Ce ne dă la examen

2 parti, teorie si probleme.
La teorie, o sa avem cateva intrebari cu raspuns scurt.
La probleme, o sa avem: calcule in retele Bayes, algoritmi in pseudocod, de demonstrat niste predicate, problema cu echilibrul nash

Ce nu ma duce capul sa scriu

Curs 2, Strategii de cautare: De la pagina 49, la 71
Curs 3.1, Cautari locale si online: De la pagina 5 la 34
Curs 3.2, CSP. De la pagina 19 la 21

Notare

Examen - 40%
Laborator - 25%
Teme - 25%
Teste de curs - 10%

Minim 7 laboratoare
Minim 50% din punctajul pe parcurs (laborator + teme)
Minim 50% din punctajul pe examen

Introducere

Alan Turing: O persoană nu poate face diferența între sistem și un om.

IA abordare simbolică - reprezentăm domeniul problemei și modul de rezolvare utilizând simboluri (modele bazate pe logica simbolică, ecuații, etc)
IA abordare non-simbolică - nu folosim abordare explicită. Folosim modele care majoritatea sunt inspirate din modelele biologice, din ființe vii (ex. Rețele neurale)

Caracteristicile problemelor de IA

Generale - trebuie să acopere mai multe instanțe de probleme
Dinamica modelului - modelul se poate schimba pe parcursul rezolvării problemei
Dificile - dpdv. al complexității calculului
Cunoștințe vs. date - Date (abordarea non-simbolică), cunoștințe (abordarea simbolică)
Utilizarea cunoștințelor euristice - capacitatea de a descoperi
Utilizarea cunoștințelor incerte
Necesită raționament, inferențe
Comportament autonom - fără intervenția directă a omului
Adaptare / învățare

Inferențe

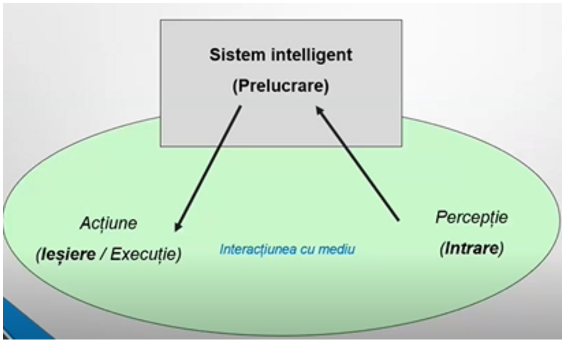
Regulă de inferență

- pornind de la anumite cunoștințe, obținem noi cunoștințe
- consistentă (sound) = garantează corectitudinea (adevărul) noilor cunoștințe obținute, dacă cunoștințele de la care s-a plecat sunt adevărate
- completă = capabilă prin utilizări repetate să obțină orice fapt rezultat care este o consecință a sistemului

Strategia de inferență

- aplicarea repetată a regulilor de inferență
- consistentă / inconsistentă, completă / incompletă

Structura unui sistem de IA



Capacitățile sistemelor de IA

- capacitatea de raționament: modele care includ reprezentarea cunoștințelor și utilizarea acestora în luarea deciziilor
- capacitatea de a învăța: modele cum ar fi rețele neurale, deep learning, sisteme de suport vectoriale, clustering, algoritmi genetici, etc. Aceste tehnici permit unui sistem să învețe cum să rezolve probleme ce nu pot fi specificate cu exactitate (non-simbolice)

Strategii de căutare

Reprezentarea soluției problemei

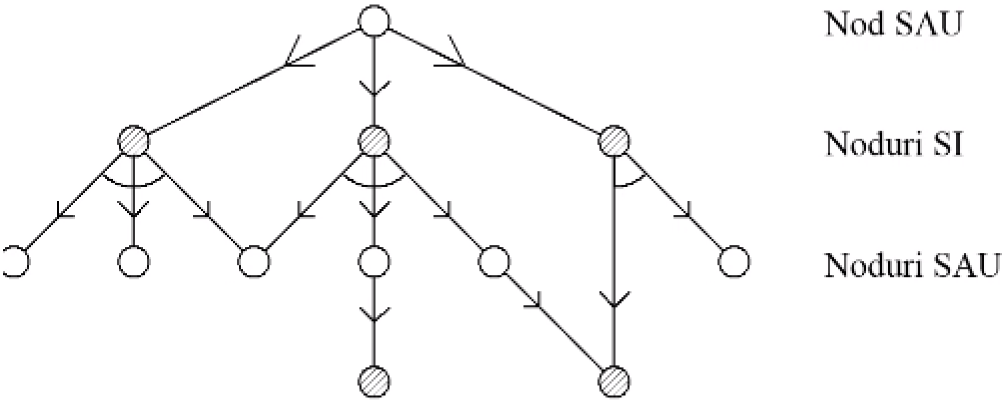
Grafuri reprezentate explicit - Știm dinainte cum arată tot graful.
Grafuri reprezentate implicit - Pe măsura căutării, anumite porțiuni devin explicite.

Reprezentarea prin spațiul stărilor

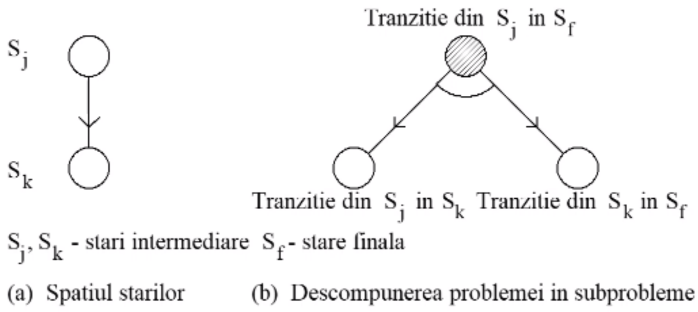
- (Si, O, Sf) - stare inițială, operatori pe stări, stări finale
- (S_Set) - spațiul stărilor
- Stările pot fi reprezentate explicit (știm exact starea finală), sau implicit (știm condițiile pe care starea finală trebuie să le îndeplinească)
- Soluția problemei: secvență de stări și operațiile dintre ele
- Exemplu de probleme: 8-puzzle, drum minim, problema comis-voiajorului

Reprezentarea prin grafuri SI-SAU

- (Pi, O, Pe) - problema inițială, operații pe probleme, problema finală
- (P_Set) - spațiul problemelor
- Nod = problema
- Problemă elementară = problemă care admite o soluție imediată
- Problemă neelementară = problemă care trebuie descompusă în subprobleme pentru a putea fi rezolvată
- Nodul poate fi rezolvat (nod care are toți copiii noduri probleme elementare sau rezolvate), nerezolvabil (are cel puțin un copil, o problemă neelementară și care nu mai poate fi descompusă)
- Nod SAU = descompuneri (modalități) alternative ale problemei în subprobleme, măcar una trebuie rezolvată
- Nod SI = descompunerea unei probleme în subprobleme, toate trebuie rezolvate
- Soluția problemei: acel sub-arbore care are ca rădăcină, nodul Pi, și care face ca Pi să devină nod rezolvat
- Exemplu de probleme: Turnurile din Hanoi, sistem Prolog



Reprezentările sunt echivalente



Caracteristicile mediului de rezolvare

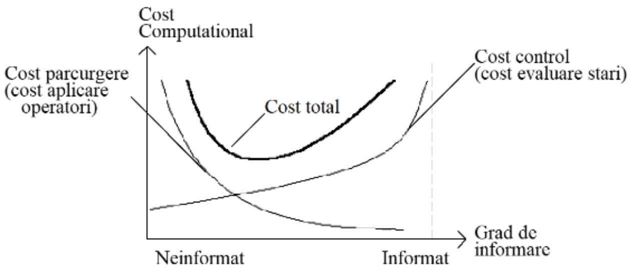
- Observabil / neobservabil
- Discret / continuu
- Finit / infinit
- Determinist (dintr-o stare cu o acțiune, pot trece într-o singură stare) / nedeterminist (dintr-o stare cu o acțiune, pot trece în diferite stări, cu o anumită probabilitate)

Strategii de căutare de bază

Criterii de caracterizare

- Completitudine
- Optimalitate (soluția de cost minim)

- Complexitate: timp, spațiu (în cazul grafurilor implicite, complexitatea se estimează în funcție de factorul de ramificare al stărilor)
- Capacitatea de revenire
- Informare



Strategii de căutare neinformate

- Grafuri specificate explicit
    - căutarea pe nivel și în adâncime
    - Algoritmi de căutare cu cost minim: Dijkstra (doar costuri pozitive), Bellman-Ford (și cu costuri negative), Floyd-Warshall (să nu aibă cicluri negative)
- Grafuri specificate implicit
    - căutarea pe nivel BFS și în adâncime DFS
    - căutare în adâncime cu nivel iterativ (iterative deepening)
    - Algoritmi de căutare: BFS, DFS, backtracking, căutare bidirectionala

- În spațiul stărilor
    - adancimea unui nod Ad(Si) = 0, starea inițială, Ad(S) = Ad(Sp) + cost, Sp = predecesorul lui S
- Grafuri SI-SAU
    - Adâncimea unui nod este diferită: Ad(Pi) = 0, Ad(P) = Ad(Pp) + cost, dacă Pp (predecesorul) este nod SAU, Ad(P) = Ad(Pp), dacă Pp (predecesorul) este nod SI

BFS / DFS

1. Initializeaza listele OPEN ← {Si }, CLOSED ← {}
    2. daca OPEN == {} atunci intoarce INSUCCES
    3. Elimina primul nod S din OPEN si insereaza-l in CLOSED
    4. daca S este in OPEN sau CLOSED (inainte de inserare S) atunci repeta de la 2
    4'. daca Ad(S) = AdMax atunci repeta de la 2 /* pt DFS */
    5. Expandeaza nodul S
    5.1.Genereaza toti succesorii directi Sj ai nodului S
        5.2.pentru fiecare succesor Sj al lui S executa
        5.2.1. Stabileste legatura Sj → S
        5.2.2. daca Sj este stare finala atunci
            i. Solutia este (Sj ,.., Si)
            ii. intoarce SUCCES
    5.2.3. Insereaza Sj in OPEN, la sfarsit / la inceput
    6. repeta de la 2 sfarsit.

Iterative Deepening (combina avantajele BFS si DFS)

pentru AdMax de la 0 la infinit
    rezultat = DFS(AdMax)
    daca am gasit solutia, intoarce rezultat

Complexitatea strategiilor de căutare neinformate

Complexitatea strategiilor de căutare

Criteriu	Nivel	Adancime	Adanc. limita	Nivel iterativ	Bidirectionala
Timp	B^d	B^d	B^m	B^d	$B^{d/2}$
Spatiu	B^d	B*d	B*m	B^d	$B^{d/2}$
Optimalitate?	Da	Nu	Nu	Da	Da
Completa?	Da	Nu	Da daca $m \geq d$	Da	Da

B – factor de ramificare, **d** – adancimea solutiei,
m – adancimea maxima de cautare (AdMax)

32

Strategii de căutare informate

Euristici folosite pentru:
- selectarea nodului următor de expandat
- în cursul expandării unui nod al spațiului de căutare, se poate decide, pe baza informațiilor euristice, care dintre succesorii săi vor fi generații care nu
- eliminarea din spațiul de căutare a anumitor noduri generate

Căutare de tip best-first
- evaluarea cantității de informație
- calitatea unui nod este estimată de funcția de evaluare euristică, notată w(n) pentru nodul n

Algoritmul Best-First

1. Initializeaza listele OPEN ← {Si }, CLOSED ← {}
2. Calculeaza w(Si) si asociaza aceasta valoare nodului Si
    3. daca OPEN == {} atunci intoarce INSUCCES
    4. Elimina nodul S cu w(S) minim din OPEN si insereaza-l in CLOSED
    5. daca S este stare finala atunci
        i. Solutia este (S,.., Si)
        ii. intoarce SUCCES
    6. Expandeaza nodul S
        6.1. Genereaza toti succesorii directi Sj ai nodului S
        6.2. pentru fiecare succesor Sj al lui S executa
            6.2.1 Calculeaza w(Sj) si asociaza-l lui Sj
            6.2.2. Stabileste legatura Sj → S
            6.2.3. daca Sj nu este in OPEN sau CLOSED atunci introduce Sj in OPEN cu w(Sj) asociat
            6.2.4. altfel
                i. Fie S'j copia lui Sj din OPEN sau CLOSED
                ii. daca w(Sj) < w(S'j)

atunci

- Elimina S'j din OPEN sau CLOSED
- Insereaza Sj cu w(Sj) asociat in OPEN

iii. altfel ignora nodul Sj

7. repeta de la 3 sfarsit.

Cazuri particulare

Strategia de cautare "best-first" este o generalizare a strategiilor de cautare neinformate

- strategia de cautare pe nivel w(S) = Ad(S)
- strategia de cautare in adincime w(S) = -Ad(S)
- Strategia de cautare de cost uniform / Dijkstra $w(S_j) = \sum_{k=i}^{j-1} cost\_arc(S_k, S_{k+1})$
- Minimizarea efortului de cautare – cautare euristica w(S) = functie euristica

Căutarea soluției optime în spațiul starilor. Algoritmul A*

w(S) = g(S) + h(S)

Strategia de cautare "best-first" se modifica:

2.Calculeaza **w(Si)=g(Si) + h(Si)** si asociaza aceasta valoare nodului Si

…

6.2.4. altfel

i. Fie S'j copia lui Sj din OPEN sau CLOSED

ii. daca **g(Sj) < g(S'j)** atunci …

w(S) devine f(S) cu 2 componente:

- g(S), o functie care estimeaza costul real g*(S) al caii de cautare intre starea initiala Si si starea S
- h(S), o functie care estimeaza costul real h*(S) al caii de cautare intre starea curenta S si starea finala Sf
- f(S) = g(S) + h(S)
- f*(S) = g*(S) + h*(S)

Calculul lui f(s)

$$g(S) = \sum_{k=i}^{n} cost\_arc(S_k, S_{k+1})$$

h(S) trebuie sa fie admisibila = pentru orice stare S, h(S) <= h*(S) si h(Sf)=0

A* - proprietatea de admisibilitate

= garantat sa gaseasca calea de cost minim spre solutie

- h(S) = admisibila
- cost_ arc(S,S') >= c, pentru orice doua stari S, S', unde c > 0 este o constanta finita

Completitudine – garantat sa gaseasca solutie daca solutia exista si costurile sunt pozitive

Daca h2(S) > h1(S), pentru orice S != Sf, => h2 domina h1 => A2* este **mai informat** decat A1*

Consistenta si monotonia functiei euristice

h(S) = consistenta, daca: h(S) <= h(S') + cost_arc(S,S') si h(Sf) = 0, pentru oricare stare S si S' = succesor al lui S

h(S) = monotona, daca f(Sy) >= f(Sx), pentru orice y > x. Estimarea costului total al caii de cautare este nedescrescatoare de la un nod la succesorii lui

O functie euristica h este **consistenta** daca si numai daca este **monotona**.
O functie euristica **consistenta** este **admisibila**.
Functiile euristice **admisibile nu sunt necesar si consistente**.
Maximul a 2 functii admisibile este admisibila. Maximum a 2 functii consistente este consistenta.
Daca h este monotona atunci avem garantia ca un nod introdus in CLOSED nu va mai fi niciodata eliminat de acolo si reintrodus in OPEN iar implementarea se poate simplifica corespunzator.

Relaxarea condiției de optimalitate a algoritmului A*

O functie euristica h se numeste epsilon-admisibila daca
h(S) <= h (S)* + epsilon, epsilon > 0
Algoritmul epsilon-admisibil cu h epsilon-admisibila gaseste intotdeauna o solutie (epsilon-optimala) al carei cost depaseste costul solutiei optime cu cel mult epsilon.

Determinarea funcției de evaluare h

Determinata manal sau generata automat.
Transformare abstracta prin relaxarea unor restrictii ale problemei.
Pattern databases – precalculeaza si memoreaza distanta pana la solutie intr-un spatiu abstract generat de relaxarea restrictiilor impuse miscarilor/actiunilor.

Strategii de cautare locala

Cautari locale – opereaza asupra starii curente generand succesorii.
Calea catre solutie nu are importanta.
Gasire solutie – CSP - nu conteaza calitatea solutiei.
Gasire solutie optima – functie de evaluare sau de cost.
Folosesc putina memorie. Gasesc solutii destul de bune in spatii finite f mari si chiar in spatii infinite.

Problema satisfacerii restrictiilor (CSP)

{X1 … XN} = o serie de variabile
D = {D1 … DN} = o serie de domenii de valori pentru fiecare variabila
R = {R1 … Rk} = o serie de restrictii (toate trebuie satisfacute)
Restrictii unare, binare, globale
Daca avem domenii discrete si finite de valori, toate restrictiile pot fi transformate in restrictii unare sau binare.
CSP partiala = ne dorim satisfacerea cat mai multor restrictii, daca nu reusim pe toate
CSP binara – graf de restrictii

Imbunatatirea performantelor BKT

- Algoritmi care modifica spatiul de cautare prin eliminarea unor portiuni care nu contin solutii
    - Algoritmi de imbunatatire a consistentei reprezentarii (utilizati inainte de inceperea cautarii): Consistenta locala a arcelor sau a cailor in graful de restrictii
    - Algoritmi de cautare (cauta solutia si elimina portiuni din spatiul de cautare): Imbunatatesc performantele rezolvarii prin reducerea numarului de teste.

- Utilizarea euristicilor in cautare

Propagarea locala a restrictiilor

Combinatia de valori x si y pentru variabilele Xi si Xj este permisa de restrictia explicita Rij(x,y).
Un arc (Xi , Xj) intr-un graf de restrictii orientat se numeste **arc-consistent** daca si numai daca pentru **orice valoare x din Di,** domeniul variabilei Xi, **exista o valoare y din Dj,** domeniul variabilei Xj, **astfel incat Rij(x,y)**.
Graf de restrictii orientat arc-consistent – orice arc din graf este arc-consistent.

AC-3: Realizarea arc-consistentei pentru un graf de restrictii

```
/* Intoarce fals daca inconsistenta, adevarat in caz contrar */
Creeaza o coada Q ← { (Xi , Xj ) | (Xi , Xj ) apartine Multime arce, i != j}
cat timp Q nu este vida executa
        Elimina primul arc (Xk , Xm) din Q
        daca Verifica(Xk , Xm) atunci
                daca |DXk| = 0 atunci intoarce fals
                Q ← Q reunit cu { (Xi , Xk ) | (Xi , Xk ) apartine Multime arce, i != k,m}
intoarce adevarat

Verifica (Xk , Xm) /* intoarce adevarat daca se modifica DXk */
modif ← fals
pentru fiecare x din DXk executa
        daca nu exista nici o valoare y in DXm astfel incat Rkm(x,y) atunci
                elimina x din DXk  /* Modifica domeniul DXk prin efect lateral */
                modif ← adevarat
intoarce modif
```

Complexitate
N - numarul de variabile
a - cardinalitatea maxima a domeniilor de valori ale variabilelor
e - numarul de restrictii.
Algoritmului de realizare a arc-consistentei - AC-3: complexitate timp este O(e*a^3)
S-a gasit si un algoritm O(e*a^2)

m-Cale-consistență = O cale de lungime m prin nodurile i0 ,…,im ale unui graf de restrictii orientat se numeste m-cale-consistenta daca si numai daca pentru orice valoare x din Di0 , domeniul variabilei i0 si o valoare y din Djm, domeniul variabilei im, pentru care Ri0im(x,y), exista o secventa de valori z1din Di1 … zm-1 din Dim-1 astfel incat Ri0i1 (x,z1), …, Rim-1im(zm-1 ,y)

Cautare cu Backtracking

```
/* Intrari: variabile V, restrictii R
Iesiri: Atribuiri pt X si adev daca R satisfacute, fals in caz contrar
L- variabile instantiate, U – variabile neinstantiate */
(b,L) ← BKT_MAC(V, {}, R,D)
daca b atunci intoarce L
intoarce fals
sfarsit
```

```
BKT_MAC(U,L,R,D)
        daca U={} atunci
                intoarce (adevarat, L)
        X ← Selectie(U)
        pentru fiecare x din DX repeta
                (b,D’) ← AC-3’ (L {(X,x),R,D)
                daca b atunci
                        (b,L) ← BKT_MAC(U\{X}, L {(X,x)},R,D’)
                        daca b atunci
                                intoarce (b,L)
                        intoarce fals sfarsit
```

Euristici generale

Ordonarea variabilelor – vezi functie Selectie(U)
- aleator
- Minimum remaining value (MRV) – se incepe cu variabila cea mai restrictionata intai (fail-first) – variabila cu cele mai putine valori legale
- Degree heuristic – selectie variabila care este implicata in cel mai mare numar de restrictii cu variabile neinstantiate

Ordonarea valorilor
- Least-constrained value – selectie valoarea care elimina cele mai putine valori din domeniul variabilelor neinstantiate cu care este legata prin restrictii (fail-last)

O solutie: variabila fail-first, valoare fail last. Toate solutiile: variabila fail-last, valoare fail first

CSP partiala

Memoreaza cea mai buna solutie gasita pana la un anumit moment (gen IDA*) – distanta d fata de solutia perfecta. Abandoneaza calea de cautare curenta in momentul in care se constata ca acea cale de cautare nu poate duce la o solutie mai buna.
NI - numarul de inconsistente gasite in "cea mai buna solutie" depistata pana la un moment dat – limita necesara
S - limita suficienta - specifica faptul ca o solutie care violeaza un numar de S restrictii (sau mai putine), este acceptabila.

```
/* intoarce GATA sau CONTINUA */
1. daca Variabile = {} atunci
        1.1 CeaMaiBuna ← Cale
        1.2 NI ← Distanta
        1.3 daca NI <= S atunci
                intoarce GATA
        altfel intoarce CONTINUA
2. altfel
        2.1 daca Valori == {} atunci CONTINUA /* s-au incercat toate valorile si se revine la var
        ant. */
        2.2 altfel
                2.2.1 daca Distanta >= NI atunci
                        intoarce CONTINUA /* revine la var ant pentru gasirea unei solutii mai
                        bune*/
                2.2.2 altfel
```

i. Var ← first(Variabile)
ii. Val ← first(Valori)
iii. DistNoua ← Distanta
iv. Cale1 ← Cale
v. cat timp Cale1 != {} si DistNoua < NI executa
    (VarC, ValC) ← first(Cale1)
    daca Rel(Var,Val,VarC,ValC) == fals
        atunci DistNoua ← DistNoua + 1
    Cale1 ← Rest(Cale1)
vi. daca DistNoua < NI si
PBKT(Cale+(Val,Var),DistNoua,Rest(Variabile),ValoriNoi) = GATA /*
ValoriNoi - domeniul de valori asociat primei variabile din Rest(Variabile)
*/
    atunci intoarce GATA
altfel intoarce PBKT(Cale,Distanta,Variabile,Rest(Valori)
sfarsit

Căutare adversarială în jocuri

S: set de stări cu S0
N – număr de jucători
A – mulțime de acțiuni
f: S x A → S (f sau next)
Q: S → RN – funcția de utilitate / recompensa; Q(S) = (recompensa1, recompensa2, ….,
recompensa pentru jucatorul N)
J: S → (1, 2…., N) – jucătorul care joaca

Jocuri cu 2 jucatori

Minimax

Etichetez fiecare nivel din arborele jocului cu MAX (jucator) și MIN (adversar)
Etichetez frunzele cu scorul jucatorului
Parcurg arborele jocului

- dacă nodul parinte este MAX atunci i se atribuie valoarea maxima a succesorilor sai;
- dacă nodul parinte este MIN atunci i se atribuie valoarea minima a succesorilor sai.

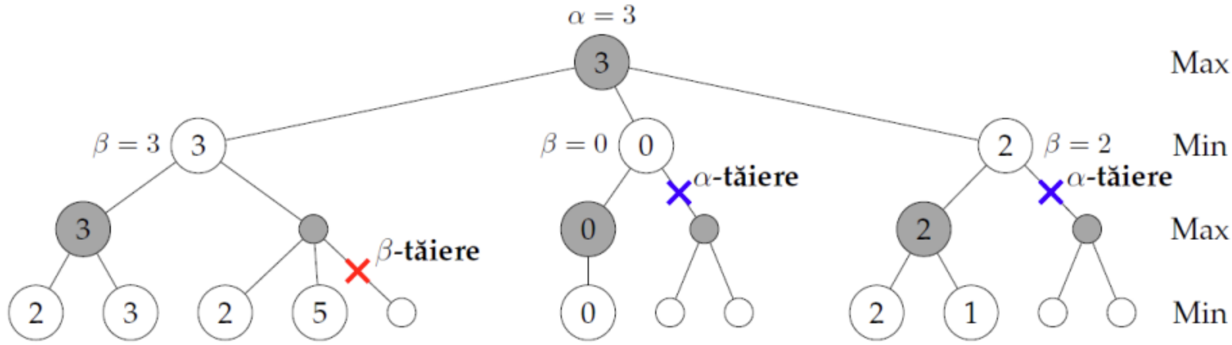
Spatiul de cautare este foarte mare => Algoritmul Minimax se face pana la o adancime n
O functie euristica de evaluare a unui nod eval(S).

Alpha-Beta pruning

Este posibil sa se obtină decizia corecta a algoritmului Minimax fara a mai inspecta toate nodurile din spatiului de cautare pana la un anumit nivel.
Fie **α** cea mai buna valoare (cea mai mare) gasita pentru MAX si **β** cea mai buna valoare (cea mai mica) gasita pentru MIN.
Algoritmul alfa-beta actualizeaza **α** si pe parcursul parcurgerii arborelui si elimina investigarile subarborilor pentru care **α** sau **β** sunt mai proaste.
Terminarea cautarii (taierea unei ramuri) se face dupa doua reguli:

- **α**-taieri - În cazul în care exista, pentru un nod MIN, o actiune ce are asociata o valoare v <= **α**, atunci putem renunta la expandarea subarborelui sau, deoarece MAX poate atinge deja un câstig mai mare, dintr-un subarbore precedent.

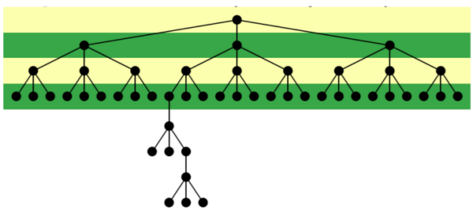
- **β**-taieri - În cazul în care exista, pentru un nod de tip MAX, o actiune ce are asociata o valoare v >= **β** , atunci putem renunta la expandarea subarborelui sau, deoarece MINa limitat deja castigul lui MAX la **β**



ALGORITMUL ALPHA BETA
Eficienta algoritmului depinde semnificativ de ordinea de examinare a starilor. Se recomanda o ordonare euristica a succesorilor, eventual de generat numai primii cei mai buni succesori.

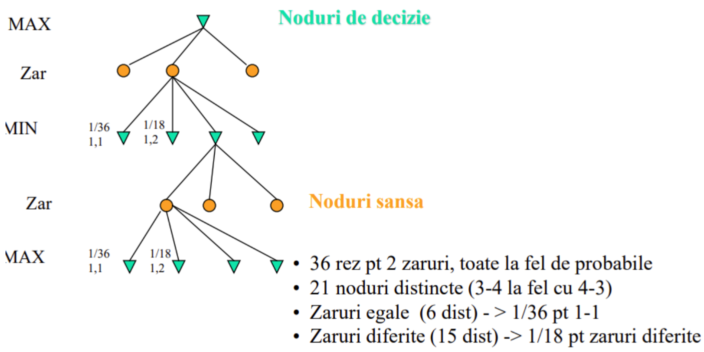
Imbunatatiri

Timp limitat pentru executarea unei miscari – anytime algorithm. Foloseste Iterative Deepening. Cobor in arbore pana expira timpul. Dar fac cautare pe nivel, pentru a ma asigura ca parcurg toate posibilitatile, si nu doar ma adancesc in arbore pe o singura varianta.
Memoize – tabela hash cu pozitiile de joc pentru a obtine: Estimari ale nodurilor si Cea mai buna miscare dintr-un nod.
Efectul de orizont – cauta mai mult decat limita de cautare pentru anumite pozitii, pentru ca e posibil ca starile foarte promitatoare, sa fie fake.



Jocuri cu elemente de șansă

Jucatorul nu cunoaste mișcările legale ale oponentului
3 tipuri de noduri: MAX, MIN, Șansă (chance nodes)
Noduri șansă – ramurile care pleaca dintr-un nod șansa indica posibile rezultate ale sansei (de exemplu zar)



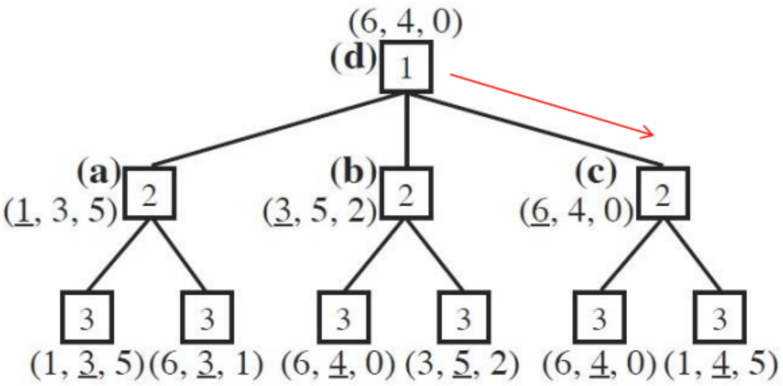
Functia de evaluare
- scor – pentru nodul terminal
- max din Minimax succesori - pentru nodul MAX
- min din Minimax succesori - pentru nodul MIN
- [P(Sj)*Minimax(Sj)] succesori - pentru nodul SANSA = suma, pentru toti succesori, din probabilitatea succesorului * valoarea minmax a succesorului

Jocuri cu mai mulți jucători

In general, 2 strategii

- Maxn - generalizare a Minimax pt n jucatori
- Paranoic – reduce la joc cu 2 jucatori in care se presupune ca toti ceilalti colaboreaza impotriva jucatorului simulat

Maxn



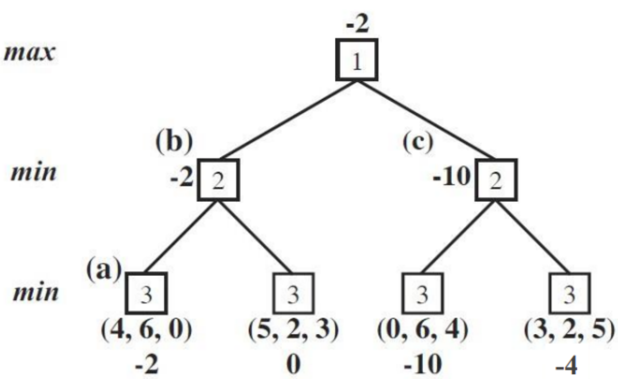
Nivelele nu se mai numesc MIN-MAX, se numesc 1-2-3….cati jucatori avem. Informatia din nod nu va mai fi un scor, ci scorul pentru fiecare jucator in parte aflat in starea respectiva.
Pentru nodurile din interior, valoarea Maxn a unui nod in care jucatorul i muta este valoarea Maxn a succesorului pentru care a i-a componenta din vector este maxima.

Maxn(Nod, Juc)

1. daca Nod este nod final atunci intoarce scor(Nod[Juc])
2. daca nivel(Nod) = n atunci intoarce eval(Nod[Juc])
3. altfel
    - 3.1 P ← Prim_succesor(Nod)
    - 3.2 Best [Juc] ← Maxn (P, Juc_Urm)
    - 3.3 pentru fiecare succesor Sj ≠P al lui Nod executa
        - Curent ← Maxn(Sj , Juc_Urm))
        - if Curent[Juc] > Best [Juc] atunci Best ← Curent
4. Intoarce Best [Juc]

Paranoic

Reduce jocul la 2 jucatori si se poate aplica Minimax. Exista cazuri in care greseste; in aceasta situatie, cu cat se cauta mai adanc cu atat este mai proasta estimarea.

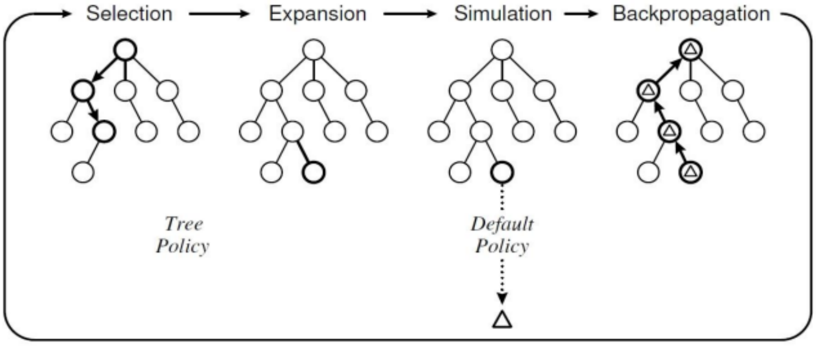


Monte Carlo Tree Search

Baza metodei este o unda de joc ("playout")

Playout = un joc rapid jucat cu mutari dintr-o anumita stare pana la sfarsitul jocului, obtinandu-se castig/pierdere sau un scor.
Fiecarui nod parcurs i se asociaza un merit.
In varainta cea mai simpla acest merit este un procent de castig = de cate ori s-a castigat daca s-a pornit unda din acel nod.
Algoritmul construieste progresiv un arbore de joc partial, ghidat de rezultatele explorarilor anterioare ale acestui arbore. Arborele este utilizat pentru a estima valoarea miscarilor, estimarile devenind din ce in ce mai bune pe masura ce arborele este construit. Algoritmul implica construirea iterativa a arborelui de cautare pana cand o anumita cantitate de efort sa atins, si intoarce cea mai buna actiune gasita.

Pentru fiecare iteratie se aplica 4 pasi:

- Selectie – Pornind de la radacina o politica de selectie a copiilor este aplicata recursiv pana cand se gaseste cel mai interesant nod neexpandat (un nod E care are un copil ce nu este inca parte a arborelui)
- Expandare – Un nod copil (sau mai multe noduri copii) a lui E este adaugate in arbore cf. actiunilor disponibile
- Simulare – Se executa o simulare de la nodul/nodurile noi cf. politicii implicite pentru a obtine un rezultat R
- Backpropagation – rezultatul simulatii este propagat inapoi catre nodurile care au fost parcurse si se actualizeaza valorile acestora
- 

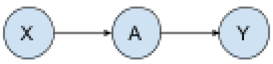
Converge spre valorile Minimax, dar lent. Cu toate acestea mai eficient decat AlfaBeta. Este un algoritm de tip anytime. Algoritmul nu gaseste întotdeauna cea mai buna mutare, dar are, în general, un succes rezonabil în cazul alegerii mutarilor care duc la sanse mari de câstig.

Invatarea functiei de evaluare pentru selectie

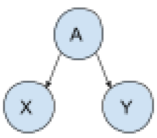
= machine learning
Gradient-descend pentru ponderile functiei de evaluare
Reinforcement Learning - Q Learning
Deep Q Learning

Teoria probabilitatilor

Probabilitate neconditionata (apriori) - probabilitatea unui eveniment inaintea obtinerii de probe.
Probabilitate conditionata (aposteriori) - probabilitatea unui eveniment dupa obtinerea de probe.
X, Y = independente => P(X, Y) = P(X) * P(Y)



=> P(X, Y | A) = P(X | A) * P(Y | A)



=>P(X, Y | A) = P(X | A) * P(Y | A)

P(X | Y) = $\frac{P(X,Y)}{P(Y)}$;

P(X, Y) = P(X | Y) * P(Y)

Daca Y poate fi descompus intr-o serie de evenimente incompatibile intre ele, {Y1, Y2, Y3, …}, atunci:

P(X) = P(X | Y1) * P(Y1) + P(X | Y2) * P(Y2) + … P(X | Yn) * P(Yn)

Particularizare: P(X) = P(X | Y) * P(Y) + P(X | !Y) * P(!Y)

Bayes => P(X | Y) = $\frac{P(Y \mid X) * P(X)}{P(Y)}$

Generalizarea la mai multe ipoteze si probe

h_i – evenimente / ipoteze probabile (i=1,k);

$e_1,\dots,e_n$ – probe (evenimente)

$P(h_i)$

$P(h_i \mid e_1,\dots,e_n)$

$P(e_1,\dots,e_n \mid h_i)$

$$P(h_i|e_1,e_2,\dots,e_n) = \frac{P(e_1,e_2,\dots,e_n|h_i)\cdot P(h_i)}{\sum_{j=1}^{k} P(e_1,e_2,\dots,e_n|h_j)\cdot P(h_j)},\ i=1,k$$

Daca $e_1,\dots,e_n$ sunt ipoteze independente atunci

$$P(e|h_j) = P(e_1,e_2,\dots,e_n|h_j) = P(e_1|h_j)\cdot P(e_2|h_j)\cdot\ldots\cdot P(e_n|h_j),\ j=1,k$$