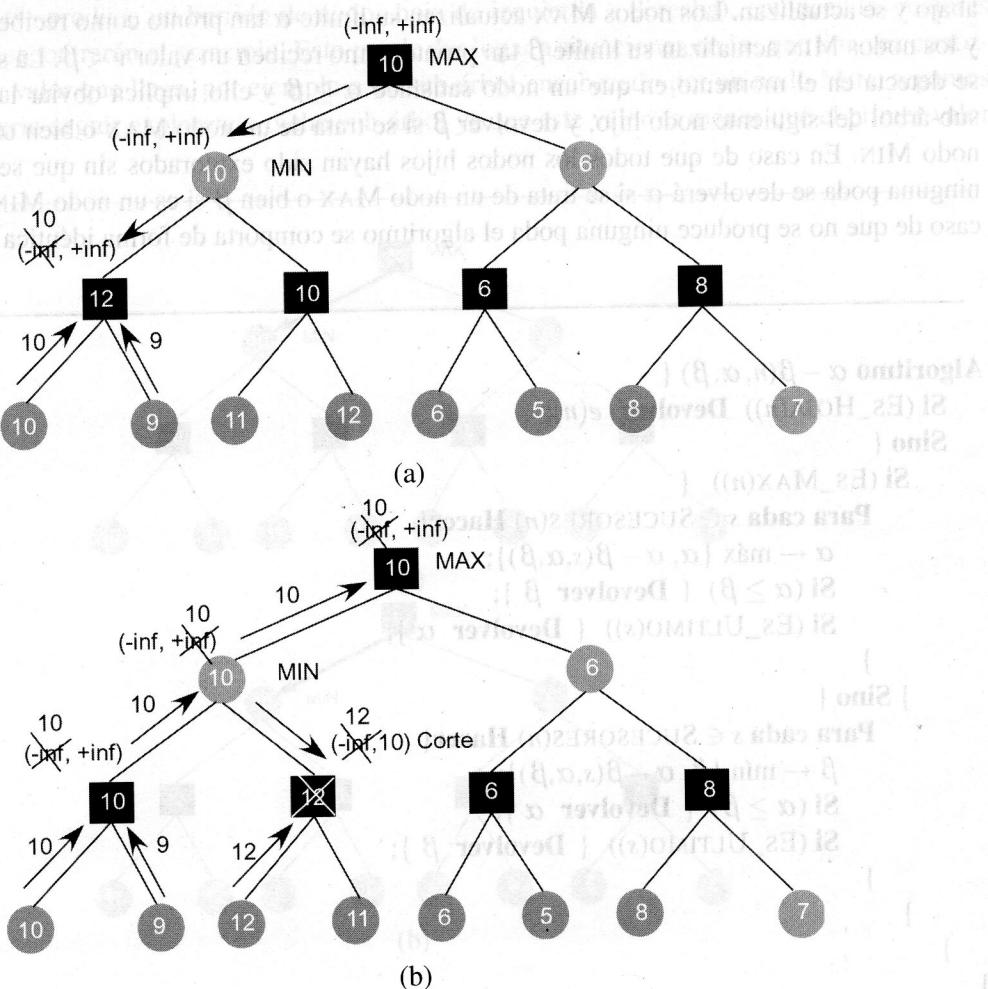


Figura 2.18. Traza de $\alpha - \beta$ (primera parte).

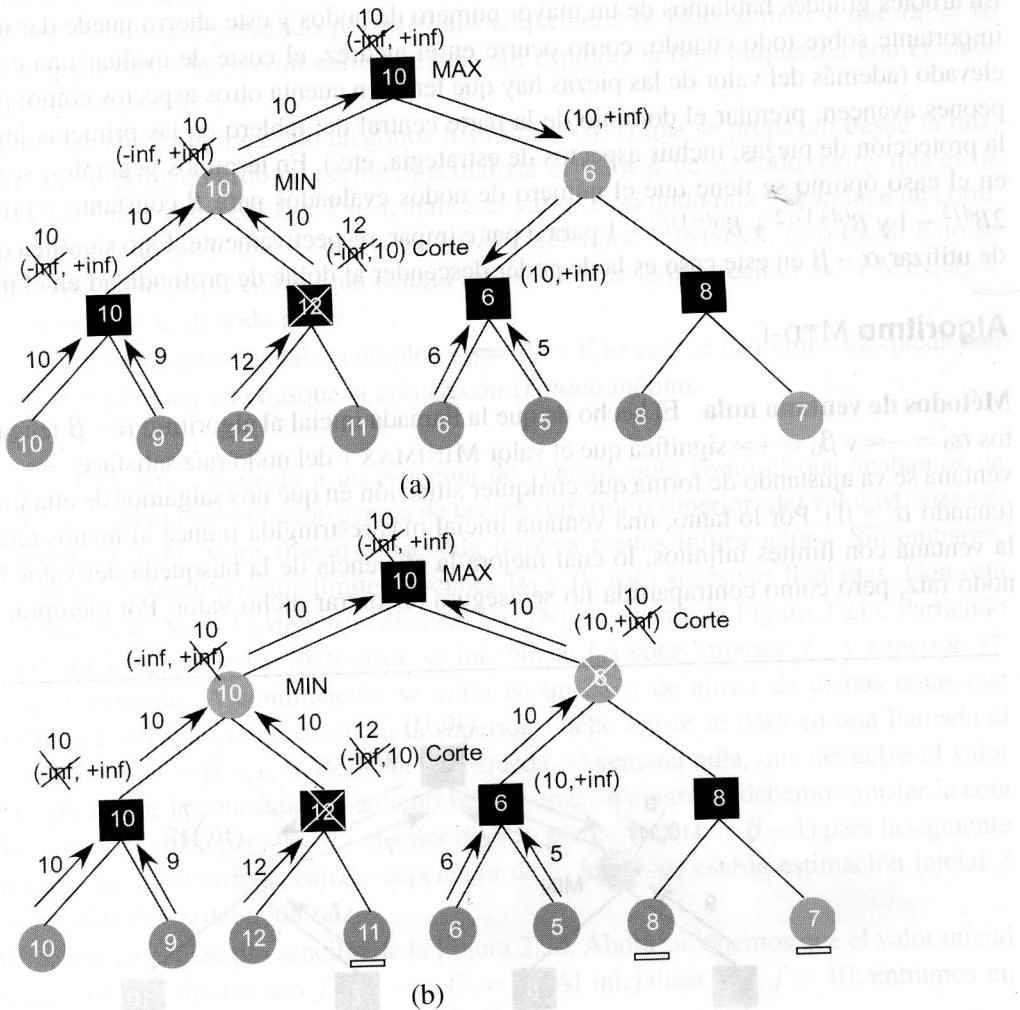


Figura 2.19. Traza de $\alpha - \beta$ (segunda parte).

opción de su padre MIN, que es el valor 10. Si por el contrario $k < 12$, ese valor no interesa al propio nodo MAX cuyo mejor valor seguirá siendo 12. Esto es lo que nos indica la situación $\alpha = 12 \geq 10$.

Por tanto, el nodo MAX que declara el corte emite hacia arriba su valor $\beta = 10$ (Figura 2.19a) y ese valor acaba modificando el valor α de la raíz, que procede propagando hacia su segundo hijo el par $(10, +\infty)$. Ese par desciende hasta el nodo MAX del penúltimo nivel y entonces se intenta actualizar con los valores $e = 6$ y $e = 5$. Como no se consigue, se envía hacia arriba el valor $\alpha = 10$ (Figura 2.19b). Es entonces cuando se actualiza el valor β del nodo padre y entonces ante el par $(10, 10)$ este nodo MIN declara un nuevo corte, con lo que no procede investigar el segundo sub-árbol de dicho nodo, puesto que ya se sabe que lo que venga por él no va a modificar la mejor opción del nodo padre (en este caso la raíz) que es el valor 10.

Finalmente, se observa que en este caso se ha omitido la evaluación de 3 de los 8 nodos hoja. En árboles grandes hablamos de un mayor número de nodos y este ahorro puede dar un tiempo extra importante sobre todo cuando, como ocurre en el ajedrez, el coste de evaluar una configuración es elevado (además del valor de las piezas hay que tener en cuenta otros aspectos como: premiar que los peones avancen, premiar el dominio de la parte central del tablero en las primeras jugadas, premiar la protección de piezas, incluir aspectos de estrategia, etc.). En términos generales, se demuestra que en el caso óptimo se tiene que el número de nodos evaluados para B constante y profundidad d es $2B^{d/2} - 1$ y $B^{(d+1)/2} + B^{(d-1)/2} - 1$ para d par e impar, respectivamente. Esto significa que la ganancia de utilizar $\alpha - \beta$ en este caso es la de poder descender al doble de profundidad en el mismo tiempo.

Algoritmo MTD-f

Métodos de ventana nula El hecho de que la llamada inicial al algoritmo $\alpha - \beta$ tenga los argumentos $\alpha_0 = -\infty$ y $\beta_0 = +\infty$ significa que el valor MINIMAX v del nodo raíz satisface $-\infty < v < +\infty$. Esta ventana se va ajustando de forma que cualquier situación en que nos salgamos de ella implica un corte (cuando $\alpha \geq \beta$). Por lo tanto, una ventana inicial más restringida induce al menos tantos cortes que la ventana con límites infinitos, lo cual mejora la eficiencia de la búsqueda del valor MINIMAX del nodo raíz, pero como contrapartida no se asegura encontrar dicho valor. Por ejemplo, en el sencillo

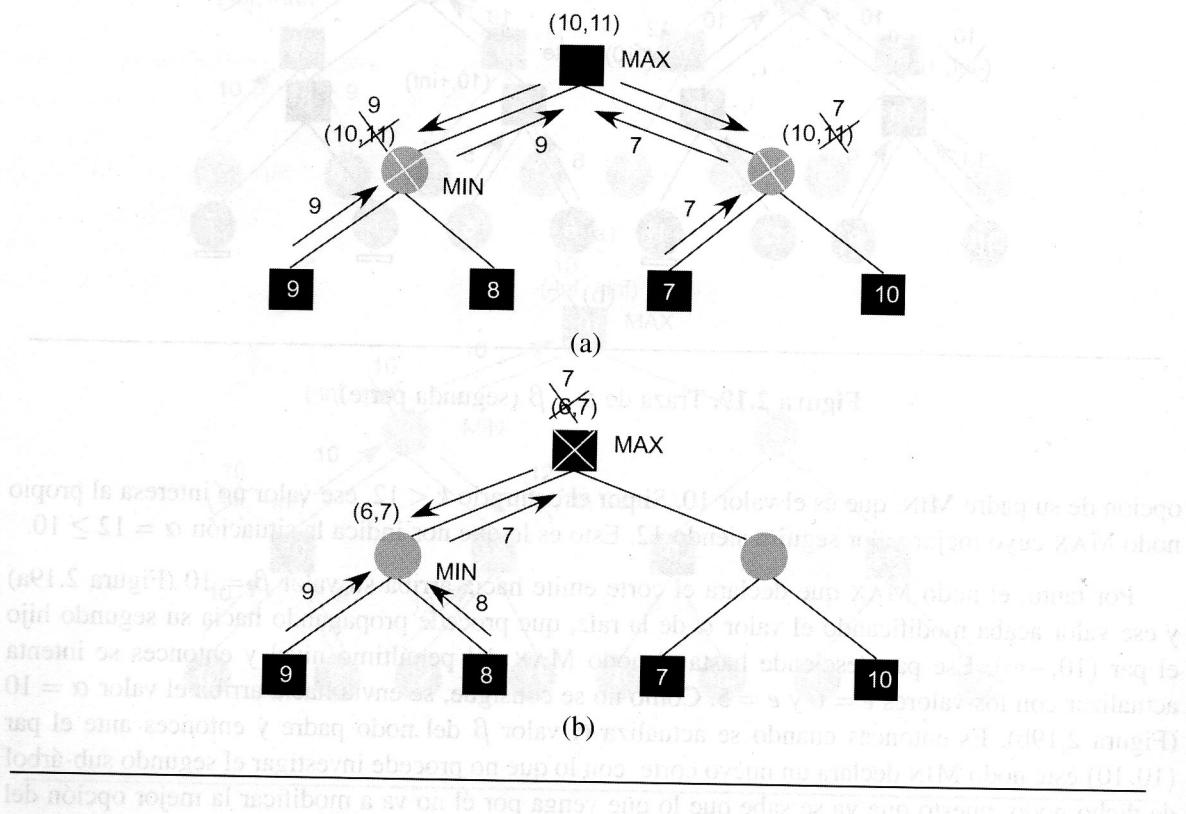


Figura 2.20. Uso de ventana nula en $\alpha - \beta$.