O implementare eficientă a principiului Minimax: Algoritmul Alpha-Beta

Tehnica pe care o vom examina, în cele ce urmează, este numită în literatura de specialitate alpha-beta prunning ("alpha-beta retezare"). Atunci când este aplicată unui arbore de tip minimax standard, ea va întoarce aceeaşi mutare pe care ar furniza-o şi Algoritmul Minimax, dar într-un timp mai scurt, întrucât realizează *o retezare a unor ramuri ale arborelui care nu pot influenţa decizia finală*.

Principiul general al acestei tehnici constă în a considera un nod oarecare *n* al arborelui, astfel încât jucătorul poate alege să facă o mutare la acel nod. Dacă acelaşi jucător dispune de o alegere mai avantajoasă, *m*, fie la nivelul nodului părinte al lui *n*, fie în orice punct de decizie aflat mai sus în arbore, atunci *n* nu va fi niciodată atins în timpul jocului. Prin urmare, de îndată ce, în urma examinării unora dintre descendenţii nodului *n*, ajungem să deţinem suficientă informaţie relativ la acesta, îl putem înlătura.

Ideea tehnicii de alpha-beta retezare este aceea de a găsi o mutare "suficient de bună", nu neapărat cea mai bună, dar suficient de bună pentru a se lua decizia corectă. Această idee poate fi formalizată prin introducerea a *două limite*, *alpha* şi *beta*, reprezentând limitări ale valorii de tip minimax corespunzătoare unui nod intern.

Semnificaţia acestor limite este următoarea: *alpha* este *valoarea minimă* pe care este deja garantat că o va obţine MAX, iar *beta* este *valoarea maximă* pe care MAX poate spera să o atingă. Din punctul de vedere al jucătorului MIN, beta este valoarea cea mai nefavorabilă pentru MIN pe care acesta o va atinge. Prin urmare, *valoarea efectivă* care va fi găsită se află *între alpha şi beta*.

Valoarea alpha, asociată nodurilor de tip MAX, nu poate niciodată să descrească, iar valoarea beta, asociată nodurilor de tip MIN, nu poate niciodată să crească.

Dacă, spre exemplu, valoarea alpha a unui nod intern de tip MAX este 6, atunci MAX nu mai trebuie să ia în consideraţie nici o valoare internă mai mică sau egală cu 6 care este asociată oricărui nod de tip MIN situat sub el. Alpha este scorul cel mai prost pe care îl poate obţine MAX, presupunând că MIN joacă perfect. În mod similar, dacă MIN are valoarea beta 6, el nu mai trebuie să ia în consideraţie nici un nod de tip MAX situat sub el care are valoarea 6 sau o valoare mai mare decât acest număr.

Cele două reguli pentru încheierea căutării, bazată pe valori alpha şi beta, pot fi formulate după cum urmează:

1. Căutarea poate fi oprită dedesubtul oricărui nod de tip MIN care are o valoare beta mai mică sau egală cu valoarea alpha a oricăruia dintre strămoşii săi de tip MAX.
2. Căutarea poate fi oprită dedesubtul oricărui nod de tip MAX care are o valoare alpha mai mare sau egală cu valoarea beta a oricăruia dintre strămoşii săi de tip MIN.

Dacă, referitor la o poziţie, se arată că valoarea corespunzătoare ei se află în afara intervalului alpha-beta, atunci această informaţie este suficientă pentru a şti că poziţia respectivă nu se află de-a lungul *variaţiei principale*, chiar dacă nu este cunoscută valoarea exactă corespunzătoare ei. Cunoaşterea valorii exacte a unei poziţii este necesară numai atunci când această valoare se află între alpha şi beta.

Din punct de vedere formal, putem defini o *valoare de tip minimax* a unui nod intern, P, V(P, alpha, beta), ca fiind "*suficient de bună*" dacă satisface următoarele cerinţe:

V(P, alpha, beta) < alpha, dacă V(P) < alpha (1)

V(P, alpha, beta) = V(P), dacă alpha ≤ V(P) ≤ beta

V(P, alpha, beta) > beta, dacă V (P) > beta,

unde prin V(P) am notat valoarea de tip minimax corespunzătoare unui nod intern.

Valoarea exactă a unui nod-rădăcină P poate fi întotdeauna calculată prin setarea limitelor după cum urmează:

V(P, -∞, +∞) = V(P).

Fig. 3.2 ilustrează acţiunea Algoritmului Alpha-Beta în cazul arborelui din Fig. 3.1. Aşa cum se vede în figură, unele dintre valorile de tip minimax ale nodurilor interne sunt aproximative. Totuşi, aceste aproximări sunt suficiente pentru a se determina în mod exact valoarea rădăcinii. Se observă că Algoritmul Alpha-Beta reduce complexitatea căutării de la 8 evaluări statice la numai 5 evaluări de acest tip:

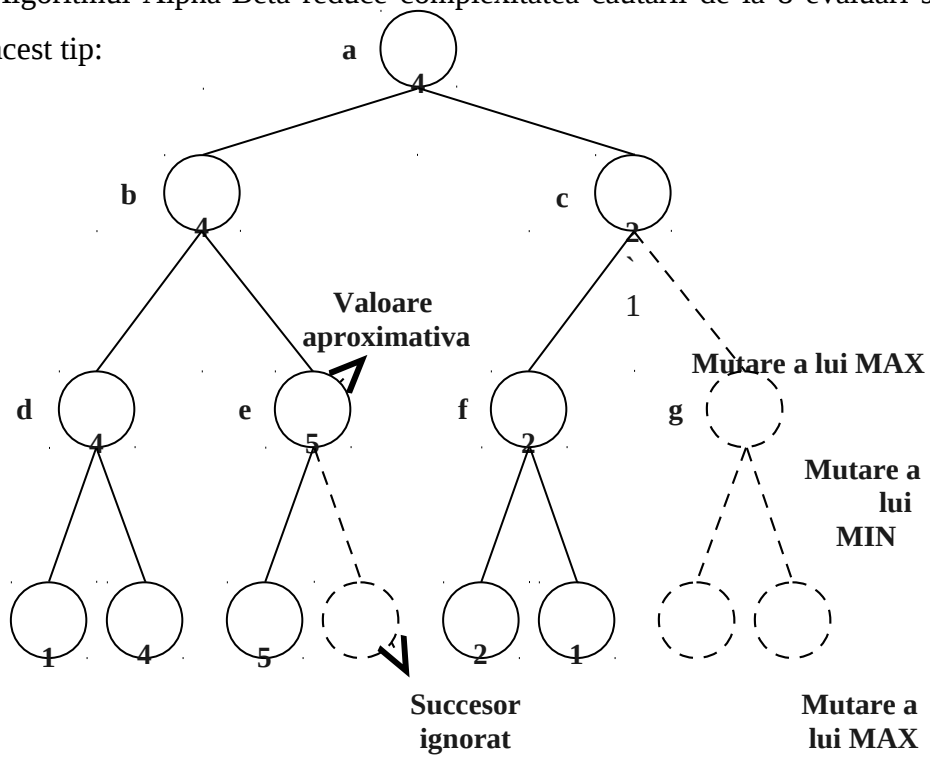

Fig. 3.2

Corespunzător arborelui din Fig. 3.2 procesul de căutare decurge după cum urmează:

3. Începe din poziţia *a*.

4. Mutare la *b*.

5. Mutare la *d*.

6. Alege valoarea maximă a succesorilor lui *d*, ceea ce conduce la V(*d*) = 4.

7. Întoarce-te în nodul *b* şi execută o mutare de aici la *e*.

8. Ia în consideraţie primul succesor al lui *e* a cărui valoare este 5. În acest moment, MAX, a cărui mutare urmează, are garantată, aflându-se în poziţia *e*, cel puţin valoarea 5, indiferent care ar fi celelalte alternative plecând din *e*. Această informaţie este suficientă pentru ca MIN să realizeze că, la nodul *b*, alternativa *e* este inferioară alternativei *d*. Această concluzie poate fi trasă fără a cunoaşte valoarea *exactă* a lui *e*. Pe această bază, cel de-al doilea succesor al lui *e* poate fi neglijat, iar nodului *e* i se poate atribui valoarea *aproximativă* 5.

Căutarea de tip alpha-beta retează nodurile figurate în mod discontinuu. Ca rezultat, câteva dintre valorile intermediare nu sunt exacte (nodurile *c*, *e*), dar aproximările făcute sunt suficiente pentru a determina atât valoarea corespunzătoare rădăcinii, cât şi variaţia principală, în mod exact.

Un alt exemplu de aplicare a Algoritmului Alpha-Beta este cel din Fig. 3.4. Astfel, corespunzător spaţiului de stări din Fig. 3.3, alpha-beta retezarea produce arborele de căutare din Fig. 3.4, în care stările fără numere nu sunt evaluate.

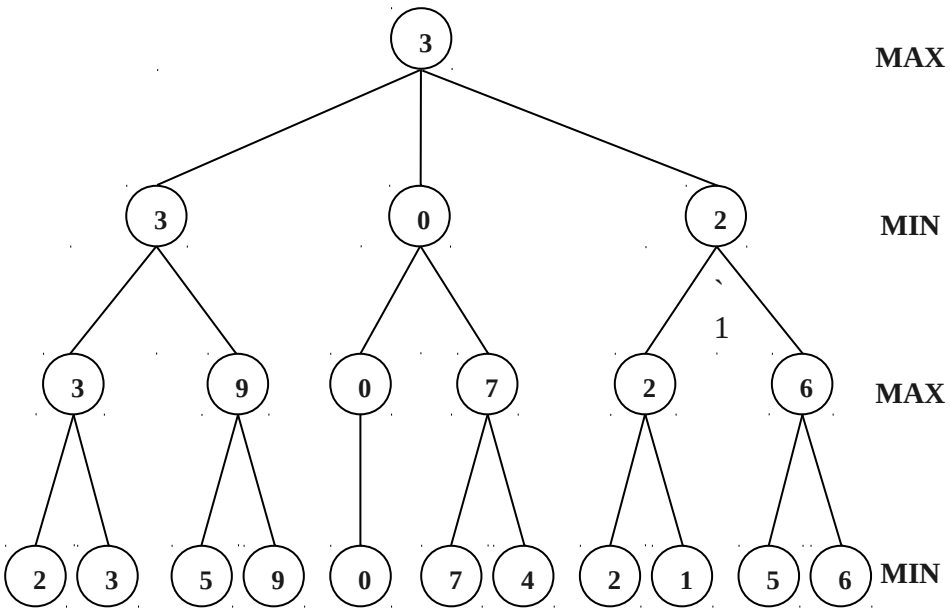

Fig. 3.3

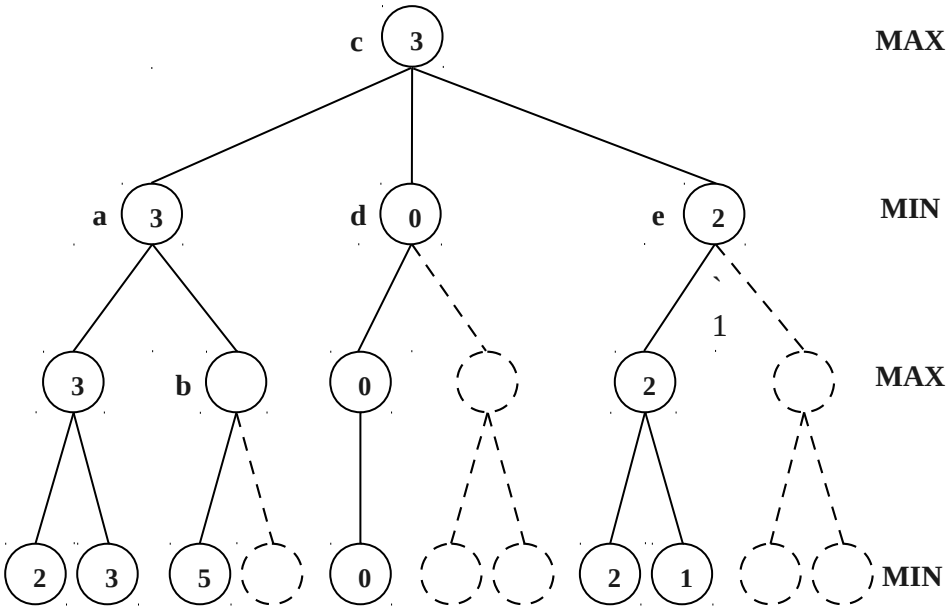

Fig. 3.4

<u>În Fig. 3.4</u>: *a* are $\beta = 3$ (valoarea lui *a* nu va depăşi 3);

b este β - retezat, deoarece $5 > 3$;

c are $\alpha = 3$ (valoarea lui *c* nu va fi mai mică decât 3);

d este α – retezat, deoarece $0 < 3$;

e este α – retezat, deoarece $2 < 3$;

c este 3.

Implementare în Prolog

În cadrul unei implementări în Prolog a Algoritmului Alpha-Beta, relaţia principală este

```
alphabeta(Poz,Alpha,Beta,PozBuna,Val)
```

unde PozBuna reprezintă un succesor “suficient de bun” al lui Poz, astfel încât valoarea sa, Val, satisface cerinţele (1):

$$Val = V(Poz, Alpha, Beta)$$

Procedura

```
limitarebuna(ListaPoz,Alpha,Beta,PozBuna,
```

```
Val)
```

găseşte, în lista ListaPoz, o poziţie suficient de bună, PozBuna, astfel încât valoarea de tip minimax, Val, a lui PozBuna, reprezintă o aproximaţie suficient de bună relativ la Alpha şi Beta.

Intervalul alpha-beta se poate îngusta (dar niciodată lărgi) în timpul apelărilor recursive, de la o mai mare adâncime, ale procedurii.

Relaţia

```
limitenoi(Alpha,Beta,Poz,Val,AlphaNou,
BetaNou)
```

defineşte noul interval [AlphaNou, BetaNou]. Acesta este întotdeauna mai îngust sau cel mult egal cu vechiul interval [Alpha, Beta]. Îngustarea intervalului conduce la operaţii suplimentare de retezare a arborelui.

Algoritmul Alpha-Beta

```
alphabeta(Poz,Alpha,Beta,PozBuna,Val ):-
     mutari(Poz,ListaPoz),!,
     limitarebuna(ListaPoz,Alpha,Beta,PozBuna,Val);
     %valoare statică a lui Poz
     staticval(Poz,Val).

limitarebuna([Poz|ListaPoz],Alpha,Beta,PozBuna,
ValBuna):-
     alphabeta(Poz,Alpha,Beta,_,Val),
          destuldebun(ListaPoz,Alpha,Beta,Poz,Val,PozBuna,
     ValBuna).

%nu există alt candidat

destuldebun([ ],_,_,Poz,Val,Poz,Val):-!.
destuldebun(_,Alpha,Beta,Poz,Val,Poz,Val):-
     %atingere limită superioară
     mutare_min(Poz),Val > Beta,!;
     %atingere limită inferioară
     mutare_max(Poz),Val < Alpha,!.
destuldebun(ListaPoz,Alpha,Beta,Poz,Val,PozBuna,
```

```
ValBuna):-
     %rafinare limite
     limitenoi(Alpha,Beta,Poz,Val,AlphaNou,BetaNou),
     limitarebuna(ListaPoz,AlphaNou,BetaNou,Poz1,
                  Val1),
     maibine(Poz,Val,Poz1,Val1,PozBuna,ValBuna).

limitenoi(Alpha,Beta,Poz,Val,Val,Beta):-
     %creştere limită inferioară
     mutare_min(Poz),Val > Alpha,!.
limitenoi(Alpha,Beta,Poz,Val,Alpha,Val):-
     %descreştere limită superioară
     mutare_max(Poz),Val < Beta,!.
%altfel,limitele nu se schimbă
limitenoi(Alpha,Beta,_,_,Alpha,Beta).

%Poz mai bună ca Poz1
maibine(Poz,Val,Poz1,Val1,Poz,Val):-
     mutare_min(Poz),Val > Val1,!;
     mutare_max(Poz),Val < Val1,!.
%altfel, Poz1 mai bună
maibine(_,_,Poz1,Val1,Poz1,Val1).
```

Consideraţii privitoare la eficienţă

Eficienţa Algoritmului Alpha-Beta depinde de *ordinea în care sunt examinaţi succesorii.* Este preferabil să fie examinaţi mai întâi succesorii despre care se crede că ar putea fi cei mai buni. În mod evident, acest lucru nu poate fi realizat în întregime. Dacă el ar fi posibil, funcţia care ordonează succesorii ar putea fi utilizată pentru a se juca un joc perfect. În ipoteza în care această ordonare ar putea fi realizată, s-a arătat că Algoritmul Alpha-Beta nu trebuie să examineze, pentru a alege cea mai bună mutare, decât $O(b^{d/2})$ noduri, în loc de $O(b^d)$, ca în cazul Algoritmului Minimax. Aceasta arată că factorul de ramificare efectiv este $\sqrt{b}$ în loc de b – în cazul jocului de şah 6, în loc de 35. Cu alte cuvinte, Algoritmul Alpha-Beta poate “privi înainte” la o adâncime dublă faţă de Algoritmul Minimax,

pentru a găsi acelaşi cost.

În cazul unei ordonări neprevăzute a stărilor în spaţiul de căutare, Algoritmul Alpha-Beta poate dubla adâncimea spaţiului de căutare (Nilsson 1980). Dacă există o anumită ordonare nefavorabilă a nodurilor, acest algoritm nu va căuta mai mult decât Algoritmul Minimax. Prin urmare, în cazul cel mai nefavorabil, Algoritmul Alpha-Beta nu va oferi nici un avantaj în comparaţie cu căutarea exhaustivă de tip minimax. În cazul unei ordonări favorabile însă, dacă notăm prin N numărul poziţiilor de căutare terminale evaluate în mod static de către Algoritmul Minimax, s-a arătat că, în cazul cel mai bun, adică atunci când mutarea cea mai puternică este prima luată în consideraţie, Algoritmul Alpha-Beta nu va evalua în mod static decât $\sqrt{N}$ poziţii. În practică, o funcţie de ordonare relativ simplă (cum ar fi încercarea mai întâi a capturilor, apoi a ameninţărilor, apoi a mutărilor înainte, apoi a celor înapoi) ne poate apropia suficient de mult de rezultatul obţinut în cazul cel mai favorabil.