# 7
# Grafos implícitos para juegos

## Grafos y/o

Muchos problemas complejos pueden descomponerse en una serie de subproblemas tales que la solución de todos, o algunos de ellos, constituya la solución del problema original. A su vez, estos subproblemas pueden descomponerse en sub-subproblemas, y así sucesivamente, hasta conseguir problemas lo suficientemente sencillos como para poderlos resolver sin dificultad.

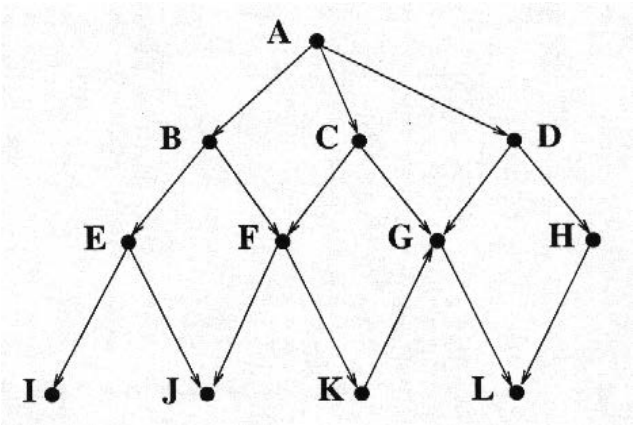
El paso previo para la búsqueda de soluciones de un problema es la especificación del problema, es decir, conocer con tanta exactitud como se pueda el problema que tenemos que resolver. Esto supone en definitiva poder describir los elementos del problema, en particular

a) ¿Cual es el objetivo final?
b) ¿Cuál es la situación inicial desde la que se parte?
c) ¿Como describir las diferentes situaciones o estados por los que podemos pasar?, y
d) ¿Qué operadores o acciones se pueden llevar a cabo en cada momento para cambiar las situaciones y como cambian?

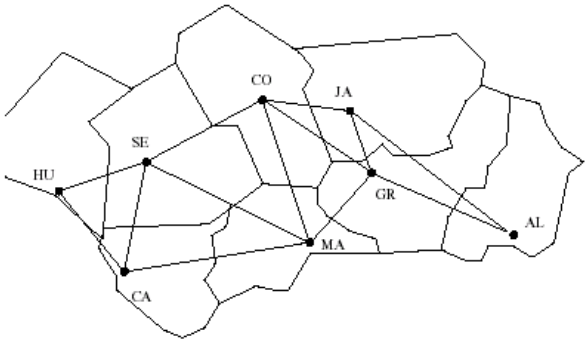
Un problema puede representarse por un triple (I, O, S) en el que I representa la situación inicial de la que partiríamos, O sería el conjunto de operadores que podrían definir nuevos subproblemas o actuar sobre cada uno de ellos, y S sería la solución final o el conjunto de soluciones que buscamos. Con esta representación, encontrar una solución para nuestro problema consiste en determinar una sucesión finita de aplicaciones de los operadores que van cambiando el problema inicial en subproblemas hasta llegar a la solución que buscamos.

Así, por medio de un proceso de abstracción, conseguimos ir descomponiendo un problema en subproblemas, por medio de esos operadores, que tendremos que definir en cada caso concreto. Estos subproblemas, en cada caso, adquieren la misma categoría del problema de partida, y es sobre cada uno de ellos donde aplicamos las técnicas de solución mas adecuadas en cada caso. Cuando, finalmente, encontremos la solución buscada, habremos resuelto el problema que teníamos planteado de partida.

Como resultará evidente, esta descomposición de un problema en subproblemas puede representarse mediante una estructura dada por un grafo dirigido, en la que los nodos representan problemas, y los descendientes de un nodo representan los subproblemas asociados al mismo, como ilustra la siguiente figura,




Antes de seguir, aclaremos este planteamiento con un clásico ejemplo, como es el del Viaje entre Ciudades: Nos encontramos en una capital andaluza (por ejemplo Granada), y deseamos ir a otra capital andaluza (sea Huelva), sabiendo que los autobuses sólo van de cada capital a sus vecinas:

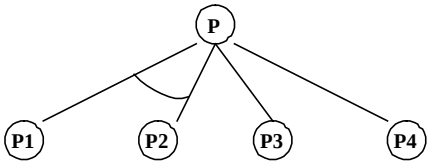



Como resultará obvio, en este problema tenemos ocho posibles subproblemas: Almeríaıa, Cádiz, Córdoba, Granada, Huelva, Jaén, Málaga y Sevilla; de entre ellos hemos seleccionado un estado inicial: Granada, y un estado final: Huelva. Los operadores a considerar serían:

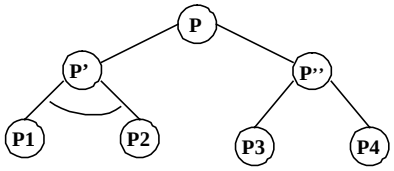
- Ir a Almería.
- Ir a Cádiz.
- Ir a Córdoba.
- Ir a Granada.
- Ir a Huelva.
- Ir a Jaén.
- Ir a Málaga.
- Ir a Sevilla.

La aplicación del operador “Ir a Granada” a un estado x, podría obtenerse de cualquier subproblema x que fuera, en este caso del ejemplo, una provincia vecina de Granada. Después de la aplicación de ese operador a la provincia, digamos, Córdoba, el nuevo subproblema resultante sería Granada.

Este marco general es en el que nos vamos a desenvolver a continuación, por lo que concretaremos algunos de los conceptos mas importantes que vamos a utilizar en lo que sigue, comenzando definir por la forma que usaremos para representar los grafos que emplearemos.
Partimos de que tenemos un problema genérico P = (I, O, S). Este problema, que como veremos, lo podremos entender en algunos casos como un juego del que nos interesa encontrar la mejor forma de jugarlo (la solución final), vamos a suponerlo representado por un grafo, y sobre este grafo es sobre el que iremos describiendo las nociones que nos interesan.
Así, por ejemplo, supongamos que el grafo de la figura




representa un problema P que puede resolverse bien mediante la resolución de los dos subproblemas P1 y P2, o bien mediante la de P3 o P4 aisladamente. Los grupos de subproblemas que deben resolverse para implicar una solución del nodo padre, se conectan mediante un arco que una las respectivas aristas (caso de P1 y P2 aquí). Introduciendo nodos fantasmas como en la siguiente figura,

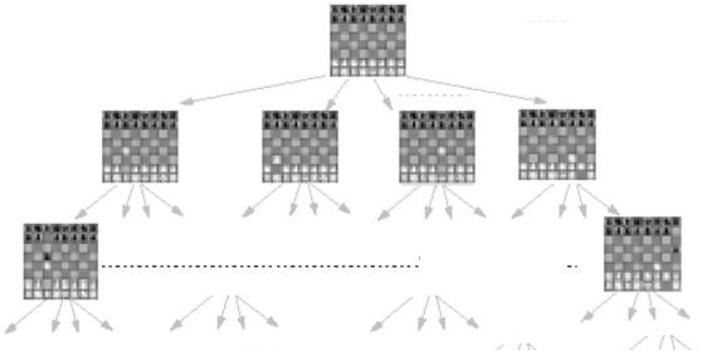



puede conseguirse que todos los nodos sean tales que sus soluciones requieran que haya que resolver todos sus descendientes, o bien que solo haya que resolver un descendiente. Los nodos del primer tipo se llaman nodos Y, y los del segundo nodos O. En esta ultima figura, P y P" son nodos O, mientras que P' es un nodo Y. Los nodos Y se suelen representarse con un arco conectando todas las aristas que salen de él. Por este motivo, a este tipo de grafos se les denomina Grafos Y/O.
A los nodos que tienen descendientes se les llama nodos padre y a los descendientes se les llama nodos hijo. A los nodos sin descendientes se les llama hoja o terminales. Estos últimos se clasifican en:

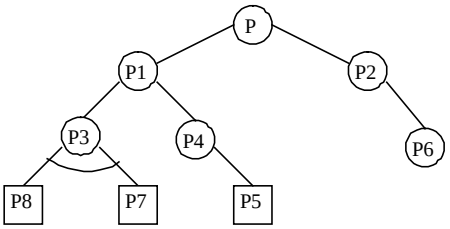
- Nodos terminales resolubles (enmarcados en un grafo dentro de un cuadrado cuando sea necesario).
- Nodos terminales no resolubles (enmarcados en un grafo dentro de un circulo cuando sea necesario).

La descomposición de un problema en diversos subproblemas se conoce con el nombre de Reducción del Problema. La Reducción de Problemas se usa frecuentemente en ámbitos como la demostración de teoremas, la integración simbólica, el análisis de calendarios industriales (métodos PERT o CPM), y conlleva el conocimiento del grafo, su estructura y sus reglas de representación, pero

no tenemos necesariamente que conocer su representación completa. Por eso, a este tipo de grafos se les llama Grafos Implícitos, porque conoceríamos como podrían representarse, digamos, gráficamente, pero no necesitamos hacerlo para conocerlos perfectamente.
El caso mas ilustrativo de lo que es un grafo implícito podemos encontrarlo en el celebre juego de ajedrez, al que es inmediato suponerle asociado un grafo Y/O implícito.



Podemos concebir el desarrollo de toda una partida, y para ello no necesitamos representar todo el grado que supondría el jugarla completamente: conocemos el grafo de esa partida de manera implícita.
Cuando se usa la reducción de problemas, dos problemas diferentes pueden generar un subproblema común. En este caso puede ser deseable tener solo un nodo representando a este subproblema, ya que esto podría significar tener que resolver ese subproblema solo una vez. Pero este hecho, aparentemente, intrascendente se traduce en que un grafo de estas características no tiene porque ser un árbol.
De hecho puede ser además que tales grafos tengan ciclos dirigidos. Sin embargo la presencia de un ciclo dirigido no implica en si misma la irresolubilidad del problema. Llamaremos grafo solución a un subgrafo de nodos resolubles que muestra que el problema puede resolverse.
Consideremos ahora el caso de un problema genérico P, descompuesto en subproblemas como muestra la figura,




y veamos como determinar si un árbol Y/0 dado representa o no un problema resoluble (la extensión al caso de grafos es inmediata).
Esta claro que, realizando un recorrido en post-orden del árbol Y/O, podemos determinar si un problema es resoluble o no. De aquí que el algoritmo correspondiente no sea mas que una extensión de aquel del recorrido en post-

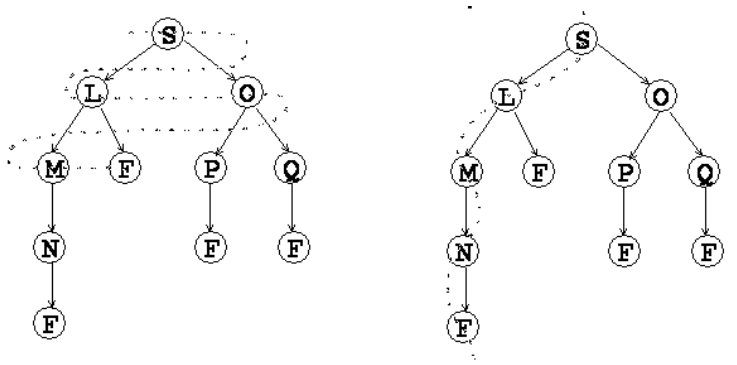
orden. En lugar de evaluar todos los hijos de un nodo, el algoritmo finaliza tan pronto como descubre que un nodo es irresoluble o resoluble. Esto reduce la cantidad de trabajo que el algoritmo tiene que hacer, sin que por ello la salida se afecte.

**PROCEDIMIENTO RESUELVE (T)**
{T es un árbol Y/O con raíz T. T ≠ 0}
**Comienzo**

```
Caso: T es un nodo terminal:   Si T es resoluble
                               Entonces Devolver (1)
                               Sino Devolver (0)
      : T es un nodo Y:            Para cada hijo S de T hacer
                               Si RESUELVE (S) = 0
                               Entonces Devolver (0)
                                        Devolver (1)
      : En otro caso:          Para cada hijo S de T hacer
                               Si RESUELVE (S) = 1
                               Entonces Devolver (1)
                                    Devolver (0)
```

**Fin**

A menudo, como hemos comentado, solo se dispone de forma implícita del árbol Y/O correspondiente a un problema. Nos interesará por tanto en cada caso definir una función que genere todos los hijos de un nodo ya generado. Entonces, dado el nodo raíz, tenemos que determinar un árbol solución (si existe) para el problema.
Los nodos del árbol pueden generarse primero en profundidad o primero en anchura, como gráficamente se ilustra en la siguiente figura a izquierda y derecha, respectivamente,

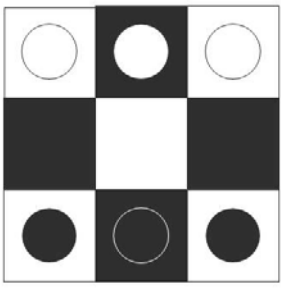



Ahora bien, como es posible que un árbol Y/O tenga profundidad infinita, si comenzáramos una generación del árbol primero en profundidad, generando por tanto todos los nodos en un camino infinito desde la raíz, podría ser que no llegáramos a determinar un subárbol solución (incluso aunque existiera). Esto puede evitarse restringiendo la búsqueda primero en profundidad a la generación del árbol Y/O dentro solo de una profundidad prefijada de valor d. Así, los nodos que a profundidad d sean no terminales, se determinan como irresolubles. En este caso,

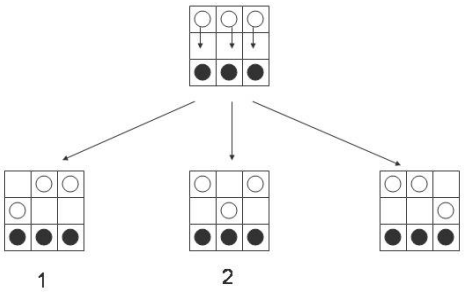
queda garantizado que la búsqueda primero en profundidad encontrará un subárbol solución, dado que hay una profundidad no mayor que d.
La búsqueda (o generación) primero en anchura no presenta este inconveniente. Ya que cada nodo puede tener solo un número finito de hijos, ningún nivel en el árbol Y/O puede tener un número infinito de nodos. De aquí que la generación primero en anchura del árbol Y/O garantice encontrar un subárbol solución, si es que este existe. Además, este procedimiento de generación debería generar un subárbol solución de mínima profundidad.
En cualquier caso la forma de generar los hijos de cada nodo se determinará, como hemos dicho, mediante una función especifica que garantice en definitiva la expansión total del árbol o grafo implícito que estemos considerando, o en su caso el subárbol o subgrafo que nos interese, si es que no tuviéramos necesidad de generarlo todo completamente (como podría pasar si quisiéramos parar en el momento en que hubiésemos encontrado una solución).

## Árboles para juegos

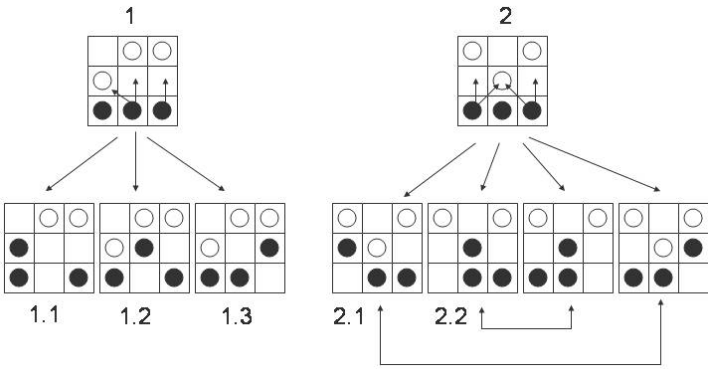
Una aplicación interesante de los grafos o árboles implícitos se encuentra en el desarrollo de juegos de estrategia (naipes, ajedrez, tres en raya, damas, etc.). Aunque solo sea desde el punto de vista de sus múltiples aplicaciones, el concepto de juego es uno de los más interesantes que podemos encontrar, y de ahí que en lo que sigue profundicemos en él antes de pasar a estudiar como desarrollar una partida, y por tanto como explorar el espacio de posibles soluciones.
Lo que por el momento nos interesa es ver que un juego, aunque de determinadas características como mas adelante veremos, se puede representar como un grafo. Para constatar esto, consideremos una versión reducida del ajedrez, en la que suponemos un tablero que solo tiene de dimensión 3 x 3, es decir con nueve casillas, en el que colocamos tres peones por cada color, que pueden moverse y comer de arreglo a las reglas normales de este juego:



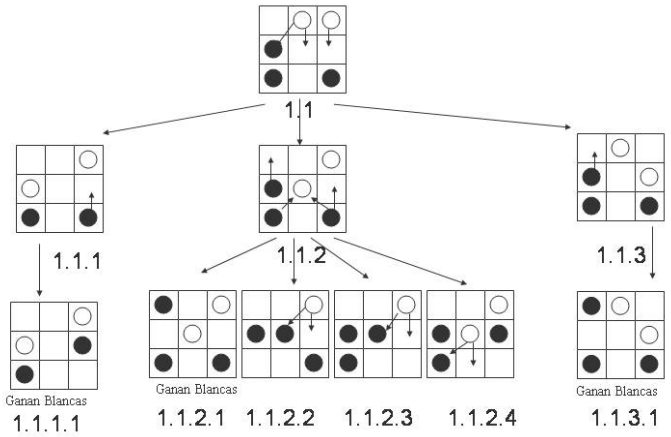
Si suponemos, como es habitual, que empieza moviendo el jugador que juega con los peones blancos, las posibilidades que se le presentan para hacer el primer movimiento de la partida, independientemente de la estrategia que quiera seguir, son las tres que se indican en la siguiente figura,




De estas posibilidades de movimiento, a continuación solo consideraremos las dos primeras (numeradas 1 y 2) ya que la tercera, por ahorro de recursos, no la iremos desplegando ya que es simétrica de la primera (1). Entonces, el primer movimiento que el otro jugador (de piezas negras) podría hacer es el que se describe en la figura siguiente,




Considerando ahora solo el desarrollo del juego a partir de la jugada 1.1, el primer jugador tendría las siguientes posibilidades,




Como es fácil de entender, la totalidad de posibilidades, describen el desarrollo de una partida, que de este modo queda conformada como un grafo.

De este modo jugar este juego (y cualquier otro en el sentido que aquí estamos considerando) significa navegar a través de todo ese grafo de acuerdo a las siguientes etapas, que conformarían el algoritmo del juego (o de búsqueda de una solución, es decir, de nuestra estrategia de juego),

1. Determinar el nodo (tablero) en el que nos encontramos
2. Determinar todos los movimientos posibles
3. Seleccionar un movimiento
4. Realizar ese movimiento
5. ¿Hemos ganado?
   Si es positiva la respuesta, Entonces Terminar.
6. Nuestro oponente mueve.
7. ¿Gana nuestro oponente?
   Si es positiva la respuesta, Entonces Terminar.
8. Ir a la etapa 1.

Pero no es tan sencillo. Efectivamente, pensemos que en el caso del ajedrez, para el primer movimiento de las blancas tenemos

- 20 posibilidades (8 peones x 2 posiciones + 4 posiciones de los caballos)

Del mismo modo, para el primer movimiento de las negras, tenemos

- 20 posibilidades (8 peones x 2 posiciones + 4 posiciones de los caballos)

Esto da una idea del número inmenso de posibilidades (nodos) que tendría un hipotético grafo para este juego, y por tanto de la importancia que tiene el disponer de técnicas de exploración efectivas y eficaces. De este modo el concepto de juego aparece asociado al de algoritmo de exploración de grafos, y de ahí la necesidad de conocer bien lo que se entiende por juego, así como de las formas que hay de jugarlos, porque de ahí obtendremos información mas que importante para conocer y diseñar nuevos algoritmos de búsqueda sobre grafos, que en definitiva es lo que aquí mas nos importa.

Para definir formalmente un juego necesitamos un concepto previo, que es el de árbol topológico. Un árbol topológico, o árbol del juego, es una colección finita de vértices, conectados por arcos que constituyen una figura conexa que no incluye curvas cerradas simples. Así, se deduce que dados dos vértices cualesquiera A y B, solamente hay una secuencia de arcos y vértices que unen A y B. Por tanto si $\Gamma$ es un árbol topológico con un vértice distinguido (raíz) A, decimos que el vértice C sigue al vértice B si la sucesión de arcos que une A con C pasa a través de B. Se dice que C sigue inmediatamente a B, si C sigue a B y, además, hay un arco que une B con C. Un vértice X se dice terminal si no hay ningún vértice que lo siga. En este punto es importante que el lector asocie e identifique definiciones comunes que nos han aparecido ya en otras ocasiones (vértice raíz, terminal, árbol,…)

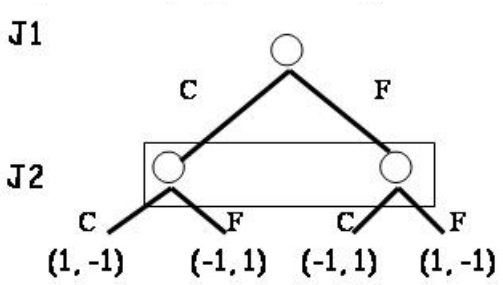
Un juego n-personal en forma extensiva se define como,

a) Un árbol topológico $\Gamma$, con un vértice distinguido A que suele llamarse punto de partida de $\Gamma$.
b) Una función, llamada función de pagos, que asigna un n-vector a cada vértice terminal de $\Gamma$.

c) Una partición de los vértices no terminales de $\Gamma$ en n+1 conjuntos, llamados conjuntos de jugadores, $S_0, S_1, ..., S_n$ .
d) Una distribución de probabilidad definida en cada vértice de $S_0$. Esta distribución de probabilidad solo existirá cuando haya movimientos de azar en el juego que se este considerando.
e) Para cada i = 1, ..., n, una subpartición de $S_i$ en subconjuntos $S_i^j$, llamados conjuntos de información, tales que dos vértices en el mismo conjunto de información tiene el mismo número de vértices seguidores inmediatos, y ningún vértice puede seguir a ningún otro en un mismo conjunto de información, es decir los vértices en estos conjuntos son indistinguibles para el correspondiente jugador.
f) Para cada conjunto de información $S_i^j$ existe un conjunto de índices $I_i^j$, y una aplicación uno a uno de $I_i^j$ en el conjunto de los vértices inmediatos seguidores de cada vértice de $S_i^j$ .

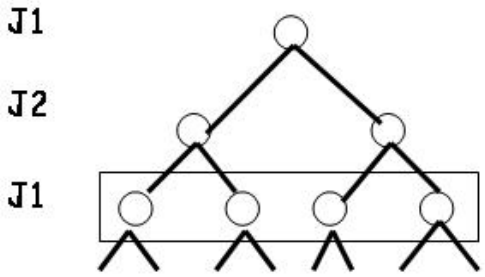
Así los elementos del juego se interpretan de la siguiente forma. La condición a) establece el punto de comienzo. La b) da la forma en que se van a obtener recompensas. La c) divide los movimientos en movimientos de azar (i = 0) y movimientos personales (i = 1, ..., n) correspondientes a los n jugadores. La d) define un esquema de aleatorización en cada movimiento de azar. La e) divide los movimientos de los jugadores en conjuntos de información, de modo que cada uno de ellos puede conocer en que conjunto de información se encuentra, pero desconoce en que vértice esta.

Por ejemplo, el siguiente juego considera todos estos elementos. Supongamos que el jugador 1 (J1) escoge cara (C) o cruz (F). Entonces el jugador 2 (J2), sin saber lo que el 1 ha escogido, también escoge entre cara o cruz. Si ambos jugadores eligen de forma desigual, entonces el jugador 2 le gana una unidad al 1. En cualquier otro caso, el primer jugador le ganara una unidad al 2. Este juego puede representarse por la siguiente figura, con significado claro.




Supongamos ahora que en la jugada I el primer jugador elige un número x entre 1 y 2; en la jugada II, el segundo jugador informado del número elegido por J1, elige a su vez un número y entre 1 y 2; en la jugada III, el primer jugador sin conocer la elección hecha por J2 en la jugada II, y habiendo olvidado el número x elegido en la I, elige un número z entre 1 y 2. Una vez elegidos los números x, y, z, J2 paga a J1 la cantidad M(x,y,z) siguiente,

M(1,1,1) = -2 M(1,1,2) = -1 M(2,1,1) = 5 M(1,2,2) = -4
M(1,2,1) = 3 M(2,1,2) = 2 M(2,2,1) = 2 M(2,2,2) = 6




Como J1 tiene dos conjuntos de información y otros tantos J2, resulta que sus estrategias son (1,1), (1,2), (2,1) y (2,2), para J1, y [1,1], [1,2], [2,1] y [2,2], para el segundo. Así, además en este caso, el juego puede representarse por una matriz como la siguiente,

| | **[1,1]** | **[1,2]** | **[2,1]** | **[2,2]** |
|---|---|---|---|---|
| **(1,1)** | -2 | -2 | 3 | 3 |
| **(1,2)** | -1 | -1 | -4 | -4 |
| **(2,1)** | 5 | 2 | 5 | 2 |
| **(2,2)** | 2 | 6 | 2 | 6 |

Es importante destacar que si quisiéramos encontrar una forma de jugar en la que ambos jugadores se encontraran cómodos, podríamos emplear la conocida estrategia maximín, de acuerdo con la cual, y por motivos obvios, el primer jugador escogería aquella estrategia que le proporcionara el valor

$$\mathrm{Max}_i \, \mathrm{Min}_j \, a_{ij}$$

mientras que J2 tendría que jugar aquella estrategia que le proporcionara el valor,

$$\mathrm{Min}_j \, \mathrm{Max}_i \, a_{ij}$$

pero el problema que aparece es que generalmente ambos valores no coinciden, por lo que hay que pasar a esquemas de resolución de juegos más complejos y que no interesan para los objetivos de este tema.

Otro ejemplo que podemos considerar, conectando ya con la forma efectiva de recorrer los árboles de juegos, es el juego de los palillos (también llamado de Nim). Este juego se juega por dos jugadores A y B. El juego queda descrito por un panel que inicialmente contiene n palillos. Los jugadores A y B hacen alternativamente sus movimientos, empezando a jugar A. Un movimiento legal consiste en eliminar 1, 2 o 3 palillos del panel. Sin embargo, un jugador no puede quitar más palillos de los que hay en el panel. El jugador que quita el último palillo pierde el juego y el otro gana. La configuración del panel esta completamente especificada en cualquier instante por el número de palillos que quedan, y en cualquier momento el status del juego se determina por la configuración del panel, junto con el jugador que le toca jugar a continuación. Una configuración terminal es aquella que representa una situación de ganar, de perder o de empate. Todas las demás configuraciones son no terminales. En el juego de Nim

solo hay una configuración terminal: Cuando no quedan palillos en el panel. Esta configuración es de ganancia para A, si B hizo el último movimiento o viceversa. El juego de Nim no puede terminar en empate.

Una sucesión C(1),...,C(m) de configuraciones se llama valida si

1) C(1) es la configuración inicial del juego
2) C(i), $0 < i < m$, son configuraciones no terminales, y
3) C(i+1) se obtiene de C(i) mediante un movimiento legal hecho por el jugador A si i es impar, y por el jugador B si i es par. Se supone que hay un número finito de movimientos legales.

Una sucesión valida C(1),...,C(m) de configuraciones, en la que C(m) es una configuración terminal, se llama un caso del juego. La longitud de la sucesión C(1),...,C(m) es m. Un Juego Finito es un juego para el que no existen sucesiones validas de longitud infinita.

Todos los posibles casos de un juego finito pueden representarse por lo que se llama el árbol del juego. En ese árbol, cada nodo representaría una configuración, y el nodo raíz seria la configuración inicial C(1). Las transiciones desde un nivel al siguiente se hacen según mueva el primer o el segundo jugador. En el juego de Nim las transiciones desde un nivel impar representarían los movimientos de A, mientras que todas las demás transiciones serían los movimientos de B.

Para facilitar la visualización, suelen usarse nodos cuadrados para representar configuraciones en las que le toca mover a A. En las demás configuraciones se usan nodos circulares. Las configuraciones terminales se representan por nodos hoja y suelen etiquetarse con el nombre del jugador que gana cuando se alcanza esa configuración.

El grado de cualquier nodo en el árbol del juego es igual, a lo más, al número de movimientos legales distintos que haya. En el juego de Nim, a lo más hay tres por configuración. Por definición, el número de movimientos legales en cualquier configuración es finito. La profundidad de un árbol de juego es la longitud del más largo caso del juego.

Los árboles de juego son útiles para determinar el siguiente movimiento que un jugador debe hacer. Partiendo de la configuración inicial representada por la raíz, el jugador A se enfrenta con tener que elegir entre varios posibles movimientos. Supuesto que A quiera ganar el juego, A debería elegir el movimiento que maximice su posibilidad de ganar. Se puede usar una función de evaluación E(X) que asigne un valor numérico a cada configuración X. Esta función dará la recompensa para el jugador A asociada a la configuración X.

Para un juego como el Nim con pocos nodos, basta definir E(X) sobre las configuraciones terminales. Así, se podría dar

$$E(x) = 1 \text{ si X es una configuración ganadora para A}$$
$$= -1 \text{ si X es una configuración perdedora para A}$$

Usando esta función de evaluación, ahora lo que queremos es determinar cual de las posibles configuraciones que el jugador A tiene como posibles (supongamos que sean b, c y d) debe elegir. Claramente, la elección será aquella cuyo valor sea el del Max{V(b), V(c), V(d)}, donde V(x) es el valor de la configuración x. Para nodos hoja x, V(x) se toma como E(x). Para todos los demás nodos x, sea $d \geq 1$ el grado de

x, y sea C(1),...,C(d) la configuración representada por los hijos de x. Entonces V(x) se define como

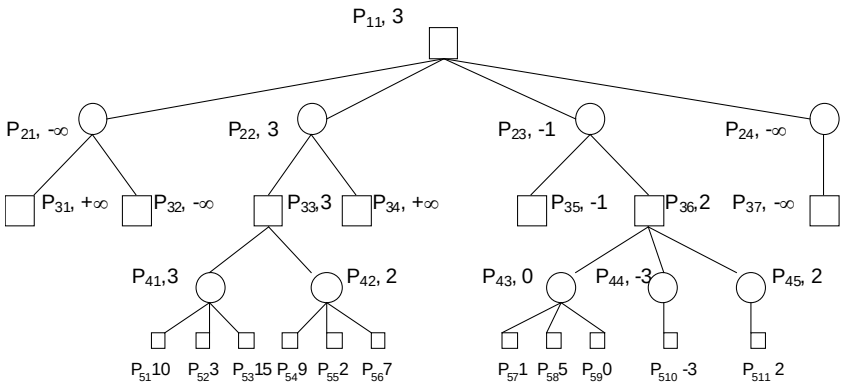
$$V(x) = \text{Max}_{1\le i\le d} \{V[C(i)]\} \text{ si } x \text{ es un nodo cuadrado} \quad (1)$$

$$= \text{Min}_{1\le i\le d} \{V[C(i)]\} \text{ si } x \text{ es un nodo circular}$$

La justificación de (1) es simple, y además completamente similar a la anteriormente explicada estrategia maximín. Tan es así, que (1) inspira una forma de exploración de grafos que recibe exactamente ese mismo nombre. En efecto, si x es un nodo cuadrado, entonces estará en un nivel impar, y seria el turno para que moviera A desde aquí, si alguna vez el juego alcanzara este nodo. Como A quiere ganar, A se moverá a un nodo hijo con valor máximo. En el caso de que x sea un nodo circular, debe estar en un nivel par, y si el nodo alguna vez alcanza ese nodo entonces será el turno para que mueva B. Como B no juega a ganar, hará un movimiento que minimice la posibilidad de que gane A. En este caso, la siguiente configuración será la que del valor

$$\text{Min } \{V[C(i)], 1\le i\le d \}$$

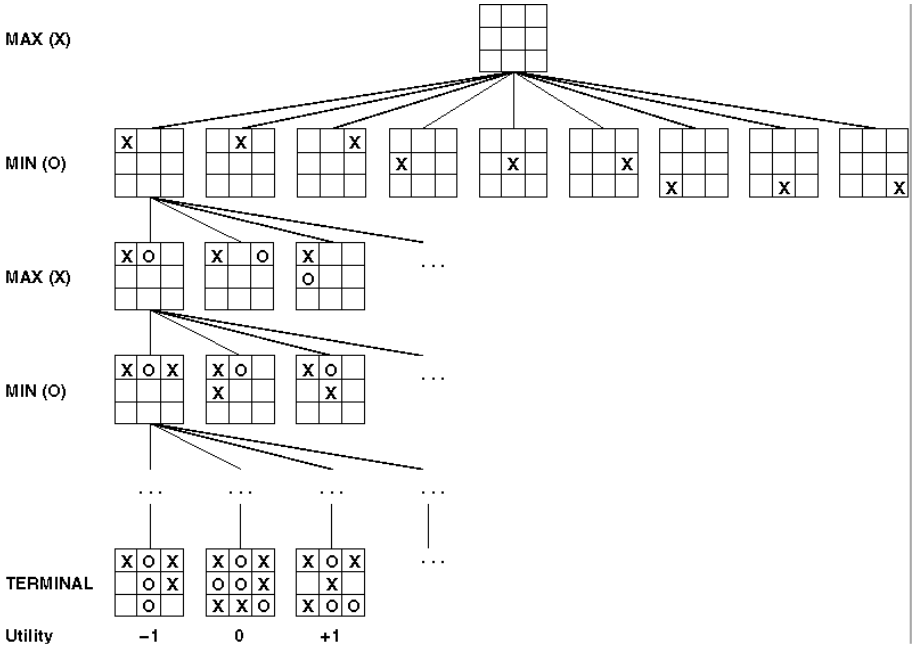
La ecuación (1) define lo que se denomina procedimiento minimax para determinar el valor de la configuración x.
Ilustremos esto sobre un juego hipotético como el que se representa en la figura siguiente, y al que nos referiremos como Juego Marco



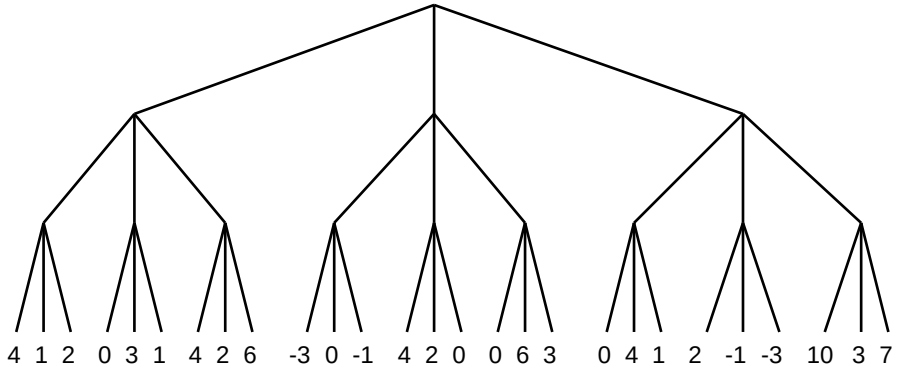

En ella P(11) representa una configuración arbitraria desde la que A tiene que efectuar un movimiento. Los valores de los nodos hijo se obtienen evaluando la función E(x). El valor de P(11) se obtiene comenzando en los nodos del nivel 4, y usando la ecuación (1) para calcular sus valores. Como el nivel 4 es un nivel con nodos circulares, todos los valores desconocidos en este nivel pueden obtenerse tomando el mínimo de los valores de los hijos. A continuación, pueden calcularse los valores en los niveles 3, 2 y 1 en ese mismo orden. El valor resultante para P(11) es 3, lo que significa que partiendo de P(11) lo mejor que A puede esperar que le pase es alcanzar una configuración de valor 3.
Nótese que aunque algunos nodos tienen valores superiores a 3, nunca serán alcanzados ya que los contramovimientos de B lo impedirán. Por ejemplo, si A se

mueve a P(21) esperando ganar el juego en P(31), A quedaría desbancado cuando viera que B hacia el contramovimiento a P(32), resultando una perdida para A. Dada la función de evaluación de A y el árbol del juego de la figura anterior, el mejor movimiento de A es hacia la configuración P(22). Habiendo hecho este movimiento, aun podría ser que el juego no alcanzara la configuración P(52) ya que, en general, B podría usar una función de evaluación distinta, lo que podría proporcionar valores diferentes a las distintas configuraciones. En cualquier caso, el procedimiento minimax puede usarse para determinar el mejor movimiento que un jugador puede hacer según su función de evaluación.

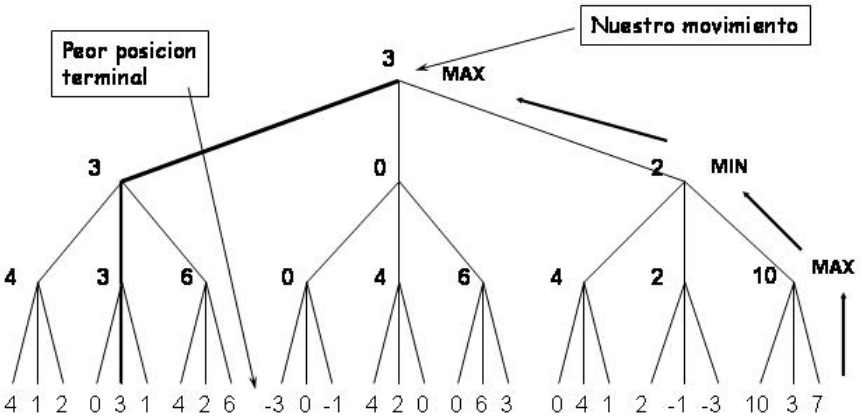
Por ejemplo, en el celebro juego del Tres en Raya, el árbol del juego sería como el que se representa en la siguiente figura (que por razones obvias solo muestra una parte del mismo),




Por ejemplo, en el siguiente árbol de un cierto juego,




el desarrollo del procedimiento minimax que acabamos de describir se realizaría como se ilustra la siguiente figura




Sin embargo este procedimiento no es interesante en juegos con árboles pequeños. Donde realmente es potente es en juegos como el ajedrez o las damas en los que el árbol es tan grande que es imposible generarlo completamente. Así en juegos de características similares, es decir, con árboles de dimensiones formidables, y por tanto en los que suelen darse fenómenos conocidos con el nombre de explosión combinatoria (por la cantidad de nodos que habría que evaluar), la decisión de como efectuar un movimiento se realiza mediante la exploración de unos cuantos niveles. La función de evaluación E(X) se usa para obtener los valores de los nodos hoja del subárbol generado, y entonces se usa la ecuación (1) para obtener los valores de los restantes nodos, y consiguientemente determinar el próximo movimiento.

Supongamos que el jugador A es un computador e intentemos escribir un algoritmo que pueda usar A para calcular V(X). Esta claro que el procedimiento para calcular V(X) puede usarse también para determinar el siguiente movimiento que debería hacer A. Puede obtenerse un procedimiento recursivo limpio para calcular V(X) usando minimax si disponemos la definición de minimax en la siguiente forma

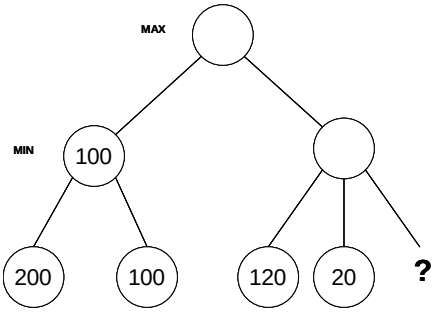
$$V'(X) = e(X) \qquad \text{si X es una hoja del subárbol generado} \qquad (2)$$

= Max {-V'[C(i)], 1≤i≤d}  si X no es una hoja del subárbol generado y los C(i), son los hijos de X

donde es evidente que e(X) = E(X) si X es una posición desde la que A se tiene que mover, pero e(X) = -E(X) en caso contrario.
Partiendo de una configuración X desde la que A tiene que mover, fácilmente puede demostrarse que la ecuación (2) calcula V'(X) = V(X) como (1), ya que los valores de todos los nodos en niveles en los que mueve A, son los que proporciona (1), y los valores de los otros niveles son los opuestos de los que daba (1). Si queremos calcular V'(X) realizando a lo sumo l movimientos hacia adelante, por e(X) notamos la función de evaluación de A, como hemos comentado, se supone por conveniencia que partiendo de una configuración X, los movimientos legales del juego solo permiten transiciones a las configuraciones C(1),...,C(d) si X no es una configuración terminal, el procedimiento recursivo para evaluar V'(X) basado en la anterior formula (2) es el que se recoge en el siguiente algoritmo,

**PROCEDIMIENTO VE**(X,l)
Comienzo
    Si X es terminal o l = 0 Entonces Devolver (e(X))
    Temp = -VE(C(1), l-1)
    Para i = 2 hasta d hacer
        Temp = Max(Temp, -VE(C(i), l-1))
    Devolver (Temp)
Fin

Este algoritmo evalúa V'(X) generando solo l niveles del árbol del juego comenzando tomando X como su raíz. Se puede verificar fácilmente que este algoritmo recorre el subárbol deseado del árbol del juego es post-orden. Es necesario ese recorrido en post-orden ya que el valor de un nodo puede determinarse solo después de que hayan sido evaluados sus hijos.
Sobre el anterior Juego Marco, una llamada a este algoritmo con X = P(11) y l = 4 produciría la generación completa del árbol del juego. Los valores de las diversas configuraciones se determinarían en el orden P(31), P(32), P(21), P(51), P(52), P(53), P(41), P(54), P(55), P(56), P(42),...,P(37), P(24) y P(11).
Consideremos de nuevo el árbol del Juego Marco. Después de haber calculado V[P(41)], se sabe que V[P(33)] es al menos V[P(41)] = 3. A continuación, cuando se ve que V[P(55)] es 2, sabemos que V[P(42)] es a lo más igual a 2. Como P(33) es una posición Max, V[P(42)] no puede afectar a V[P(33)]. Independientemente de los valores de los restantes hijos de P(42), el valor de P(33) no se determina por V[P(42)] ya que V[P(42)] no puede ser mas que V[P(41)].
Más claramente, supongamos la situación de la siguiente figura, asociada a la evaluación de los valores de los nodos de cierto grafo,



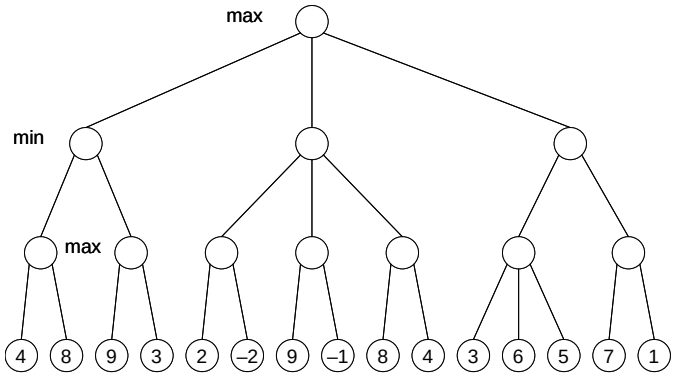

en la que hemos obtenido que el primer nodo min tiene de valor 100, y el signo de interrogación se corresponde a la pregunta de si deberíamos evaluar el nodo al que va asociado ese símbolo. Bien, lo que pasa es que después de saber que el primer nodo min tiene de valor 100, y obtener en el subárbol de la derecha, como valor en el último nivel 20, tenemos la certeza de que en el nivel superior siempre se seleccionará el valor 20, o algún hipotético menor valor. Pero en cualquier caso, en el siguiente nivel siempre se escogería el nodo de la izquierda, con valor 100, por lo que en estas circunstancias no interesa proseguir la generación de nodos. Así en el momento en que obtenemos el valor 20, la generación de nodos debería detenerse. Esta observación puede establecerse mas formalmente como la siguiente regla, que ilustra lo que se conoce como Poda alfa:

El valor alfa de una posición max se define como el mínimo valor posible para esa posición. Si se observa que el valor de una posición min es menor o igual que el valor alfa de su padre, entonces podemos parar la generación de los restantes hijos de esa posición min. La finalización de la generación de nodos bajo esta regla se conoce con el nombre de poda alfa.

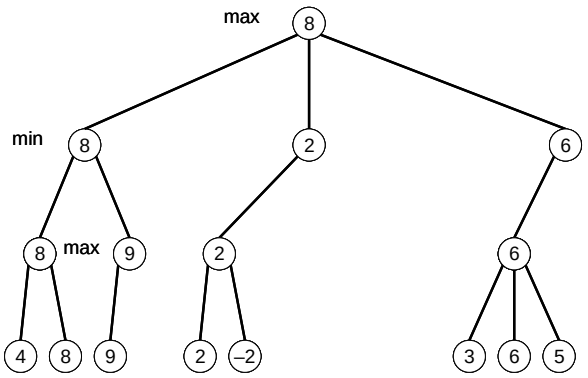
Una regla análoga puede establecerse para las posiciones min: El valor beta de una posición min es el máximo valor posible para esa posición. Si se observa que el valor de una posición max es mayor o igual que el valor beta de su nodo padre, se puede parar la generación de los restantes hijos de esa posición max. El término de la generación de nodos bajo esta regla se llama poda beta.

En el Juego Marco, cuando se calcula V[P(35)], el valor beta de P(23) es -1. La generación P(57), P(58) y P(59) da V[P(43)] = 0. Así, V[P(43)] es mayor o igual que el valor beta de P(23), y podemos terminar la generación de hijos de P(36).

Las dos reglas anteriores pueden combinarse juntas para dar lo que se denomina Poda Alfa-Beta. Para comprobar el interés de este tipo de poda en la generación de árboles o grafos, lo ilustraremos con un sencillo ejemplo. La siguiente figura se corresponde con un árbol generado sin tener en cuenta la poda alfa-beta




Es inmediato comprobar que si consideramos la generación de este árbol con la poda alfa-beta, la generación real se correspondería con la de la siguiente figura




que deja claramente de manifiesto el ahorro que supone su aplicación.

Para introducir la poda alfa-beta en el algoritmo VE es necesario establecer esta regla en los términos de la ecuación (2). Bajo el esquema de la ecuación (2), todas las posiciones son posiciones max ya que los valores de las posiciones min de la ecuación (1) se han multiplicado por -1.

La regla de la poda alfa-beta se reduce entonces a lo siguiente: Llamemos B-valor al mínimo valor que una cierta posición puede tener. Para cualquier posición X, sea B el B-valor de su padre y D = -B. Entonces si tenemos que el valor de X es mayor o igual que D, podemos terminar la generación de los restantes hijos de X. La incorporación de esta regla en el anterior algoritmo VE es sencilla, y produce el algoritmo VEB siguiente, que tiene un parámetro adicional D que es el valor negativo del B-valor del padre de X,

**PROCEDIMIENTO VEB**(X,l,D)

{Determina V'(X) como en la ecuación (2) usando la regla B y efectuando solo l movimientos hacia adelante. Las restantes hipótesis y notaciones son las mismas que en el algoritmo VE}

Comienzo

Si X es terminal o l = 0 Entonces Devolver (e(X))

```
Temp = -VEB(C(1), l-1, ∞)
Para i = 2 hasta d hacer
        Si Temp ≥ D Entonces Devolver (Temp)
Temp = Max(Temp, -VEB(C(i), l-1, -Temp))
Devolver (Temp)

Fin
```

Si Y es la posición desde la que ha de moverse A, entonces la llamada inicial a VEB(Y,l, ∞) calcula correctamente V'(Y) con un movimiento l hacia adelante. Después puede efectuarse la poda del árbol del juego teniendo en cuenta que el B-valor de un nodo X da una cota inferior sobre el valor que un nieto de X debe tener para que el valor de X se afecte.

En lo que concierne a la eficiencia de este tipo de algoritmos, debido a la naturaleza de los problemas que tratan se entenderá fácilmente que es extraordinariamente difícil hacer afirmaciones precisas acerca de que fracción de nodos de un árbol de juego se generaran. No obstante, los siguientes resultados debidos a Knuth y Moore (y que pueden consultarse en E. Horowitz y S. Sahni: Fundamentals of Computer Algorithms. Comp Science Press, 1978) pueden arrojar un poco de luz al respecto

**Definición.** Un árbol de juego uniforme de grado d y altura h es un árbol de juego en el que cualquier nodo en los niveles 1,2, ..., h-1 tiene exactamente d hijos. Además, cualquier nodo en el nivel h es terminal. Un árbol de juego uniforme aleatorio es un árbol de juego uniforme en el que los nodos terminales tienen valores aleatorios independientes.

**Teorema.** El número esperado T(d,h) de posiciones terminales examinadas por el procedimiento alfa-beta sin podas profundas (el procedimiento VEB), en un árbol de juego uniforme aleatorio de grado d y altura h, es menor que $c(d)r(d)^h$, donde r(d) es el mayor autovalor de la matriz $M_d$ cuyos términos están dados por

$$M_d(i,j) = (\mathbf{C}_{(i-1)+(j-1)/d,\,(i-1)})^{-1/2} \qquad 1 \le i \le d \text{ y } 1 \le j \le d$$

siendo c(d) una constante apropiada.

**Teorema.** T(d,h) para un árbol de juego uniforme aleatorio de grado d y altura h+1 satisface la igualdad,

$$\lim_{h\to\infty} T(d,h)^{1/h} = r(d)$$

donde

$$c_1 d (\log d)^{-1} \le r(d) \le c_2 d (\log d)^{-1}$$

para ciertas constantes positivas $c_1$ y $c_2$.