Jocuri

Un joc este o activitate ce presupune o succesiune de decizii luate de părți diferite ale căror scopuri și interese sunt opuse și poate fi definit formal ca o problemă de căutare cu următoarele componente:

a) Starea inițiala: se referă la pozițiile de pe tablă și la cine este cel care este la mutare
b) Mulțimea de acțiuni ce definesc mutările admise pe cre le poate face un jucător
c) Starea terminală ce determină când se sfârșește jocul
d) Funcția de utilitate care întoarce o valoare numerică pentru rezultaul jocului (posibilități: 1, 0, -1 reprezentând victorie, egal sau înfrângere)

Acestea pot fi clasificate după mai multe criterii:

1) în funcție de numărul de jucători: 1, 2, n (unde n nu reprezintă numărulde jucători, ci că reglile permit împărțirea jucătorului în n mulțimi disjuncte, unde jucatorii din fiecare mulțime au interese comune)
2) în funcție de natura mutărilor:
   - mutări libere (mutare aleasă în mod conștient, dintr-o mulțime de mutări posibile-șah)
   -mutări aleatorii (mutare aleasă de un factor aleatoriu -zaruri, monede, cărți de joc, etc)
   -ambele tipuri de mutări
3) în funcție de cantitatea informației de care dispune jucătorul
   -jucător ce vede configurația completă a pieselor adversarului (șah)
   -jucător ce nu vede configurația completă a pieselor adversarului (joc de cărți)

Algoritmul minimax

În algoritmul minimax, strategia este următoarea:

- jucătorul 1 (maxi) va încerca mereu să-și maximizeze propriul câștig prin mutarea pe care o are de făcut;
- Jucătorul 2 (mini) va încerca mereu să minimizeze câștigul jucătorului 1 la fiecare mutare.

Discuția se axează pe jocuri zero-sum, acest lucru garantează, printre altele, că orice câștig al jucătorului 1 este egal cu modulul sumei pierdute de jucătorul 2. Cu alte cuvinte, cu cât pierde jucătorul 2, atât câștigă jucătorul 1 și vice versa.

$$Win\_Player\_1 = |\ Loss\_Player\_2\ |$$
$$|\ Loss\_Player\_1\ | = Win\_Player\_2$$

Minimax este un algoritm de căutare limitată în adâncime (depth-first, depth-limited). Se pleacă de la poziția curentă și se generează o mulțime d epoziții următoare posibile. În acest scop, structura de date cel mai des folosită este arborele. Se aplică funcția de evaluare statică și se alege poziția cea mai bună.

Se presupune că jocul este de sumă nulă. Deci se paote folosi o singură funcție de evaluare pentru ambii jucători. Dacă f(n)>0, poziția n este bună pentru calculator și rea pentru om, iar dacă f(n)<0, poziția n este rea pentru calculator și bună pentru om. Funcția de evaluare este stabilită de cel ce proiectează aplicația.

Pentru aplicarea algoritmului, se selectează o limită de adâncime și o funcție de evaluare. Se construiește arborele până la limita de adâncime. Se calculează funcția de evaluare pentru frunze și se propagă evaluarea în sus, selectând minimele pe nivelul minimizant (decizia omului) și maximele pe nivelul maximizant (decizia calculatorului).

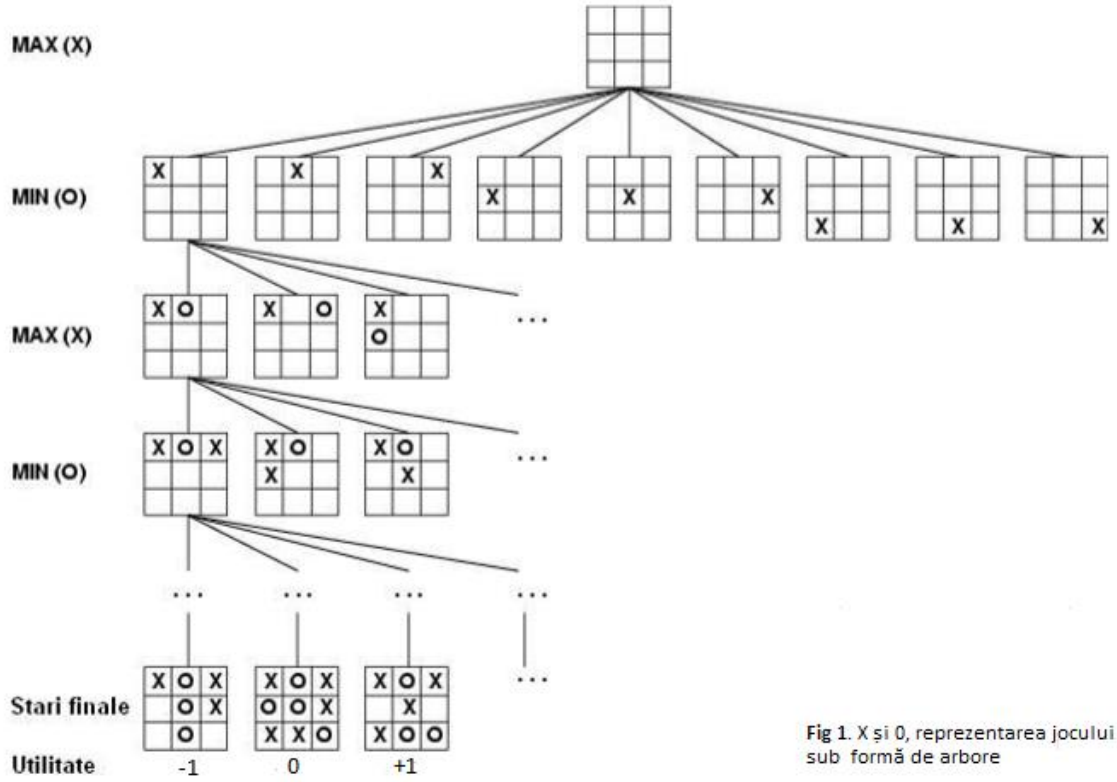

Fig 1. X și 0, reprezentarea jocului sub formă de arbore

De la starea inițială, MAX are posibilitatea de a alege din 9 stări posibile. Jucătorii alternează punând X si 0 până când se ajunge la o stare terminală – stare în care un jucător are trei elemente pe o linie, coloană sau diagonală, ori toate căsuțele sunt completate. Numărul atașat la fiecare nod frunză se referă la utilitatea stării terminale pentru jucătorul MAX. Valorile mari sunt considerate bune pentru MAX și proaste pentru MIN (și invers), de aici și numele celor doi jucători. Sarcina lui MAX este să folosească arborele de căutare pentru a determina cele mai bune mutări, ținând cont de utilitățile stărilor terminale.

Algoritmul minimax determină strategia optimă pentru MAX. Constă din 5 pași:

1. Generează tot arborele pentru joc până la stările terminale.
2. Aplică funcția de utilitate pentru fiecare stare terminală pentru a-i determina valoarea.
3. Folosește utilitatea stărilor terminale pentru a determina utilitatea stărilor de la un nivel superior din arborele de cautare.
4. Continuă evaluarea utilităților nodurilor pe niveluri mergând până la rădăcină.
5. Când se ajunge la rădăcină, MAX alege nodul de pe nivelul inferior cu valoarea cea mai mare.

Proprietățile algoritmului minimax:

-Completitudine? Da (dacă arborele este finit)

-Optimal? Da (împotriva unui adversar optimal)

-Complexitatea temporară? O(bm)

-Complexitatea spațială? O(bm) (explorare în adâncime)

(b este factorul de ramificare; m este adâncimea arborelui)

Alpha-beta pruning

O îmbunătățire substanțială a minimax este Alpha-beta pruning. Acest algoritm încearcă să optimizeze minimax profitând de o observație importantă: pe parcursul examinării arborelui de soluții se pot elimina întregi subarbori, corespunzători unei mișcări m, dacă pe parcursul analizei găsim că mișcarea m este mai slabă calitativ decât cea mai bună mișcare curentă.

Algoritmul alfa-beta face aceleași calcule ca si minimax, dar este mai eficient pentru că elimină ramurile arborelui de căutare care nu au relevanță pentru rezultatul final. De obicei nu este convenabil să se considere întregul arbore al jocului, chiar dacă se folosește și alfa-beta, motiv pentru care se oprește căutarea la un anumit punct și se aplică o funcție de performanță care estimează utilitatea unei stări.

Astfel, considerăm ca pornim cu o prima mișcare M1. După ce analizăm această mișcare în totalitate și îi atribuim un scor, continuăm să analizăm mișcarea M2. Dacă în analiza ulterioară găsim ca adversarul are cel puțin o mișcare care transformă M2 într-o mișcare mai slabă decât M1 atunci orice alte variante ce corespund mișcării M2 (subarbori) nu mai trebuie analizate.

Nu contează exact cât de slabă poate fi mișcarea M2 față de M1. O analiză amănunțită ar putea releva că poate fi și mai slabă decât am constatat inițial, însă acest lucru este irelevant. De ce însă ignorăm întregi subarbori și mișcări potențial bune numai pentru o mișcare slabă găsită? Pentru că, în conformitate cu principiul de maximizare al câștigului folosit de fiecare jucător, adeversarul va alege exact acea mișcare ce ii va da un câștig maximal. Dacă există o variantă si mai bună pentru el este irelevant, deoarece noi suntem interesați dacă cea mai slabă mișcare bună a lui este mai bună decât mișcarea noastră curent analizată.

Rolul mișcărilor analizate la început presupune stabilirea unor plafoane de minim și maxim legate de cât de bune/slabe pot fi mișcările. Astfel, plafonul de minim (Lower Bound), numit alpha stabilește că o mișcare nu poate fi mai slabă decât valoarea acestui plafon. Plafonul de maxim (Upper Bound), numit beta, este important doarece el folosește la a stabili dacă o mișcare este prea bună pentru a fi luată în considerare. Depășirea plafonului de maxim înseamnă că o mișcare este atât de bună încât adversarul nu ar fi permis-o, adică mai sus în arbore există o mișcare pe care ar fi putut s-o joace pentru a nu ajunge in situația curent analizată.

Astfel alpha si beta furnizează o fereastră folosită pentru a filtra mișcările posibile pentru cei doi jucatori. Evident, această fereastră se poate actualiza pe măsură ce se analizează mai multe mișcări.