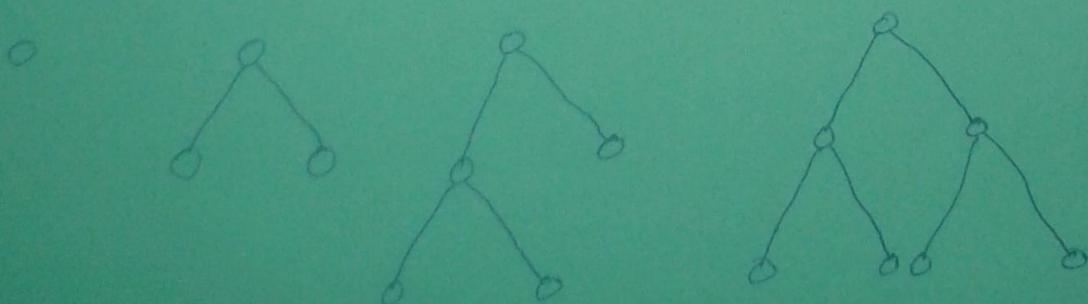


SUBIECTE IA

① Căutarea de tip breadth-first. Prezentare generală și implementare

a) Prezentare generală

- Strategia de căutare de tip breadth-first mai întâi modul rădăcină. Apoi se extind toate nodurile generate de nodul rădăcină, apoi succesorii lor și asa mai departe. În general, toate nodurile aflate la adâncimea d în arborele de căutare sunt extinse înaintea nodurilor aflate la adâncimea $d+1$. Spunem că acesta este o căutare în latime ^{method}.
- Căutarea de tip breadth-first poate fi implementată clândind algoritmul general, CAUTARE-GENERALĂ, cu o funcție COTRĂ-FN care plasează stările nou generate la sfârșitul casii, după stările generate anterior.
- Strategia breadth-first este foarte susținătoare deoarece ia în considerație toate drumurile de lungime 1, apoi pe cele de lungime 2 etc., ~~asa~~. Dacă există o soluție, este sigur că aceasta metodă o va găsi, dar dacă există multe soluții va găsi întotdeauna mai întâi soluția cea mai puțin adâncă.



• Pseudo code

- 1) Creează o variabilă numită LISTA-NODURI și setează la starea initială.
- 2) Până când este găsită o stare-scop sau până când LISTA-NODURI devine vidă, execută:
 - 2.1) Înălțăruți primul element din LISTA-NODURI și numește-l E.
Dacă LISTA-NODURI a fost vidă, STOP.
 - 2.2) Pentru fiecare nod în care fiecare regulă se potrivește cu starea descisă în E, execută:
 - 2.2.1) Aplicați regula pentru a genera o nouă stare.
 - 2.2.2) Dacă noua stare este o stare-scop, înțorce această stare și STOP.
 - 2.2.3) Atât timp ce noua stare nu este stare-scop, adăugați noua stare la sfârșitul lui LISTA-NODURI.

b) Implementarea în Prolog

rezolvă_b([Start, Sol]) :- breadthfirst([[], Start], Sol).
breadthfirst([[], Nod | Dnum] | _), [Nod | Dnum] :- scop(Nod).
breadthfirst([[], Dnum | Dnumuri], Sol) :-
 extinde(Dnum, DnumuriNoi),
 concat(Dnumuri), DnumuriNoi, Dnumuri1),
 breadthfirst(Dnumuri1, Sol).
extinde([[], Nod | Dnum] | DnumuriNoi) :-
 bagof([NodNou, Nod | Dnum],
 (SC(Nod, NodNou), \+ (membru(NodNou, [Nod | Dnum])),
 DnumuriNoi)), !.
extinde(-, []).

- Predicatul rezolvă_b(Start, Sol) este adevărat dacă Sol este un drum (în ordine inversă) de la nodul initial Start la o stăru-scop, drum obținut folosind cantică b-f.
- Predicatul breadthfirst(Dnumuri, Sol) este adevărat dacă un drum din multimea de drumuri candidat nu este drumuri care să fie extinse la o stare-scop, un astfel de drum este Sol.
- Predicatul extinde(Dnum, DnumuriNoi) este adevărat dacă prin extinderea multimii de noduri Dnum obținem multimea multă DnumuriNoi, el generând multimea tuturor extensiilor acestui drum.
- Predicatul concat(Dnumuri, DnumuriNoi, Dnumuri1) este adevărat dacă, atunci când concatenăm lista de noduri Dnumuri cu lista de noduri DnumuriNoi, obținem lista de noduri Dnumuri1.
- Predicatul membru(NodNou, [Nod | Dnum]) este adevărat dacă nodul NodNou aparține listei de noduri [Nod | Dnum].
- Predicatul scop(Nod) arată că Nod este un nod scop.

- Predecatul sc(Nod, NodNou) reprezintă funcția de succesiune și desemnează faptul că NodNou este nodul succesor al nodului Nod.

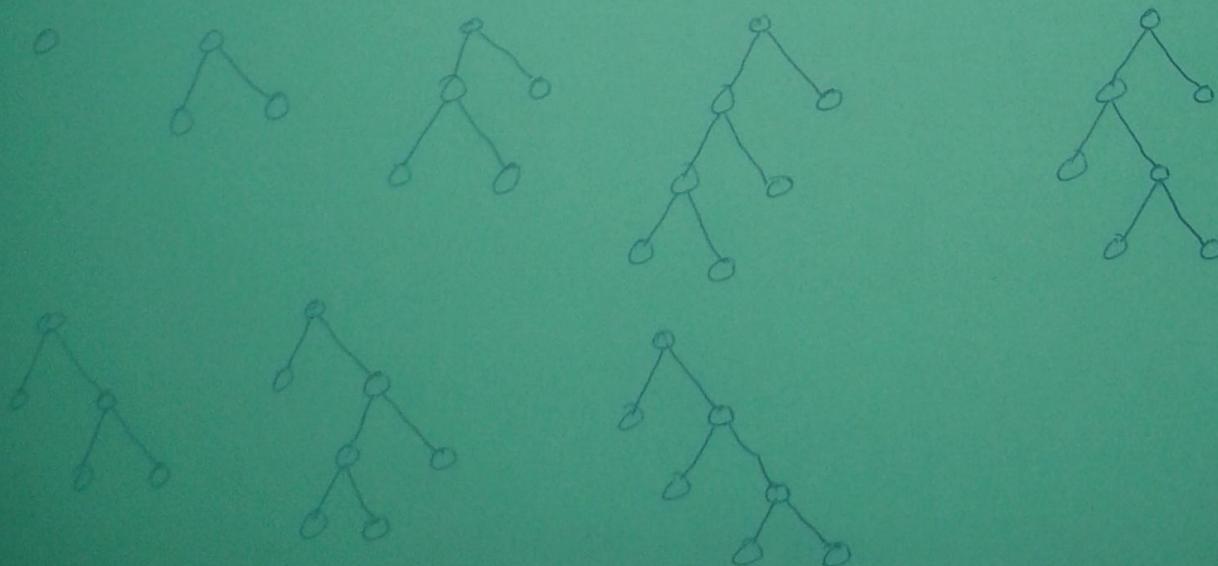
Complexitate

- b = factorul de ramificare
- d = lungimea drumului ce reprezintă soluția
 $\Rightarrow \Theta(b^d)$

② Căutarea de tip depth-first. Prezentau general și implementație

a) Prezentare generală

- Strategia de căutare de tip depth-first extinde înălțimea unui altă nodurile aflate la nivelul cel mai adânc din arbore. Căutarea se întoarce înapoi și sărăcă nodurile aflate la adâncină mai mică numai atunci când a fost extins un nod care nu reprezintă un nod scop și care nu mai poate fi extins. Să spun că aceasta este o căutare în adâncină. Modul
- Implementarea se poate face cu o structură de date de tip coadă, cum înălțimea va plasa stările noi generate în faza costului.
- Necesităriile de memorie sunt foarte mici. Este necesar să se mențină un singur drum de la radacina la un nod frumos, împreună cu podurile fructelor sănătoase corespunzător fiecărui nod de pe drum.
- Principialul dezavantaj al acestui stratégiei este acela că o căutare va continua în jos, chiar dacă o soluție nu există.



b) Implementarea în Prolog

• Pseudocode

- 1) Dacă N este un nod-scop, atunci $Sol = EN \cup \{N\}$ sau
- 2) Dacă există un succesor, N_1 , al lui N , astfel încât să existe un drum, Sol_1 , de la N_1 la un nod-scop, atunci $Sol = EN \cup Sol_1 \cup \{N\}$.

b) Implementarea în Prolog

`resolvă-d(CN, EN) :- scop(CN).`

`resolvă-d(CN, EN \cup Sol1) :- s(CN, N1), resolvă-d(CN1, Sol1).`

• Varianta cu detectare a ciclurilor

`resolvă1-d(CN, Sol) :- depthfirst(EN, CN, Sol).`

`depthfirst(Drum, Nod, ENod \ Drum) :- scop(Nod).`

`depthfirst(Drum, Nod, Sol) :-`

`s(Nod, Nod1),`

`\+ membru(Nod1, Drum)),`

`depthfirst(ENod \ Drum, Nod1, Sol).`

• Predicatul `resolvă1-d(CN, Sol)` este aderent dacă Sol este un drum (în ordine inversă) de la nodul initial N la o stare scop.

• Predicatul `depthfirst(Drum, Nod, Sol)` este aderent dacă urmărește drumul $MaNod$ și ajunge la un nod scop.

• Predicatul `membru(Nod1, Nod2, Drum)` este aderent dacă nodul $membru(Nod1, Nod2, Drum)$ este succesorul nodului $Nod1$ pe drumul $Drum$.

• Predicatul `s(Nod, Nod1)` reprezintă funcția de succesiune și desemnează faptul că $Nod1$ este nodul succesor al Nod .

③ Căutare în adâncime iterativă - Prezentare generală și implementare

a) Prezentare generală.

- Căutarea în adâncime iterativă este o strategie care evită cheerea unei stabilități unei adâncimi optime la ceea ce trebuie căutată soluția, prin testarea tuturor limitelor de adâncime 0, 1, 2 ...
- Acest tip de căutare combină beneficiile căutării breadth-first și depth-first, după cum urmăză:
 - este optima și completă ca și căutarea breadth-first;
 - conservă numai cantitatea mică de memorie necesară căutării depth-first
- Ordinea extinderei nodurilor este similară cu cea de la căutarea de tip breadth-first, murad că amanuntele sunt reținute de mai multe ori. Această strategie de căutare garantează găirea nodului-scop de la adâncimea minima, dacă un scop poate fi găsit.

Complexitate

- b = factorul de ramificare
- d = lungimea drumului ce reprezintă soluția
- $\Rightarrow \mathcal{O}(b^d)$

5) Implementarea în Prolog

cale (Nod1, Nod2, L Nod3).

cale (PrimNod, UltimNod, E UltimNod | Druim) :-

cale (PrimNod, PenultimNod, Druim),

S (PenultimNod, UltimNod),

\+ (membru (UltimNod, Druim)).

depth-first-iterative-deepening (Nod, Sol) :-

cale (Nod, NodScop, Sol),

scop (NodScop), !.

• Predicatul cale (Nod1, Nod2, Druim) este aderent dacă Druim reprezintă o cale aciclică între nodurile Nod1 și Nod2 în spațiu standarde. Această cale va fi reprezentată ca o listă de noduri date în ordine inversă. Se generează toate dinurile aciclice posibile de lungime care crește cu cătă o unitate.

• Predicatul SC PenultimNod, UltimNod reprezintă faptul de succesiune și dezcompunerea faptului că UltimNod este valoarea lui el modulul PenultimNod.

• Predicatul membru (UltimNod, Druim) este aderent dacă nodul UltimNod aparține listei de noduri Druim.

• Fapta \exists Scop (Nod) arată că Nod este un nod-scop.

• Predicatul depth-first-iterative-deepening (Nod, Sol) este aderent dacă Sol este un drum (în ordine inversă) de la nodul initial Nod la o stare - scop.

④ Algoritmul A*. Textul algoritmului și admissibilitatea acestuia

a) Textul Algoritmului

- 1) Creașă un graf de căutare G , constând numai din nodul initial n_0 . Platează n_0 într-o listă numită $OPEN$.
- 2) Creașă o listă numită $CLOSED$ care initial este vidă.
- 3) Dacă lista $OPEN$ este vidă, $EXIT$ cu eroare.
- 4) Selectează primul nod din lista $OPEN$, înălță-l din $OPEN$ și platează-l în lista $CLOSED$. Numește-l n .
- 5) Dacă n este un nod-scop, oprește execuția cu succes.
Reținem noastră soluția obținută urmând un drum de-a lungul parțialor de la n la n_0 în G .
- 6) Extinde nodul n , generând o mulțime, M , de succesiuni ale lui n care nu sunt deja stocate adiacenți n în G . Instalează aceste membri ai lui M ca succesiuni adiacenți n în G .
- 7) Stabileste un pointer către n de la fiecare dintre membrii lui M care nu se găseau deja în G (adică nu se aflau deja nici în $OPEN$ nici în $CLOSED$). Adaugă acesta membru al lui M întriu $OPEN$. Pe trai fiecare membru m al lui M care se află deja în $OPEN$ sau în $CLOSED$ redirecționează pointerul său către n , dacă cel mai bun drum la n găsit până în acel moment trece prin m . Pe trai fiecare membru al lui M care se află deja în lista $CLOSED$, redirecționează pointerul fără ca să fie direct către descendenții săi din G astfel încât această să treacă în apoi de-a lungul căruia să fie drumuri până la acești descendenți, găzduindu-și acel moment.

- 8) Recordarea lista CPBN în ordinea valorilor curențe ale funcțiilor f . (Evenimentele legături între valorile mănuștele ale lui f sunt rezolvate în formăa redusă din arborele de căutare aflat la ea nu vor adăuga).
- 9) Mergi la pasul 3.

5) Admisibilitatea algoritmului A*

Există anumite condiții asupra profunzorii și a lui h care garantează că algoritmul A*, aplicat acestor profundi, găsește întotdeauna drumuri de cost minim.

1) Orice nod al profunzorii, dacă admite succesor, are un număr finit de succesoare.

2) Toate nodele din profunzor au costuri mai mici decât o căditate pozitivă E .

• Condiția asupra lui h este:

1) pentru toate nodurile n din profunzor de căutare, $h(n) \leq h^*(n)$. Cu alte cuvinte, h nu supraestimează niciodată valoarea efectivă h^* . O astfel de funcție h este numită un estimator optimist.

Observații:

- Este relativ ușor să se găsească în probleme, o funcție h care să satisfacă această condiție a limitelor de jos.
- În cele 3 condiții formulate anterior, algoritmul A* garantează găirea unui drum optim la un scop, în cazul în care f .

Teorema

Stim că sunt îndeplinite condițiiile asupra profunzorii și asupra lui h și că condiția că f este un drum de cost finit de la noeudul un nod-scop. Alg. A* garantează găirea drumurilor de cost minim.

⑤ Admisibilitatea și optimitatea alg. A*

a) Admisibilitatea

demonstrată în 4.6)

b) Optimalitatea

Fie G_0 stoc-scop optimă cu un cost al drumului rotat f^* . Fie G_2 o a doua stoc-scop, suboptimă, care este o stoc-scop cu un cost al drumului

$$f(G_2) > f^*$$

Presupunem că A^* selectează din coadă, pentru extindere, pe G_2 . Întrucât G_2 este o stoc-scop, aceasta alegeră să închidă cantică cu o soluție suboptimă. Dar săn că acest lucru nu este posibil.

Fie un nod n , care este la pasul curent, un nod finisă pe un drum optim al G . Când oricare nod trebuie să existe, în afara canticii de cost drumul a fost complet extins, cost în care algoritmul îl va returna G .

Pt. acest nod n , întrucât n este admisibilă, trebuie să avem:

$$f^* \geq f(n) \quad (1)$$

Mai mult, dacă n nu este ales pentru extindere în favoarea lui G_2 , trebuie să avem:

$$f(n) \geq f(G_2) \quad (2)$$

$$(1), (2) \Rightarrow f^* \geq f(n) \geq f(G_2) \quad (3)$$

În schimb, dacă G_2 este o stoc-scop, suntem $f(G_2) = 0$. În schimb,

$$f(G_2) > g(G_2) \quad (4)$$

$$(3), (4) \Rightarrow f^* \geq f(G_2)$$

Aceasta concluzie contrazice faptul că G_2 este suboptimal.
Ea arată că A^* nu selectează nicio soluție pentru extragere un
scop suboptimal. A^* întoarce o soluție numără după ce a
selectat-o pentru extragere.

$\Rightarrow A^*$ este optim.

② Implementarea în Prolog a căutării best-first

% Predicatul bestfirst(Nod-initial, Soluție) este aderat
dacsă soluție este un drum (obținut folosind strategia
best-first) de la nodul Nod-initial la o stare-scop.%
bestfirst(Nod-initial, Soluție):-

expansie([], l(Nod-initial, []), [], da, Soluție).

expansie([Drum, l(N, _)], - , da, [N|Drum]):- scop(N).

% Această fază 1. dacă N este nod-scop, atunci construim o
cale soluție %

expansie([Drum, l(N, F/G)], Limita, Arb1, Res, Sol):-

F = L(Limita,

(bagaj(M/C, e3(N,M,C)), t(marbur(M,Drum))), Succ), !,

Limita succ(G, Succ, As),

că-mai-bună-f(As, F1),

expansie([Drum, t(N,F1/G,As)], Limita, Arb1, Res, Sol);

Res = disponibil).

% fază 2. dacă N este nod frunză a cărui f-valoare este
mai mică decât Limita, atunci îl generez succesorul și il
metin în harta Limita. %

expansie([Drum, t(N, F/G, L1/A1)], Limita, Arb1, Res, Sol):-

F = L(Limita,

că-mai-bună-f(As, BF),

arb1(Limita, BP, Arb1A1),

expansie([CN1/Drum, t(N, F/G, L1/A1/A2)], Limita, Arb1, Res, Sol),

expansie([Drum, t(N, F/G, L1/A1/A2)], Limita, Arb1, Res, Sol)

% Caz 3: dacă arborele de radacina N nu subarbore reprezintă
P și valoarea este mai mare decât Limită, atunci extindem
ul radă prioritată subarborele său; în funcție de rezultatul
astăzi, respectiv, vom decide cum amine să constituim
continuă sau întregul predicetului continuă %
Ex pandează (-, +(-,-, E3), -,-, ImparSel, -) :- !.

% Caz 4: pe acestă rădăcina vorbăță nu vom obține nicio soluție
a soluției %.

Ex pandează (-, Arb, Limită, Arb, nu, -) :-

f(Arb, P), P > Limită.

% Caz 5: în cazul unor f-valori mai mici decât Bound, arborele
nu mai poate fi extins. %

continuă (-, -, -, -, da, da, Sel).

continuă (P, +(N, P/G, EA1/1AS3), Limită, Arb1, nu, Res, Sel) :-

rezolvă (A1, AS, NAS),

cea - mai - bună - f(CNAS, PA1),

Ex pandează (P, +(N, P1/G, NAS), Limită, Arb1, Res, Sel).

continuă (P1, +(N, P/G, E-1AS3), Limită, Arb1, ImparSel, Res, Sel) :-

cea - mai - bună - f(AS, P1),

Ex pandează (P1, +(N, P1/G, AS), Limită, Arb1, Res, Sel).

Ultimucc (-, E3, ES).

Ultimucc (G0, EN/1NCS3, TS) :-

G = G0+C,

H = (N, H1),

F = G + eH,

Ultimucc (G0, NCS, TS1),

Ultimucc (E(N, P/G, TS1, TS)).

% Predicatul inseră (A, As, As1) este utilizat pentru
adăugarea unui arbore A într-o listă de arbori As,
menținând ordinea impusă de \hat{f} -valoarele lor. %

Inseră (A, As, EA/As1):-

$f(A, P)$,

ceas-nod-bună- $f(As, P_1)$,

$P_2 \leftarrow P_1$, !.

Inseră (A, EA1/As1, EA1/As13):- inseră (A, As, As1).

subr (x, Y, x) :- x = Y, !.

subr (-, Y, Y).

$f(l(-, P/-), P)$. % f -val unei frunze

$f(+(-, P/-, -), P)$. % f -val unui arbore

% Predicatul cea-nod-bună- $f(As, P)$ este utilizat pentru
a determina cea mai bună \hat{f} -valoare a unui arbore din
listă de arbori As, dacă această listă este nevodată;
lățea As este ordonată după \hat{f} -valoare subarborilor
constituenți %

cea-nod-bună- $f(EA1-3, P) :- f(A, P)$.

Cea-nod-bună- $f(E3, 999\ 999)$.

% În cazul unei liste de arbori nicide, \hat{f} -valoarea
determinată este foarte mare. %

④ Algoritmul minimax. Textul algoritmului și implementarea

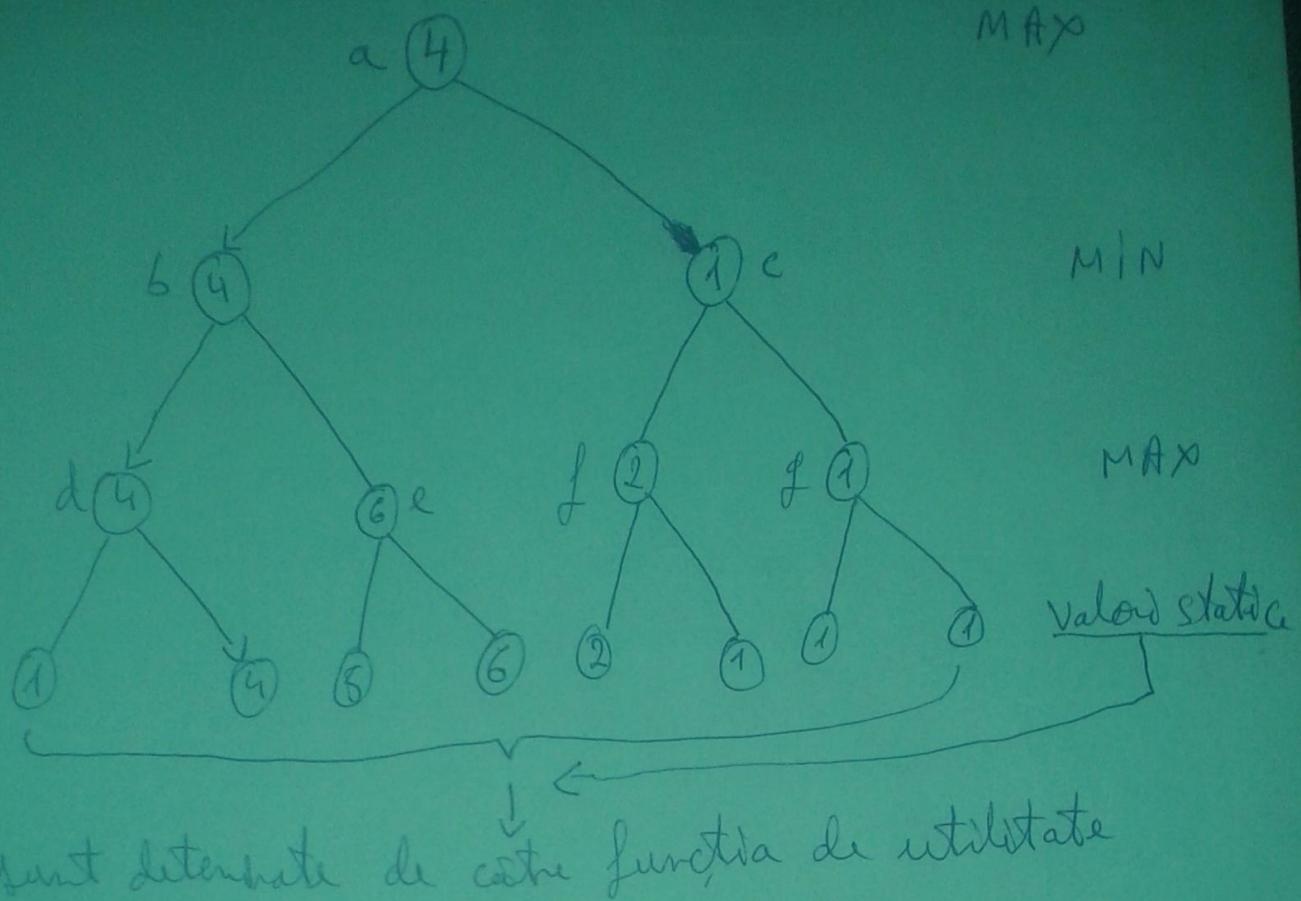
a) Textul algoritmului

- MAX = reprezentă jucătorul care încercă să câștige sau să nu maximizeze urmărușul opus.
- MIN = este opozitul care încercă să minimeze scorul lui MAX

- 1) Generează întreg arborele de joc, până la stările finale (terminal).
- 2) Aplică funcția de utilitate fiecărui stări terminală pentru a obține valoarea corespunzătoare stării.
- 3) Deplasează-te înapoi în arbore, de la nodurile-frunze spre nodul rădăcină, determinând, corespunzător fiecărui nivel al arborelui, valoare care reprezintă utilitatea nodurilor afilate la acel nivel. Propagarea acestor valori la niveluri anterioare se face prin intermediul nodurilor părinte succinți, conform următoarei reguli:
 - dacă starea-părinte este un nod de tip MAX, atribuie-i maximul dintre valoările obținute de fișii săi.
 - dacă starea-părinte este un nod de tip MIN, atribuie-i minimul dintre valoările obținute de fișii săi.
- 4) Ajuns în nodul - rădăcină, alege pentru MAX acea mutare care conduce la valoarea maximă.

Complexitate

- m = adâncimea maximă a arborelui
 - b = mutari legale la fiecare punct
- $\Rightarrow \Theta(b^m)$



- Predicatul numărător (Pos, succBun, Val) Pos este o posibilă, Val este valoarea ei de tip numărător, dar ea nu sunt mutare de la Pos conduce la pozitiva succBun.
- Predicatul mutare (Pos, listaPos) corespunde regulilor procedurii care dă loc mutările legale: listaPos este lista posibilităților succesor legale ale lui Pos. Predicatul este ocazional dacă Pos este o posibilă de către terminală.
- Predicatul cel mai bun (listaPos, PosBun, ValBun) selectează cea mai bună posibilă PosBun dintre-o lista de posibile candidate. ListaPos, ValBun este valoarea lui PosBun și prima urmărlă a lui Pos. „Cel mai bun” înseamnă că acest obiectul de maton său de mândru în funcție de poziția care are cea mai mare.

2) Prezentare generală (cu un exemplu)

- Atunci când este aplicată unui arbore de tip minmax Stanford, ea va întoarce acesta mutare pe care o furniză și o fuziune. Întrucât acesta este un mod foarte scurt, întrucât rezultă o reținere a unei ramuri ale arborelui care nu pot influenta decizia finală.
- Principul general al acestui tehnic constă în a considera un nod concret n al arborelui, astfel încât jucatorul poate alege să facă o mutare la acel nod. Dacă acesta jucător dispune de o alegere nod orientațională, m, și la urmările modului posibile al lui m, și în orice punct de decizie aflat mai sus în arbore, atunci nu va fi să fie menționat în timpul jocului. Prin urmare, de înțeles că în una exponențială mare doar descendenții modului m, pot să obțină suficiente informații relative la acesta, și putem elibera.
- Ideea tehnică de alpha-beta reținere este aceea de a face o mutare « suficient de bună », nu neapărat cea mai bună, dar suficientă pentru a se lăsa decizia corectă.
- Această idee poate fi formalizată prin introducerea a două limite, alpha și beta, reprezentând limitări ale valorii de tip minimax compunându-un nod intern. Semnificația acestor limite este următoarea: alpha este valoarea minimă pe care este deja garantat că o va obține MAX, iar beta este valoarea maximă pe care MAX poate spera să o obțină. În punctul de vedere al jucătorului MIN, beta este valoarea cea mai reațabilă pe care MIN pe care acesta o va obține.
- Prin urmare, valoarea efectivă conținută va fi să situa reația între α și β.
- Valoarea alpha, o valoare redusă de tip MAX, nu poate mențină să crească, iar valoarea beta, o valoare redusă de tip MIN nu poate mențină să crească.

Cele 2 reguli pt. buclarea cantică:

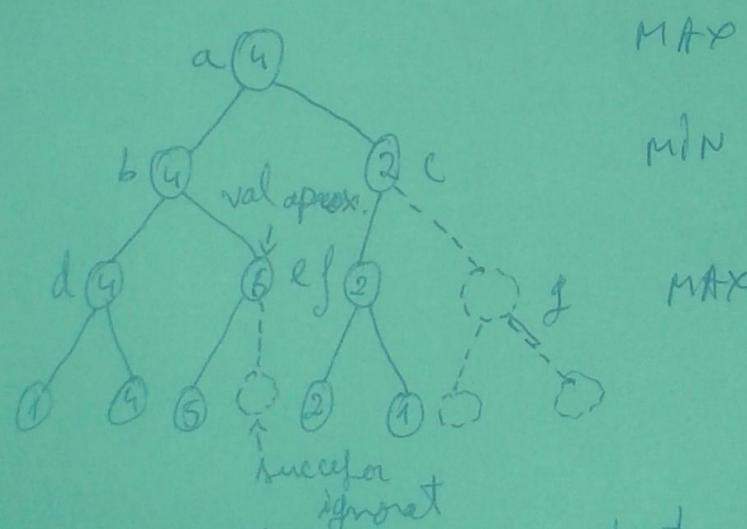
- 1) Cantică poate fi apărtă de deschisul ordinii mod de tip MIN. care are o valoare baza mai mare sau egală cu valoarea alpha a ordinii dintr-o cantică sănătă de tip MAX.
- 2) Cantică poate fi apărtă de deschisul ordinii mod de tip MAX care are o valoare alpha mai mare sau egală cu valoarea baza a ordinii dintr-o cantică sănătă de tip MIN.

Complexitate

- d = adâncimea maximă a arborelui
- b = mutari legale la fiecare punct

$$\Rightarrow O(b^d)$$

Exemplu



- 1) Scoape din poz a; 2) mutare la b; 3) mutare la d;
- 4) Aceea valoarea maximă a succesorului lui d care conduce la $V(d) = 4$.
- 5) Introduc în nodul s-a executat o mutare de la e la f.
- 6) Iată că considerăm primul succesor al lui e a căruia valoare este 5. În acest moment, MAX, a cărui mutare următoare este să devină în poziția e, va păstra valoarea 5, indiferent care din cele trei alternative pleacă din poz e. Această informație este suficientă pentru ca MIN să realizeze că la nodul b

alternativa e este inferioră alternatiei d.

Aceasta concluzie poate fi făcută fără a cunoaște valoarea exactă a lui ϵ . Pe același bază, cel de-al doilea succesor al lui ϵ poate fi neglijat, dar rezultatul e în același valoare aproximativă 5.

5) Implementarea în Prolog

alphaseta (Pos, Alpha, Beta, PosBună, Val) :-

mutare (Pos, ListaPos), !,

lrbună (ListaPos, Alpha, Beta, PosBună, Val);
static_val (Pos, Val).

lrbună ([Pos | ListaPos], Alpha, Beta, PosBună, ValBună) :-

destul_de_sus (ListaPos, Alpha, Beta, -, Val),

% nu există alt candidat

destul_de_sus ([], -, Pos, Val, Pos, Val) :- !.

destul_de_sus (-, Alpha, Beta, Pos, Val, Pos, Val) :-

% atingere lărgită superioară

mutare_min (Pos), Val > Beta, !;

% atingere lărgită inferioră

mutare_max (Pos), Val < Alpha, !.

destul_de_sus (ListaPos, Alpha, Beta, Pos, Val, PosBună, ValBună) :-

lrbună (Alpha, Beta, Pos, Val, AlphaNou, BetaNou),

lrbună_bună (ListaPos, AlphaNou, BetaNou, Pos1, Val1),

val_bine (Pos, Val, Pos1, Val1, PosBună, ValBună).

lrbună (Alpha, Beta, Pos, Val, Val, Beta) :-

peste_lărgită_inferioră

mutare_min (Pos), Val > Alpha, !.

% descrie latura superioara % Imitare - mă (Alpha, Beta, Poz, Val, Alpha, Val) :-
mutare - mă (Poz), Val < Beta, !.

% altfel latura nu se schimbă
latura (Alpha, Beta, --, Alpha, Beta).

% Poz nu sănătă ca Poz1

nu sănătă (Poz, Val, Poz1, Val1, Poz, Val) :-

mutare - mă (Poz), Val > Val1, !;

mutare - mă (Poz), Val < Val1, !.

% altfel, Poz1 nu sănătă

nu sănătă (--, Poz1, Val1, Poz1, Val1).

- În relația alphabeta (Poz, Alpha, Beta, PozBună, Val) :
PozBună reprezintă un succesor "sufulent de bună" al lui Poz,
astfel încât valoarea sa, Val, satisfacă condiție: Val = V(Poz, Alpha, Beta)

- Procedura lărgirea bună (Lătă Poz, Alpha, Beta, PozBună, Val)

Găsește în lătă lătă Poz, o poză suffudent de bună, PozBună și
valoarea de tip numărăt, Val, a lui PozBună, reprezentă o
aproximație suffudent de bună relativ la Alpha și Beta.

Întreabă alpha, beta se poate doar înțelege.

- Relația lărgirea (Alpha, Beta, Poz, Val, AlphaNou, BetaNou)
defineste nouă interval $\llbracket \text{AlphaNou}, \text{BetaNou} \rrbracket$. Aceasta este atâta
nouă infuză sau că al mult zgădui cu vecindul interval $\llbracket \text{Alpha}, \text{Beta} \rrbracket$.
Infuzarea întreabăului conduce la operări suplimentare de
intersecție a intervalelor.

③ Reprezentarea cunoștințelor cu reguli if-then. Interpretarea acestor reguli în cazul întâlnirii nedorită

a) Reprezentarea cunoștințelor cu reguli if-then

- Reguli de tip if-then, numite și reguli de producție, constituie o formă naturală de exprimare a cunoștințelor și au un rol important în cunoștințele suplimentare:

- 1) Modularitatea: fiecare regula definește o cunoștință relativă care nu este independentă de celelalte.
- 2) Invariabilitatea: regulile pot fi adăugate sau scăzute cunoștințe și nu relație independentă de celelalte reguli.
- 3) Modificabilitatea: regulile vechi pot fi modificate relativ independent de celelalte reguli.
- 4) Sunt doar transportabile între domenii.

- Reguli de tip if-then adesea definesc relații logice între concepții apartinând domeniului problemei. Relațiile logice pot fi considerate ca apartinând domeniului problemei și a multilor cunoștințe categorice, adică cunoștințe care vor fi întărite sau adăvădate.
- Regulile de producție pot fi modificate prin adăugarea la interpretarea lor logica a unei clase calificări de vioritate, obținându-se reguli de formă următoare:
$$\text{if } \text{condiție A} \text{ then } \text{concluzie B} \text{ cu certitudine F}$$

⑩ Reprezentarea anotăntilor cu regulă if-then. Interpretator pentru regulă în cazul înțelâptului inadăpt.

a) Reprezentarea anotăntelor cu regulă if-then.

Definit în §.a)

b) Interpretator pentru regulă în cazul înțelâptului inadăpt

Ințelâptura înțelege nu începe cu o spovedă, ci face un răzbunare în direcția opusă, de la poarta cu sf la poarta cu then.

inadăpt :-

faptă - nouă - dedusă (P), !, % o nouă faptă.

write ('Sedus:'), write (P), nl,

assert (faptă (P)), inadăpt; % continuă

write ('Nu vă există faptă'). % toate faptele au fost deduse

faptă - nouă - dedusă (cond) :-

% o regulă

Codul său regulă nu este
încă o faptă %

if cond then cond,

not faptă (cond),

faptă - corporată (cond).

% Condiția este atenuată?

faptă - corporată (cond) :-

% faptă nupla

faptă (cond).

faptă - corporată (cond1 and cond2) :-

faptă - corporată (cond1),

% ambele corpură sunt aderante

faptă - corporată (cond2).

faptă - corporată (cond1 or cond2) :-

faptă - corporată (cond1),

faptă - corporată (cond2).

⑩ Generarea explicativă și introducerea incertitudinii în răspunsuri reportajelor

a) Generarea explicativelor

- Regulile de producție facilitează generarea răspunsului pt urmat 2 tipuri de înțelesuri ale interlocutorului:
 - înțeles de tipul „cum”: Cum ad ajuns la această concluzie?
 - înțeles de tipul „de ce”: De ce te interesează această informație?
- În cazul înțeleselor de tipul „cum”, explicația pe care sistemul o furnizează cu privire la modul în care a fost dedus răspunsul său constituie un arbore de demonstrație a modulului în care concluzia finală decurge din regulile și faptele oferite în baza de cunoștințe.
- Fie $\vdash \vdash$ un operator diferențial. Atunci arborele de demonstrație al unei proprietăți poate fi reprezentat în următoarele forme, în funcție de varianta:

 - 1) Dacă P este o faptă, atunci arborele de demonstrație este P .
 - 2) Dacă P a fost dedus folosind o regulă f și C , atunci arborele de demonstrație este $P \vdash \vdash \text{Der}f(C)$, unde $\text{Der}f(C)$ este un arbore de demonstrație a lui C .
 - 3) Fie $P_1 \wedge P_2$ propoziții ale căror arbore de demonstrație sunt Der_1 și Der_2 . Dacă P este de forma $P_1 \wedge P_2$, atunci arborele de demonstrație corespunzător este $\text{B}(\text{Der}_1 \wedge \text{Der}_2)$.
Dacă P este de forma $P_1 \vee P_2$, atunci arborele de demonstrație este $\text{fr}(\text{Der}_1, \text{Der}_2)$.

Construcția arborelor de demonstrație în Prolog este devenită
% este aderent(P, Dem) dacă Dem constituie o
demonstrație a faptului că P este aderent.

i- $\exists P (\exists \theta_0, \forall f(x), \angle)$

este-aderent(P, P) :- lampa(P).

este-aderent(P, P \angle Dem(Cord)) :- if Cord then P,
este-aderent(Cord, Dem(Cord)).

este-aderent(P₁ and P₂, Dem₁ and Dem₂) :-

este-aderent(P₁, Dem₁),

este-aderent(P₂, Dem₂).

este-aderent(P₁ or P₂, Dem) :- este-aderent(P₁, Dem) ;

este-aderent(P₂, Dem).

5) Introducerea incertitudinii

- Incertitudinea poate fi modelată prin distribuția unei calificări altă decât aderent sau fals, respectivă alegibile.
- Gradul de aderență poate fi exprimat prin intervalele unei măsură reală aflat într-un anumit interval - spre exemplu, un număr între 0 și 1 sau între -5 și 5. Astfel de măsuri cauzează literatura de specialitate, o întreagă varietate de denumiri care ar fi făcut de certitudine, măsură a incertitudinii.
- Fiecare propoziție își va adăuga un număr între 0 și 1 ca factor de certitudine. Reprezentarea folosită va consta dintr-o percheie de formă: Propoziție: Factor certitudine.
- Această notăție va fi aplicată și regulilor. Astfel, măsuraa forței va fi definită regulă și gradul de certitudine până la care acea regulă este validă: If Condition then Conclusion: Certitudine.
- În cazul oricărui reprezentare cu incertitudine este nevoie specificarea modului în care se combină certitudinile propozitiilor și a regulilor.
- Spre exemplu să presupunem că sunt 2 propoziții $P_1 \wedge P_2$ având certitudinile $c(P_1)$ și respective $c(P_2)$. Atunci putem defini $c(P_1 \wedge P_2) = \min(c(P_1), c(P_2))$, $c(P_1 \vee P_2) = \max(c(P_1), c(P_2))$.
- Dacă există regula $f: P_1 \rightarrow P_2: C$ cu C reprezentând factorul de certitudine, atunci $c(P_2) = f c(P_1) + C$.
- Implementarea în Prolog a unui interpretor de reguli corespunzător schemei de incertitudine descrie presupune specificarea de către utilizator a estimării de certitudine corespunzătoare datelor obținute (modurile căreia din stocă ale relațiilor) și apoi să fie dat (Propoziție, Certitudine).

% contitudine (Proposición, contitudine
contitudine (P, Cst) :- dat (P, Cst).

contitudine (Cord 1 and Cord 2, Cst) :-

contitudine (Cord 1, Cst 1),

contitudine (Cord 2, Cst 2),

minimun (Cst 1, Cst 2, Cst).

contitudine (Cord 1 or Cord 2, Cst) :-

contitudine (Cord 1, Cst 1),

contitudine (Cord 2, Cst 2),

maximun (Cst 1, Cst 2, Cst).

contitudine (P, Cst) :-

if Cord Then P : C 1,

contitudine (Cord 1, Cst 1),

Cst 1 : C 1, C 2).

12) Înferențe în Rețele Bayesiene. Algoritmul pentru rezolvare
interrogându probabilisticele creșpuna unui poligonar

a) Înferențe în Rețele Bayesiene

- Rețelele Bayesiene sunt suficient de flexibile pentru ca orice mod să poată servi fie ca o vorabilitate de întreagere, fie ca o vorabilitate dovedă. În general, un agent primește valori ale vorabilitelor dovedă de la rețeaua saia (sau în una altor răssoarăți) și întrebă despre posibilele valori ale altor vorabilități astfel încât să poată decide ce să facă trăsătura întreprinsă.
- Literatura de specialitate discută 4 tipuri de metode de inferențe care pot fi realizate de rețelele Bayesiene:
 - inferență de tip diagnostic (de la efect la cauză);
 - inferență causală (de la cauze la efect);
 - inferență intercausală (trei cauze ale unui efect comun);
 - inferență mixtă (reprezentând combinația a două sau mai multe dintr-oferitele anterioare).

Functii evitnice probleme laborator

1) Problema combinatorilor de multimi

$\text{scstone}(B, NMB, NCB, NMIS, NCant, M),$

$\text{stone}(B_1, NMB_1, NCB_1, NMIS_1, NCant_1, M)) :-$

$M1 \leq m_{mb}(M, NMB), M2 \leq m_{mc}(M, NCB),$

$\text{for}(0, M_1, k_1), \text{for}(0, M_2, k_2), (k_1 \geq k_2, k_1 \geq 0),$

$k_1 + k_2 \leq M, k_1 + k_2 \geq 0, NMIS_1 \leq NMB - k_1, NCant_1 \leq$

$NCB - k_2, (NMIS_1 \geq NCant_1; NMIS_1 = 0),$

$NMB_1 \geq NMIS_1 + k_1, NCB_1 \geq NCant_1 + k_2, (NMB_1 \geq NCB_1;$

$NMB_1 = 0),$

$\text{opus}(B, B_1).$

$\text{opus}(\text{est}, \text{west}).$

$\text{opus}(\text{west}, \text{est}).$

2) Problema mutarii blocurilor

$\text{M}(\text{Lista} - \text{stoc}, \text{List} \ddot{\text{a}} - \text{stoc} - \text{nr}) :- \text{membar}(\text{x}, \text{Lista} - \text{stoc}),$

$\text{x} \in \text{var} \{ 1 - 7, \text{det} - \text{pos} - \text{el} (\text{Lista} - \text{stoc}, \text{N}, \text{x}),$

$\text{adug} - \text{la} - \text{m} (\text{Lista} - \text{stoc}, \text{List} \ddot{\text{a}} - \text{stoc} - \text{iter}, \text{N}),$

$\text{membar}(\text{Y}, \text{Lista} - \text{stoc}),$

$\text{det} - \text{pos} - \text{el} (\text{Lista} - \text{stoc}, \text{N}1, \text{Y}), \text{N}1 \geq \text{N},$

$\text{adug} - \text{la} - \text{m} (\text{Varf}, \text{Lista} - \text{stoc} - \text{iter}, \text{Lista} - \text{stoc} - \text{nr}, \text{N}1),$

$\text{el} - \text{permute} (\text{Lista} - \text{stoc}, \text{List} \ddot{\text{a}} - \text{stoc} - \text{nr}).$

3) Problema Varselen reprezante (3, 5 → 4)

Se $st(A, B), st(A_1, B_1)$:-

$A > 0, A_1 \leq 0, B_1 = B;$

$B > 0, B_1 \leq 0, A_1 = A;$

$A \geq 5, A_1 \leq 5, B_1 = B;$

$B \leq 3, B_1 \leq 3, A_1 = A;$

$B_1 \leq \min(A+B, 3), A_1 \leq A+B-B_1;$

$A_1 \leq \min(A+B, 5), B_1 \leq A+B-A_1;$

4) X și O

$s(st(jucator, T, N, M), st(jucator_opus, Tablou1, N1, M1))$:-

$\neg tablou_fbal(T, -), N > 0,$

$juc_opus(jucator, jucator_opus),$

$N_1 \leq N-1, s(s(jucator, T, Tablou1)).$

5) Laborint

Se $pct(L, C), pct(L_1, C_1), l1$:-

$\neg od_zid(pct(L, C), pct(L_1, C_1)),$

$\neg od_zid(pct(L_1, C_1), pct(L, C)).$