Inteligenţa artificială **poate fi privită ca fiind o ştiinţă sui-generis al cărei obiect de studiu îl constituie modul de programare a calculatoarelor în vederea săvârşirii unor operaţiuni pe care, deocamdată, oamenii le efectuează mai bine. Conform acestei definiţii primare, putem gândi inteligenţa artificială ca fiind acel domeniu al informaticii care se ocupă de *automatizarea comportamentului inteligent*.**

Domeniu al informaticii => opereaza cu:

- ***structurile de date*** **folosite în reprezentarea cunoştinţelor;**
- ***algoritmii*** **necesari pentru aplicarea cunoştinţelor;**
- ***limbajele*** **şi *tehnicile de programare*** **folosite la implementarea acestor algoritmi.**

- **problema definirii domeniului inteligenţei artificiale devine, până la urmă, una a definirii conceptului însuşi de *inteligenţă* => legătura inteligenţei artificiale cu alte domenii, cum ar fi, în primul rând, filozofia.**

- **Domeniul inteligenţei artificiale îşi propune:**

- ***înţelegerea entităţilor inteligente*;**
- ***construirea* unor asemenea entităţi.**

- **IA are drept scop apariţia unor calculatoare care să aibă o inteligenţă de nivel uman sau chiar mai bună.**

Exista mai multe *definiţii ale domeniului*. Acestea variază de-a lungul a *două mari dimensiuni*. Prima dimensiune este aceea a *procesului de gândire şi a raţionamentului*. Cea de-a doua adresează *comportamentul* ("behavior"). De asemenea, unele definiţii măsoară *succesul* în termenii *performanţei umane*, în timp ce altele îl măsoară relativ la un concept ideal de inteligenţă, pe care îl vom numi *raţiune*.

Aceste definiţii pot fi grupate în patru mari categorii (care indică şi patru ţeluri majore urmărite în inteligenţa artificială):

- **sisteme care gândesc ca şi fiinţele umane;**
- **sisteme care se comportă ca şi fiinţele umane;**
- **sisteme care gândesc raţional;**
- **sisteme care se comportă raţional.**

Varietate de definiţii $\Rightarrow$

- **cercetători diferiţi gândesc în mod diferit despre inteligenţa artificială;**
- **IA apare ca o ştiinţă inter-disciplinară, în care işi aduc contribuţia filozofi, psihologi, lingvişti, matematicieni, informaticieni şi ingineri.**

Prima lucrare recunoscută astăzi ca fiind de inteligenţă artificială aparţine lui Warren McCulloch şi Walter Pitts (1943). Aceştia s-au bazat pe *trei surse* şi au conceput *un model de neuroni artificiali*. Cele *trei surse* utilizate au fost:

- **cunoştinţele despre fiziologia şi funcţiile de bază ale neuronilor în creier;**
- **analiza formală a logicii propoziţionale datorate lui Russel şi Whitehead;**
- **teoria despre calcul a lui Turing.**

O altă figură extrem de influentă în inteligenţa artificială este cea a lui John McCarthy, de la Princeton. După absolvire, McCarthy se mută la Dartmouth College, care va deveni locul oficial în care s-a născut domeniul.

In timpul workshop-ului organizat la Dartmouth, în vara anului 1956, se hotărăşte şi adoptarea, pentru noul domeniu, a numelui propus de McCarthy: cel de inteligenţă artificială.

SCURT ISTORIC

Un *an istoric* în evoluţia domeniului este 1958, an în care McCarthy se mută de la Dartmouth la MIT ("Massachusetts Institute of Technology"). Aici el aduce trei contribuţii vitale, toate în acelaşi an, 1958:

- **defineşte limbajul de nivel înalt LISP, care urma să devină limbajul de programare dominant în inteligenţa artificială;**
- **introduce conceptul de *partajare* ("time-sharing");**
- **publică articolul intitulat *Programs with Common Sense* ("Programe cu bun simţ") în care descrie un program ipotetic, numit "Advice Taker", care poate fi privit ca reprezentând *primul sistem complet de inteligenţă artificială*.**

ADVICE TAKER

- **Programul era proiectat să folosească *cunoştinţe* pentru a căuta soluţii la probleme.**
- **Spre deosebire de programele de până atunci, încorpora *cunoştinţe generale despre lume*.**
- **Era conceput în aşa fel încât să poată accepta noi axiome pe parcurs ⇒ i se permitea să dobândească competenţă în noi domenii, fără a fi reprogramat.**

Acest program încorpora *principiile de bază ale reprezentării cunoştinţelor şi ale raţionamentului*, şi anume:

- **este necesar să dispunem de o reprezentare formală a lumii şi a felului în care acţiunile unui agent afectează lumea;**
- **este necesar să putem manipula aceste reprezentări cu ajutorul unor procese deductive.**

Discuţiile de început asupra domeniului s-au concentrat mai degrabă asupra *reprezentării problemelor* decât asupra *reprezentării cunoştinţelor*. Accentul s-a pus pe formularea problemei care trebuie rezolvată şi NU pe formularea resurselor care sunt disponibile programului.

- **În perioada 1969-1979 apar şi se dezvoltă *sistemele expert* (sau *sistemele bazate pe cunoştinţe*).**

Caracteristica majoră a sistemelor expert este aceea că ele se bazează pe cunoştinţele unui expert uman în domeniul care este studiat; mai exact, pe cunoştinţele expertului uman asupra *strategiilor de rezolvare* a problemelor *tipice unui domeniu*.

La baza sistemelor expert se află utilizarea în rezolvarea problemelor a unor mari cantităţi de cunoştinţe specifice domeniului.

Tot în perioada 1969-1979, numărul mare de aplicaţii referitoare la *problemele lumii reale* a determinat creşterea cererii de *scheme de reprezentare a cunoştinţelor*. Au fost astfel dezvoltate diferite *limbaje de reprezentare*. Unele dintre ele se bazau pe *logică*, cum este cazul limbajului Prolog, extrem de popular în Europa.

***Limbajele programării logice* sunt *limbaje declarative*. Un limbaj de programare declarativ scuteşte programatorul de a mai menţiona procedura exactă pe care trebuie să o execute calculatorul pentru a realiza o funcţie. Programatorii folosesc limbajul pentru *a descrie* o mulţime de *fapte* şi de *relaţii* astfel încât utilizatorul să poată *interoga* apoi sistemul pentru a obţine un anumit rezultat.**

- **Primul compilator de Prolog a fost creat în 1972 de către Philippe Roussel, la Universitatea din Marsilia.**

În perioada 1980-1988 în inteligenţa artificială începe să se lucreze *la nivel industrial*.

Tot în această perioadă, în care inteligenţa artificială devine industrie, ea işi propune noi ţeluri ambiţioase, cum ar fi acela al *înţelegerii limbajului natural*.

Începând din 1987 şi până în prezent cercetătorii se bazează mult mai mult pe teoreme riguroase şi mai puţin decât până acum pe intuiţie. Spre exemplu, domeniul *recunoaşterii limbajului* a ajuns să fie dominat de aşa-numitele "hidden Markov models" (HMM-uri).

Subdomenii ale inteligenţei artificiale

Principalele subdomenii ale inteligenţei artificiale sunt considerate a fi următoarele:

- **jocurile (bazate pe *căutarea* efectuată într-un spaţiu de stări ale problemei);**
- **raţionamentul automat şi demonstrarea teoremelor (bazate pe rigoarea şi generalitatea *logicii matematice*. Este cea mai veche ramură a inteligenţei artificiale şi cea care înregistrează cel mai mare succes. Cercetarea în domeniul demonstrării automate a teoremelor a dus la *formalizarea algoritmilor de căutare* şi la dezvoltarea unor *limbaje de reprezentare formală*, cum ar fi calculul predicatelor şi limbajul pentru programare logică Prolog).**
- **sistemele expert (care pun în evidenţă importanţa *cunoştinţelor specifice unui domeniu*).**

- **<u>înţelegerea limbajului natural şi modelarea semantică</u> (Caracteristica de bază a oricărui sistem de înţelegere a limbajului natural o constituie reprezentarea sensului propoziţiilor într-un anumit limbaj de reprezentare astfel încât aceasta să poată fi utilizată în prelucrări ulterioare).**
- **<u>planificarea şi robotica</u> (Planificarea presupune existenţa unui robot capabil să execute anumite acţiuni atomice, cum ar fi deplasarea într-o cameră plină cu obstacole).**
- **<u>învăţarea automată</u> (datorită căreia se realizează adaptarea la noi circumstanţe, precum şi detectarea şi extrapolarea unor şabloane - "patterns". Învăţarea se realizează, spre exemplu, prin intermediul aşa-numitelor *reţele neurale* sau *neuronale*. O asemenea reţea reprezintă un tip de sistem de inteligenţă artificială modelat după neuronii -celulele nervoase- dintr-un sistem nervos biologic, în încercarea de a simula modul în care creierul prelucrează informaţiile, învaţă sau îşi aduce aminte).**

INVATAREA AUTOMATA

- **Este subdomeniul IA care se ocupa de programe care invata din experienta.**

- **Invatarea statistica:**

  - **Intr-un scenariu tipic avem o masura de iesire care este cantitativa (pretul actiunilor) sau categorica (pacientul va face sau nu va face infarct) pe care vrem s-o prezicem pe baza unei multimi de caracteristici (cum ar fi dieta sau masuratorile clinice).**

  - **Avem o multime de date de antrenare corespunzator carora se observa rezultatul, precum si masuratorile caracteristicilor pentru o multime de obiecte (de ex. oameni).**

  - **Folosind aceste date construim un model de predictie – "learner" – care ne va permite sa prezicem rezultatul pentru obiecte noi, nevazute.**

  - **Acuratetea predictiei da calitatea learner-ului.**

  - **Aceata problema de invatare este numita supervizata datorita prezentei variabilei rezultat, care ghideaza procesul de invatare.**

  - **In cazul unei probleme de invatare nesupervizate, sunt observate numai caracteristicile si nu exista masuratori ale rezultatului. Sarcina programatorului este, mai degraba, aceea de a descrie felul in care datele sunt organizate (în cluster-e; cluster = grup mic, multime).**

Majoritatea tehnicilor din inteligenţa artificială folosesc pentru <u>implementarea inteligenţei</u>

- ***cunoştinţe reprezentate în mod explicit***
- ***algoritmi de căutare***
- ***învăţare automată***

- ***cunoştinţe reprezentate în mod explicit***
- ***algoritmi de căutare***

Toate aceste subdomenii ale inteligenţei artificiale au anumite trăsături în comun, şi anume:

- **O concentrare asupra problemelor care *nu răspund la soluţii algoritmice*; din această cauză tehnica de rezolvare a problemelor specifică inteligenţei artificiale este aceea de a se baza pe o *căutare euristică*.**

- **IA rezolvă probleme folosind şi *informaţie inexactă*, care *lipseşte* sau care *nu este complet definită* şi utilizează formalisme de reprezentare ce constituie pentru programator o compensaţie faţă de aceste probleme.**

- **IA foloseşte *raţionamente asupra trăsăturilor calitative semnificative ale unei situaţii*.**

- **IA foloseşte, în rezolvarea problemelor, mari cantităţi de *cunoştinţe specifice domeniului* investigat.**

Vom folosi abordarea în cadrul căreia:

- ***Reprezentarea cunoştinţelor*** **adresează problema reprezentării într-un limbaj formal, adică un limbaj adecvat pentru prelucrarea ulterioară de către un calculator, a întregii game de cunoştinţe necesare comportamentului inteligent; in functie de tehnica de reprezentare a cunostintelor aleasa se foloseste un anumit mecanism de inferenta;**

- ***căutarea*** **este o tehnică de rezolvare a problemelor care explorează în mod sistematic un spaţiu de stări ale problemei, adică de stadii succesive şi alternative în procesul de rezolvare a acesteia. (Exemple de stări ale problemei pot fi configuraţiile diferite ale tablei de şah în cadrul unui joc sau paşii intermediari într-un proces de raţionament).**

Tehnici ale IA

O tehnică a inteligenţei artificiale **este o *metodă de exploatare şi valorificare a cunoştinţelor* care, la rândul lor, ar trebui reprezentate într-un anumit mod, şi anume conform următoarelor cerinţe:**

- **Cunoştinţele trebuie să înglobeze *generalizări*. Nu este necesară reprezentarea separată a fiecărei situaţii în parte. În schimb, situaţiile care au în comun proprietăţi importante pot fi grupate laolaltă. Dacă cunoştinţele nu ar avea această proprietate, ar fi nevoie de foarte multă memorie şi de multe operaţii de actualizare. Prin urmare, o mulţime de informaţii care nu posedă această proprietate nu va reprezenta cunoştinţe, ci *date*.**

- **Cunoştinţele trebuie *să poată fi înţelese* de către oamenii care le furnizează.**

- **Cunoştinţele trebuie să poată fi uşor modificate pentru a se corecta erori şi pentru a reflecta atât schimbările din lumea înconjurătoare, cât şi schimbările din percepţia şi imaginea noastră despre aceasta.**

- **Cunoştinţele trebuie să poată fi folosite în foarte multe situaţii, chiar dacă uneori duc lipsă de acurateţe şi nu sunt complete.**

- **Cunoştinţele trebuie să poată fi folosite astfel încât să ajute la îngustarea gamei de posibilităţi care trebuie luate în consideraţie.**

➢ **<u>Scopul tehnicilor inteligenţei artificiale</u> este acela de a putea susţine *<u>o folosire eficientă a cunoştinţelor.</u>***

Un exemplu de problemă tipică în inteligenţa artificială (QUESTION ANSWERING)

Un exemplu de problemă tipică tratată de inteligenţa artificială îl constituie problema furnizării răspunsului la întrebări ("Question Answering").

- **Problema: Se citeşte de la tastatură text într-un limbaj natural (spre exemplu în limba engleză) şi apoi se răspunde unor întrebări referitoare la acel text, formulate tot în engleză.**

- **Observaţie: În cazul unei astfel de probleme este mai greu de precizat care este răspunsul corect, asa cum se va vedea în următorul exemplu:**

Exemplul nr. 1

Presupunem că textul reprezentând input-ul este dat de unica propoziţie

“*Russia massed troops on the Czech border.*”

Corespunzător unui astfel de input este posibilă existenţa ambelor dialoguri care urmează. Aceste dialoguri au reprezentat efectiv output-ul programului numit POLITICS (Carbonell, 1980):

Dialogul nr. 1

Q: Why did Russia do this?

A: Because Russia thought that it could take political control of Czechoslovakia by sending troops.

Q: What should the United States do?

A: The United States should intervene militarily.

Dialogul nr. 2

Q: Why did Russia do this?

A: Because Russia wanted to increase its political influence over Czechoslovakia.

Q: What should the United States do?

A: The United States should denounce the Russian action in the United Nations.

În programul POLITICS răspunsurile au fost construite luând în consideraţie atât textul reprezentând input-ul, cât şi un model separat referitor la concepţiile şi acţiunile diverselor entităţi politice, inclusiv cele ale fostei Uniuni Sovietice. Între cele două dialoguri acest model a fost schimbat. Atunci când modelul se schimbă şi răspunsurile sistemului se schimbă. Astfel, primul dialog a fost produs de programul POLITICS atunci când i s-a dat acestuia un model care corespunde concepţiilor unui *conservator* american. Cel de-al doilea a fost produs atunci când i s-a dat un model care corespunde concepţiilor unui *liberal* american. Prin urmare, în acest caz este foarte greu de spus ce înseamnă un *răspuns corect*. Acest lucru *depinde de model*.

- **Corectitudinea răspunsului *depinde de model*.**

- **O procedură eficientă de "răspunsuri la întrebări" trebuie să se bazeze în mod solid pe *cunoştinţe* şi pe *exploatarea computaţională a acelor cunoştinţe*.**

- **Însuşi scopul tehnicilor inteligenţei artificiale este acela de a putea susţine *o folosire eficientă a cunoştinţelor*.**

Concluzii

IA ≡ acel domeniu al informaticii care se ocupă de *automatizarea comportamentului inteligent*.

Scopul urmarit de domeniul IA

Scopul stiintific **al IA este acela al determinarii de teorii cu privire la reprezentarea cunostintelor, invatare, sisteme bazate pe reguli si cautare, teorii care sa explice diverse tipuri de inteligenta.**

Scopul ingineresc **al IA este acela de a conferi masinii abilitatea de a rezolva problem ale lumii reale.**

Principalele tehnici folosite in IA cu acest scop sunt: reprezentarea cunostintelor, invatarea automata, sistemele bazate pe reguli si cautarea in spatiul de stari.

Tehnici folosite in IA: reprezentarea, invatarea, regulile, cautarea

Reprezentarea

Toate sistemele IA au o importanta trasatura (caracteristica) de reprezentare a cunostintelor. Sistemele bazate pe reguli, sistemele bazate pe cadre si retelele semantice folosesc o secventa de *reguli if-then*, in timp ce retelele neuronale artificiale folosesc conexiuni impreuna cu ponderi ale conexiunilor.

Invatarea

Toate sistemele IA au capacitatea de a invata. Folosind aceasta capacitate, ele achizitioneaza automat cunostinte din mediul inconjurator. De exemplu, pot achizitiona regulile pentru un sistem expert bazat pe reguli sau determina ponderile adecvate ale conexiunilor intr-o retea neuronala artificiala.

Reguli

Regulile unui sistem IA pot fi implicite sau explicite. Atunci cand sunt explicite, regulile sunt create de catre un inginer specializat in ingineria cunostintelor (de pilda regulile pentru un sistem expert). Reguli implicite pot sa apara, de pilda, sub forma ponderilor conexiunilor intr-o retea neuronala.

Cautarea

Cautarea poate sa apara in multiple forme, de pilda sub forma cautarii succesiunii de stari care conduce cel mai repede la o solutie, ori sub forma cautarii unei multimi optime de ponderi ale conexiunilor intr-un ANN – prin minimizarea functiei de fitness. (ANN ≡ Artificial Neural Network).

N.B. **Acest curs introductiv are doua parti distincte: una dintre ele se refera la cautare si reprezentarea cunostintelor, cealalta la invatarea automata.**

TEHNICI DE CĂUTARE

Căutarea este o traversare *sistematică* a unui *spaţiu de soluţii* posibile ale unei probleme.

Un *spaţiu de căutare* este de obicei un graf (sau, mai exact, un arbore) în care un nod desemnează o soluţie parţială, iar o muchie reprezintă un pas în construirea unei soluţii. Scopul căutării poate fi acela de

- **a găsi *un drum* în graf de la o situaţie iniţială la una finală;**
- **a ajunge într-un nod care reprezintă situaţia finală.**

Programul

- **Programul reprezinta un agent inteligent.**
- **Agenţii cu care vom lucra vor adopta un *scop* şi vor urmări *satisfacerea* lui.**

Rezolvarea problemelor prin intermediul căutării

- **În procesul de rezolvare a problemelor, *formularea scopului*, bazată pe situaţia curentă, reprezintă primul pas.**
- **Vom considera un *scop* ca fiind o mulţime de stări ale universului, şi anume acele stări în care scopul este satisfăcut.**
- ***Acţiunile* pot fi privite ca generând tranziţii între stări ale universului.**
- **Agentul va trebui să afle care acţiuni îl vor conduce la o stare în care scopul este satisfăcut. Înainte de a face asta el trebuie să decidă ce tipuri de acţiuni şi de stări să ia în consideraţie.**
- **Procesul decizional cu privire la acţiunile şi stările ce trebuie luate în consideraţie reprezintă *formularea problemei*. Formularea problemei urmează după formularea scopului.**

- **Un agent care va avea la dispoziţie mai multe opţiuni imediate va decide ce să facă examinând mai întâi diferite *secvenţe de acţiuni* posibile, care conduc la stări de valori necunoscute, urmând ca, în urma acestei examinări, să o aleagă pe cea mai bună. Procesul de examinare a unei astfel de succesiuni de acţiuni se numeşte *căutare*.**
- **Un *algoritm de căutare* primeşte ca *input* o *problemă* şi întoarce ca *output* o *soluţie* sub forma unei succesiuni de acţiuni.**
- **Odată cu găsirea unei soluţii, acţiunile recomandate de aceasta pot fi duse la îndeplinire. Aceasta este *faza de execuţie*. Prin urmare:**
  - ✓ **agentul *formulează*, *caută* şi *execută*.**

- **După formularea unui scop şi a unei probleme de rezolvat, agentul cheamă o procedură de căutare pentru a o rezolva. El foloseşte apoi soluţia pentru a-l ghida în acţiunile sale, executând ceea ce îi recomandă soluţia ca fiind următoarea acţiune de îndeplinit şi apoi înlătură acest pas din succesiunea de acţiuni. Odată ce soluţia a fost executată, agentul va găsi un nou scop.**

Probleme şi soluţii corect definite

- ***Probleme cu o singură stare***

Elementele de bază ale definirii unei probleme sunt *stările* şi *acţiunile*. Pentru a descrie stările şi acţiunile, din punct de vedere formal, este nevoie de următoarele elemente:

- ***Starea iniţială* în care agentul ştie că se află.**
- **Mulţimea acţiunilor posibile disponibile agentului. Termenul de *operator* este folosit pentru a desemna descrierea unei acţiuni, prin specificarea stării în care se va ajunge ca urmare a îndeplinirii acţiunii respective, atunci când ne aflăm într-o anumită stare. (O formulare alternativă foloseşte o *funcţie succesor S*. Fiind dată o anumită stare *x*, *S(x)* întoarce mulţimea stărilor în care se poate ajunge din *x*, printr-o unică acţiune).**

- ***Spaţiul de stări*** **al unei probleme reprezintă mulţimea tuturor stărilor în care se poate ajunge plecând din starea iniţială, prin intermediul oricărei secvenţe de acţiuni.**
- **Un *drum* în spaţiul de stări este orice secvenţă de acţiuni care conduce de la o stare la alta.**
- ***Testul scop*** **este testul pe care un agent îl poate aplica unei singure descrieri de stare pentru a determina dacă ea este o *stare de tip scop*, adică o stare în care scopul este atins (sau realizat). Uneori există o mulţime explicită de stări scop posibile şi testul efectuat nu face decât să verifice dacă s-a ajuns în una dintre ele. Alteori, scopul este specificat printr-o proprietate abstractă şi nu prin enumerarea unei mulţimi de stări. De exemplu, în şah, scopul este să se ajungă la o stare numită "şah mat", în care regele adversarului poate fi capturat la următoarea mutare, orice ar face adversarul. S-ar putea întâmpla ca o soluţie să fie preferabilă alteia, chiar dacă amândouă ating scopul. Spre exemplu, pot fi preferate drumuri cu mai puţine acţiuni sau cu acţiuni mai puţin costisitoare.**

- ***Funcţia de cost a unui drum*** **este o funcţie care atribuie un cost unui drum. Ea este adeseori notată prin *g*. Vom considera costul unui drum ca fiind suma costurilor acţiunilor individuale care compun drumul.**

Împreună *starea iniţială*, *mulţimea operatorilor*, *testul scop* şi *funcţia de cost a unui drum* definesc o *problemă*.

- ***Probleme cu stări multiple***

Pentru definirea unei astfel de probleme trebuie specificate:

- **o *mulţime* de stări iniţiale;**
- **o mulţime de operatori care indică, în cazul fiecărei acţiuni, mulţimea stărilor în care se ajunge plecând de la orice stare dată;**
- **un test scop (la fel ca la problema cu o singură stare);**
- **funcţia de cost a unui drum (la fel ca la problema cu o singură stare).**

- ✓ **Un *operator* se aplică unei mulţimi de stări prin reunirea rezultatelor aplicării operatorului fiecărei stări din mulţime.**
- ✓ **Aici un *drum* leagă *mulţimi de stări*, iar o *soluţie* este un drum care conduce la o *mulţime de stări*, dintre care *toate sunt stări scop*.**

- **Spaţiul de stări este aici înlocuit de *spaţiul mulţimii de stări*.**

Un exemplu: Problema misionarilor si a canibalilor

Definiţie formală a problemei:

- ***Stări***: **o stare constă dintr-o secvenţă ordonată de trei numere reprezentând numărul de misionari, de canibali şi de bărci, care se află pe malul râului. Starea de pornire (iniţială) este (3,3,1).**
- ***Operatori***: **din fiecare stare, posibilii operatori trebuie să ia fie un misionar, fie un canibal, fie doi misionari, fie doi canibali, fie câte unul din fiecare şi să îi transporte cu barca. Prin urmare, există *cel mult cinci operatori*, deşi majorităţii stărilor le corespund mai puţini operatori, intrucât trebuie evitate stările interzise. (*Observaţie*: Dacă am fi ales să distingem între indivizi, în loc de cinci operatori ar fi existat 27).**
- ***Testul scop***: **să se ajungă în starea (0,0,0).**
- ***Costul drumului***: **este dat de numărul de traversări.**

Căutarea soluţiilor şi generarea secvenţelor de acţiuni

- **Rezolvarea unei probleme începe cu *starea iniţială*.**
- **Primul pas este acela de a testa dacă starea iniţială este o *stare scop*.**
- **Dacă nu, se iau în consideraţie şi alte stări. Acest lucru se realizează aplicând *operatorii* asupra stării curente şi, în consecinţă, generând o mulţime de stări. Procesul poartă denumirea de *extinderea stării*.**
- **Atunci când se generează mai multe posibilităţi, trebuie făcută o alegere relativ la cea care va fi luată în consideraţie în continuare, aceasta fiind esenţa căutării.**

- ***Alegerea referitoare la care dintre stări trebuie extinsă prima este determinată de strategia de căutare.***

- **Procesul de căutare construieşte un *arbore de căutare*, a cărui rădăcină este un *nod de căutare* corespunzând stării iniţiale. La fiecare pas, algoritmul de căutare alege un nod-frunză pentru a-l extinde.**

- **Observaţie:**

Este important să facem *distincţia între spaţiul stărilor şi arborele de căutare*. Spre exemplu, într-o problemă de căutare a unui drum pe o hartă, pot exista doar 20 de stări în spaţiul stărilor, câte una pentru fiecare oraş. Dar există un număr infinit de drumuri în acest spaţiu de stări. Prin urmare, arborele de căutare are un număr infinit de noduri. Evident, un bun algoritm de căutare trebuie să evite urmarea unor asemenea drumuri.

- **Observaţie:**

Este importantă distincţia între noduri şi stări:

- **Un nod este o structură de date folosită pentru a reprezenta arborele de căutare corespunzător unei anumite realizări a unei probleme, generată de un anumit algoritm.**
- **O stare reprezintă o configuraţie a lumii înconjurătoare.**

De aceea, nodurile au adâncimi şi părinţi, iar stările nu le au. Mai mult, este posibil ca două noduri diferite să conţină aceeaşi stare, dacă acea stare este generată prin intermediul a două secvenţe de acţiuni diferite.

Reprezentarea nodurilor în program

Există numeroase moduri de a reprezenta nodurile. În general, se consideră că un nod este o structură de date cu cinci componente:

- starea din spaţiul de stări căreia îi corespunde nodul;

- nodul din arborele de căutare care a generat acest nod (nodul părinte);

- operatorul care a fost aplicat pentru a se genera nodul;

- numărul de noduri aflate pe drumul de la rădăcină la acest nod (*adâncimea* nodului);

- costul drumului de la starea iniţială la acest nod.

Reprezentarea colecţiei de noduri care aşteaptă pentru a fi extinse

Această colecţie de noduri poartă denumirea de *frontieră*. Cea mai simplă reprezentare ar fi aceea a unei mulţimi de noduri, iar strategia de căutare ar fi o funcţie care selectează, din această mulţime, următorul nod ce trebuie extins. Deşi din punct de vedere conceptual această cale este una directă, din punct de vedere computaţional ea poate fi foarte scumpă, pentru că funcţia strategie ar trebui să se "uite" la fiecare element al mulţimii pentru a-l alege pe cel mai bun. De aceea, *vom presupune că această colecţie de noduri este implementată ca o coadă*.

Evaluarea strategiilor de căutare

Strategiile de căutare se evaluează conform următoarelor patru criterii:

- **Completitudine: dacă, atunci când o soluţie există, strategia dată garantează găsirea acesteia;**
- **Complexitate a timpului: durata de timp pentru găsirea unei soluţii;**
- **Complexitate a spaţiului: necesităţile de memorie pentru efectuarea căutării;**
- **Optimalitate: atunci când există mai multe soluţii, strategia dată să o găsească pe cea mai de calitate dintre ele.**

Căutarea neinformată

Termenul de *căutare neinformată* desemnează faptul că o strategie de acest tip nu deţine nici o informaţie despre numărul de paşi sau despre costul drumului de la starea curentă la scop. Tot ceea ce se poate face este să se distingă o stare-scop de o stare care nu este scop. Căutarea neinformată se mai numeşte şi *căutarea oarbă*.

Căutarea informată

Să considerăm, de pildă, problema găsirii unui drum de la Arad la Bucureşti, având în faţă o hartă. De la starea iniţială, Arad, există trei acţiuni care conduc la trei noi stări: Sibiu, Timişoara şi Zerind. O căutare neinformată nu are nici o preferinţă între cele trei variante. Un agent mai inteligent va observa însă că scopul, Bucureşti, se află la sud-est de Arad şi că numai *Sibiu* este în această direcţie, care reprezintă, probabil, cea mai bună alegere. Strategiile care folosesc asemenea consideraţii se numesc *strategii de căutare informată* sau *strategii de căutare euristică*.

Căutarea informată

- ***Căutarea informată*** **se mai numeşte şi** ***căutare euristică*****.**

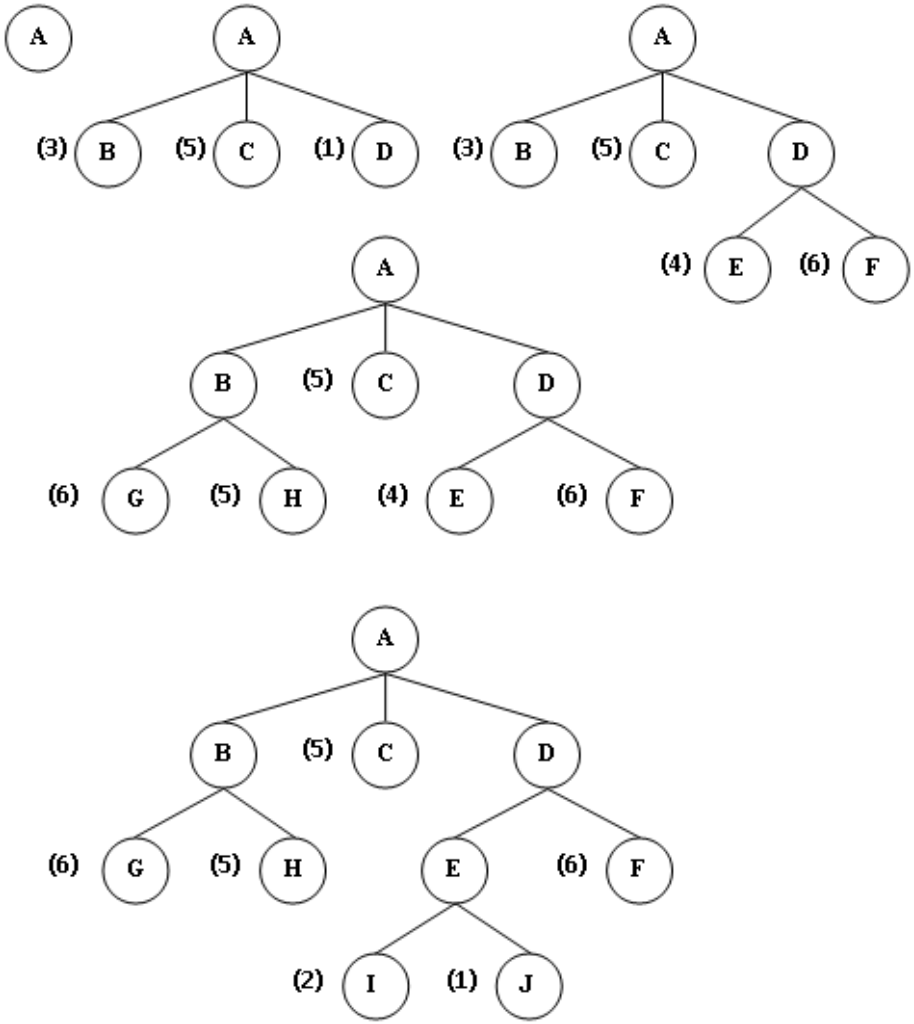
- **Euristica este o metodă de studiu şi de cercetare bazată pe descoperirea de fapte noi. În acest tip de căutare vom folosi informaţia despre spaţiul de stări. Se folosesc cunoştinţe specifice problemei şi se rezolvă probleme de optim.**

Căutarea de tip best-first

- **Procesul de căutare nu se desfăşoară în mod uniform plecând de la nodul iniţial. El înaintează în mod preferenţial de-a lungul unor noduri pe care informaţia euristică, specifică problemei, le indică ca aflându-se pe drumul cel mai bun către un scop. Un asemenea proces de căutare se numeşte *căutare euristică* sau *căutare de tip best-first*.**

- **Principiile pe care se bazează căutarea de tip best-first sunt următoarele:**
  1. **Se presupune existenţa unei funcţii euristice de evaluare, $\hat{f}$, cu rolul de a ne ajuta să decidem care nod ar trebui extins la pasul următor. Se va adopta *convenţia* că valori mici ale lui $\hat{f}$ indică nodurile cele mai bune. Această funcţie se bazează pe informaţie specifică domeniului pentru care s-a formulat problema. Este o funcţie de descriere a stărilor, cu valori reale.**
  2. **Se extinde nodul cu cea mai mică valoare a lui $\hat{f}(n)$. *În cele ce urmează, se va presupune că extinderea unui nod va produce toţi succesorii acelui nod.***

Figura următoare ilustrează începutul unei căutări de tip best-first:



Aici există iniţial un singur nod, A, astfel încât acesta va fi extins.

Pentru a nu fi induşi în eroare de o euristică extrem de optimistă, este necesar să înclinăm căutarea în favoarea posibilităţii de a ne întoarce înapoi, cu scopul de a explora drumuri găsite mai devreme. De aceea, vom adăuga lui $\hat{f}$ *un* *factor de adâncime*,

$$\hat{f}(\mathrm{n}) = \hat{g}(\mathrm{n}) + \hat{h}(\mathrm{n}),$$

unde:

- $\hat{g}(\mathrm{n})$ **este** ***o estimaţie a adâncimii lui n în graf*****, adică reprezintă lungimea celui mai scurt drum de la nodul de start la *n*;**
- $\hat{h}(\mathrm{n})$ **este** ***o evaluare euristică a nodului n*****.**

Prezentăm un algoritm de căutare generală bazat pe grafuri. Algoritmul include versiuni ale căutării de tip best-first ca reprezentând cazuri particulare.

Algoritm de căutare general bazat pe grafuri

Acest algoritm, pe care îl vom numi GraphSearch, este unul general, care permite orice tip de ordonare preferată de utilizator - euristică sau neinformată. Iată o *primă variantă* a definiţiei sale:

GraphSearch

1. **Creează un arbore de căutare, T_r, care constă numai din nodul de start n_0. Plasează pe n_0 într-o listă ordonată numită OPEN.**
2. **Creează o listă numită CLOSED, care iniţial este vidă.**
3. **Dacă lista OPEN este vidă, EXIT cu eşec.**
4. **Selectează primul nod din OPEN, înlătură-l din lista OPEN şi include-l în lista CLOSED. Numeşte acest nod *n*.**
5. **Dacă *n* este un nod scop, algoritmul se încheie cu succes, iar soluţia este cea obţinută prin urmarea în sens invers a unui drum de-a lungul arcelor din arborele T_r, de la *n* la n_0. (Arcele sunt create la pasul 6).**
6. **Extinde nodul *n*, generând o mulţime, *M*, de succesori. Include *M* ca succesori ai lui *n* în T_r, prin crearea de arce de la *n* la fiecare membru al mulţimii *M*.**
7. **Reordonează lista OPEN, fie în concordanţă cu un plan arbitrar, fie în mod euristic.**
8. **Mergi la pasul 3.**

Observaţie: Acest algoritm poate fi folosit pentru a efectua căutări de tip best-first, breadth-first sau depth-first. În cazul algoritmului *breadth-first* noile noduri sunt puse la sfârşitul listei OPEN (organizată ca o coadă), iar nodurile nu sunt reordonate. În cazul căutării de tip *depth-first* noile noduri sunt plasate la începutul listei OPEN (organizată ca o stivă). În cazul căutării de tip *best-first*, numită şi căutare euristică, lista OPEN este reordonată în funcţie de meritele euristice ale nodurilor.

Algoritmul A*

Vom particulariza algoritmul GraphSearch la un algoritm de căutare best-first care reordonează, la pasul 7, nodurile listei OPEN în funcţie de *valorile crescătoare ale funcţiei* $\hat{f}$. Această versiune a algoritmului GraphSearch se va numi Algoritmul A*.

Pentru a specifica familia funcţiilor $\hat{f}$ care vor fi folosite, introducem următoarele notaţii:

- **$h(n)$ = costul *efectiv* al drumului de cost minim dintre nodul n şi un nod-scop, luând în consideraţie toate nodurile-scop posibile şi toate drumurile posibile de la n la ele;**
- **$g(n)$ = costul unui drum de cost minim de la nodul de start n_0 la nodul n.**

Atunci, $f(n) = g(n) + h(n)$ este costul unui drum de cost minim de la n_0 la un nod-scop, drum ales dintre toate drumurile care trebuie să treacă prin nodul n.

Observaţie: $f(n_0) = h(n_0)$ reprezintă costul unui drum de cost minim nerestricţionat, de la nodul n_0 la un nod-scop.

Pentru fiecare nod n, fie $\hat{h}(n)$, numit *factor euristic*, o estimaţie a lui $h(n)$ şi fie $\hat{g}(n)$, numit *factor de adâncime*, costul drumului de cost minim până la n găsit de A* până la pasul curent. Algoritmul A* va folosi funcţia $\hat{f} = \hat{g} + \hat{h}$.

- **Observaţie:**

În definirea Algoritmului A* de până acum nu s-a ţinut cont de următoarea *problemă*: ce se întâmplă dacă graful implicit în care se efectuează căutarea nu este un arbore? Cu alte cuvinte, există mai mult decât o unică secvenţă de acţiuni care pot conduce la aceeaşi stare a lumii plecând din starea iniţială. (Există situaţii în care fiecare dintre succesorii nodului *n* îl are pe *n* ca succesor i.e. acţiunile sunt reversibile). Pentru a rezolva astfel de cazuri, pasul 6 al algoritmului GraphSearch trebuie înlocuit cu următorul pas 6' :

6'. Extinde nodul *n*, generând o mulţime, *M*, de succesori care nu sunt deja *părinţi* ai lui *n* în T_r. Instalează *M* ca succesori ai lui *n* în T_r prin crearea de arce de la *n* la fiecare membru al lui *M*.

Pentru a rezolva problema ciclurilor mai lungi, se înlocuieşte pasul 6 prin următorul pas 6":

6". Extinde nodul *n*, generând o mulţime, *M*, de succesori care nu sunt deja *strămoşi* ai lui *n* în T_r. Instalează *M* ca succesori ai lui *n* în T_r prin crearea de arce de la *n* la fiecare membru al lui *M*.

- **Observaţie:**

Pentru a verifica existenţa acestor cicluri mai lungi, trebuie văzut dacă structura de date care etichetează fiecare succesor al nodului *n* este egală cu structura de date care etichetează pe oricare dintre strămoşii nodului *n*. Pentru structuri de date complexe, acest pas poate mări complexitatea algoritmului. Pasul 6 modificat în pasul 6” face însă ca algoritmul să nu se mai învârtă în cerc, în căutarea unui drum la scop.

- **Observaţie:**

Există încă posibilitatea de a vizita aceeaşi stare a lumii via drumuri diferite. Dacă două noduri din T_r sunt etichetate cu aceeaşi structură de date, vor avea sub ele subarbori identici. Prin urmare, algoritmul va duplica anumite eforturi de căutare.

- **Pentru a preveni duplicarea efortului de căutare atunci când nu s-au impus condiţii suplimentare asupra lui $\hat{f}$, sunt necesare nişte modificări în algoritmul A*, şi anume: deoarece căutarea poate ajunge la acelaşi nod de-a lungul unor drumuri diferite, algoritmul A* generează un *graf de căutare*, notat cu G. G este structura de noduri şi de arce generată de A* pe măsură ce algoritmul extinde nodul iniţial, succesorii lui ş.a.m.d.. A* menţine şi un arbore de căutare, T_r.**
- **T_r, un subgraf al lui G, este arborele cu cele mai bune drumuri (de cost minim) produse până la pasul curent, drumuri până la toate nodurile din graful de căutare. Prin urmare, unele drumuri pot fi în graful de căutare, dar nu şi în arborele de căutare.**
- **Graful de căutare este menţinut deoarece căutări ulterioare pot găsi drumuri mai scurte, care folosesc anumite arce din graful de căutare anterior ce nu se aflau şi în arborele de căutare anterior.**

Dăm, în continuare, versiunea algoritmului A* care menţine graful de căutare. *În practică*, această versiune este folosită mai rar deoarece, de obicei, se pot impune *condiţii asupra lui* $\hat{f}$ care garantează faptul că, atunci când algoritmul A* extinde un nod, el a găsit deja drumul de cost minim până la acel nod.

Algoritmul A*

1. **Creează un graf de căutare G, constând numai din nodul iniţial n_0. Plasează n_0 într-o listă numită OPEN.**
2. **Creează o listă numită CLOSED, care iniţial este vidă.**
3. **Dacă lista OPEN este vidă, EXIT cu eşec.**
4. **Selectează primul nod din lista OPEN, înlătură-l din OPEN şi plasează-l în lista CLOSED. Numeşte acest nod *n*.**
5. **Dacă *n* este un nod scop, opreşte execuţia cu succes. Returnează soluţia obţinută urmând un drum de-a lungul pointerilor de la *n* la n_0 în G. (Pointerii definesc un arbore de căutare şi sunt stabiliţi la pasul 7).**
6. **Extinde nodul *n*, generând o mulţime, *M*, de succesori ai lui care nu sunt deja strămoşi ai lui *n* în G. Instalează aceşti membri ai lui *M* ca succesori ai lui *n* în G.**
7. **Stabileşte 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 aceşti membri ai lui *M* listei OPEN. Pentru fiecare membru, *m*, al lui *M*, care se afla deja în OPEN sau în CLOSED, redirecţionează pointerul său către *n*, dacă cel mai bun drum la *m* găsit până în acel moment trece prin *n*. Pentru fiecare membru al lui *M* care se află deja în lista CLOSED, redirecţionează pointerii fiecăruia dintre descendenţii săi din G astfel încât aceştia să ţintească înapoi de-a lungul celor mai bune drumuri până la aceşti descendenţi, găsite până în acel moment.**
8. **Reordonează lista OPEN în ordinea valorilor crescătoare ale funcţiei $\hat{f}$. (Eventuale legături între valori minimale ale lui $\hat{f}$ sunt rezolvate în favoarea nodului din arborele de căutare aflat la cea mai mare adâncime).**
9. **Mergi la pasul 3.**

Observaţie:

La pasul 7 sunt redirecţionaţi pointeri de la un nod dacă procesul de căutare descoperă un drum la acel nod care are costul mai mic decât acela indicat de pointerii existenţi. Redirecţionarea pointerilor descendenţilor nodurilor care deja se află în lista CLOSED economiseşte efortul de căutare, dar poate duce la o cantitate exponenţială de calcule. De aceea, această parte a pasului 7 de obicei nu este implementată. Unii dintre aceşti pointeri vor fi până la urmă redirecţionaţi oricum, pe măsură ce căutarea progresează. (A se vedea slide-urile cu implementarea algoritmului).

Admisibilitatea Algoritmului A*

Există anumite condiţii asupra grafurilor şi a lui $\hat{h}$ care garantează că algoritmul A*, aplicat acestor grafuri, găseşte întotdeauna drumuri de cost minim. Condiţiile asupra *grafurilor* sunt:

1. **Orice nod al grafului, dacă admite succesori, are un număr finit de succesori.**
2. **Toate arcele din graf au costuri mai mari decât o cantitate pozitivă, ε.**

   Condiţia asupra lui $\hat{h}$ este:

3. **Pentru toate nodurile *n* din graful de căutare, $\hat{h}(n) \leq h(n)$. Cu alte cuvinte, $\hat{h}$ nu supraestimează niciodată valoarea efectivă *h*. O asemenea funcţie $\hat{h}$ este uneori numită un *estimator optimist*.**

- **Observaţie:**

Este relativ uşor să se găsească, în probleme, o funcţie $\widehat{h}$ care satisface această *condiţie a limitei de jos*. De exemplu, în probleme de găsire a drumurilor în cadrul unor grafuri ale căror noduri sunt oraşe, distanţa de tip linie dreaptă de la un oraş *n* la un oraş-scop constituie o limită inferioară asupra distanţei reprezentând un drum optim de la nodul *n* la nodul-scop.

- **Cu cele trei condiţii formulate anterior, algoritmul A* garantează găsirea unui drum optim la un scop, în cazul în care există un drum la scop.**

- **Teoremă**

Atunci când sunt îndeplinite condiţiile asupra grafurilor şi asupra lui $\widehat{h}$ enunţate anterior şi cu condiţia să existe un drum de cost finit de la nodul iniţial, n_0, la un nod-scop, algoritmul A* garantează găsirea unui drum de cost minim la un scop.

- **Definiţie**

  Orice algoritm care garantează găsirea unui drum optim la scop este un algoritm *admisibil*.

- **Atunci când cele trei condiţii ale Teoremei sunt îndeplinite, A* este un algoritm admisibil. Prin extensie, vom spune că orice funcţie $\hat{h}$ care nu supraestimează pe *h* este *admisibilă*. (Estimaţia $\hat{h}$ trebuie să aibă valori cât mai apropiate de cele ale lui *h*- pentru a nu mări efortul de căutare - dar fără a supraestima pe *h*).**

- **În cele ce urmează, atunci când ne vom referi la Algoritmul A*, vom presupune că cele trei condiţii ale Teoremei sunt verificate.**

- **A* este un algoritm complet, admisibil şi optim!**

Complexitatea Algoritmului A*

S-a arătat că o creştere exponenţială va interveni, în afara cazului în care eroarea în funcţia euristică nu creşte mai repede decât logaritmul costului efectiv al drumului. Cu alte cuvinte, *condiţia pentru o creştere subexponenţială* este:

$$|h(n) - h^*(n)| \leq O(\log h^*(n)),$$

unde $h^*(n)$ este *adevăratul* cost de a ajunge de la *n* la scop.

În afară de timpul mare calculator, algoritmul A* consumă şi mult spaţiu de memorie deoarece *păstrează în memorie toate nodurile generate*.

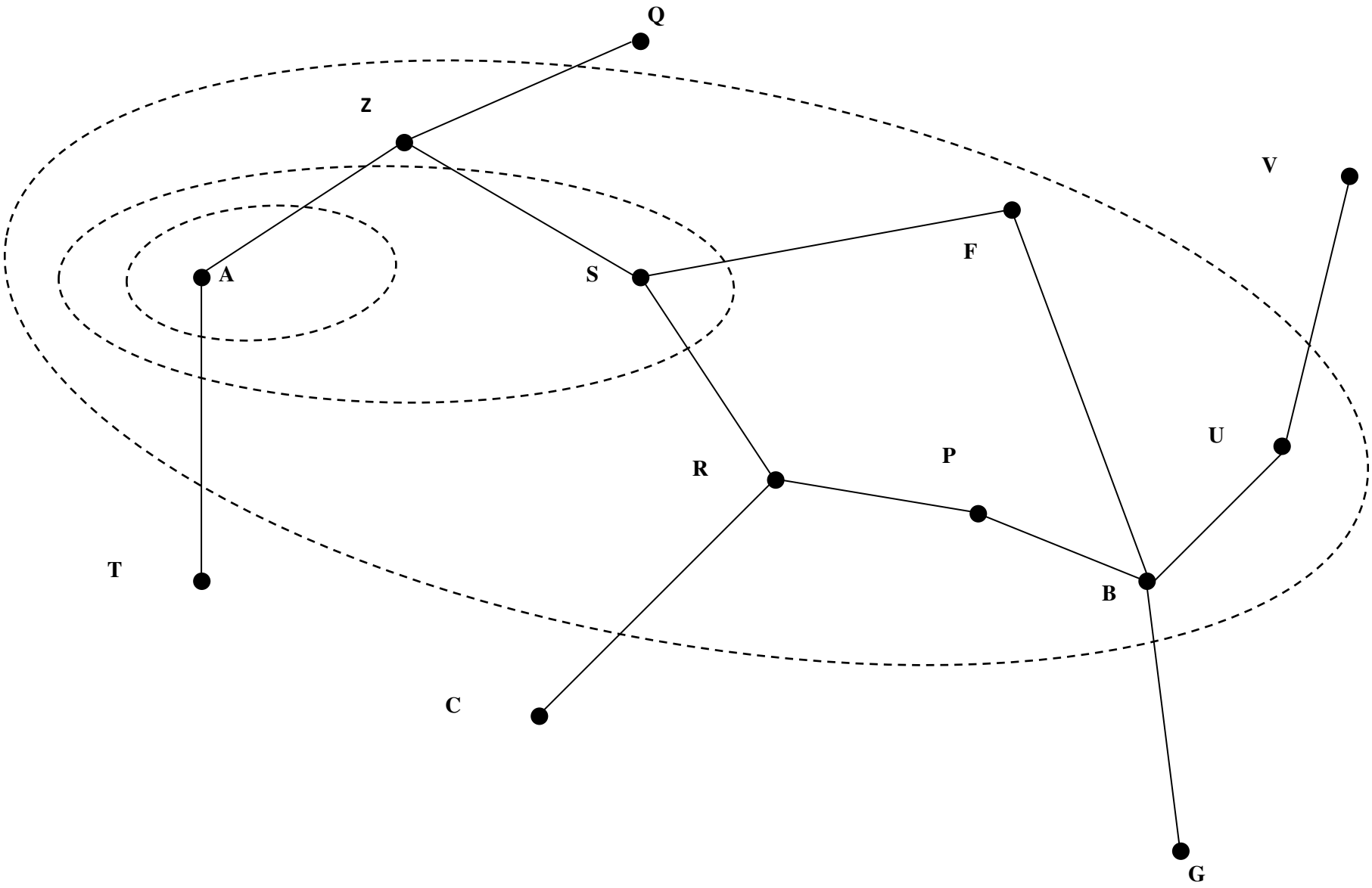
Algoritmi de căutare mai noi, de tip "memory-bounded" (cu limitare a memoriei), au reuşit să înlăture neajunsul legat de problema spaţiului de memorie folosit, fără a sacrifica optimalitatea sau completitudinea. Unul dintre aceştia este algoritmul IDA*.

Iterative Deepening A* (IDA*)

- **Algoritmul IDA* se referă la o căutare iterativă în adâncime de tip A* şi este o extensie logică a lui Iterative Deepening Search care foloseşte, în plus, informaţia euristică.**

- **În cadrul acestui algoritm fiecare iteraţie reprezintă o căutare de tip depth-first, iar căutarea de tip depth-first este modificată astfel încât ea să folosească *o limită a costului* şi nu o limită a adâncimii.**

Faptul că în cadrul algoritmului A* *f* nu descreşte niciodată de-a lungul oricărui drum care pleacă din rădăcină ne permite să trasăm, din punct de vedere conceptual, *contururi* în spaţiul stărilor. Astfel, în interiorul unui contur, toate nodurile au valoarea *f(n)* mai mică sau egală cu o aceeaşi valoare. În cazul algoritmului IDA* fiecare iteraţie extinde toate nodurile din interiorul conturului determinat de costul *f* curent, după care se trece la conturul următor. De îndată ce căutarea în interiorul unui contur dat a fost completată, este declanşată o nouă iteraţie, folosind un nou cost *f*, corespunzător următorului contur. Figura următoare prezintă căutări iterative în interiorul câte unui contur.


Q
Z
V
A
S
F
U
P
R
T
B
C
G

Algoritmul IDA* este optim cu aceleaşi amendamente ca şi A*. Deoarece este de tip depth-first *nu necesită decât un spaţiu proporţional cu cel mai lung drum pe care îl explorează.*

Dacă δ este cel mai mic cost de operator, iar f^* este costul soluţiei optime, atunci, în cazul cel mai nefavorabil, IDA* va necesita spaţiu pentru memorarea a $\frac{bf^*}{\delta}$ noduri, unde b este acelaşi factor de ramificare.

Complexitatea de timp a algoritmului depinde în mare măsură de numărul valorilor diferite pe care le poate lua funcţia euristică.

Implementarea căutarii de tip best-first

- **Vom imagina căutarea de tip best-first funcţionând în felul următor: căutarea constă dintr-un număr de subprocese "concurente", fiecare explorând alternativa sa, adică propriul subarbore. Subarborii au subarbori, care vor fi la rândul lor exploraţi de subprocese ale subproceselor, ş.a.m.d..**
- **Dintre toate aceste subprocese doar unul este activ la un moment dat şi anume cel care se ocupă de alternativa cea mai promiţătoare (adică alternativa corespunzătoare celei mai mici $\hat{f}$- valori). Celelalte procese aşteaptă până când $\hat{f}$ - valorile se schimbă astfel încât o altă alternativă devine mai promiţătoare, caz în care procesul corespunzător acesteia devine activ.**

- **Acest mecanism de activare-dezactivare poate fi privit după cum urmează: procesului corespunzător alternativei curente de prioritate maximă i se alocă un buget şi, atâta vreme cât acest buget nu este epuizat, procesul este activ. Pe durata activităţii sale, procesul îşi expandează propriul subarbore, iar în cazul atingerii unei stări-scop este anunţată găsirea unei soluţii. Bugetul acestei funcţionări este determinat de $\hat{f}$ -valoarea corespunzătoare celei mai apropiate alternative concurente.**

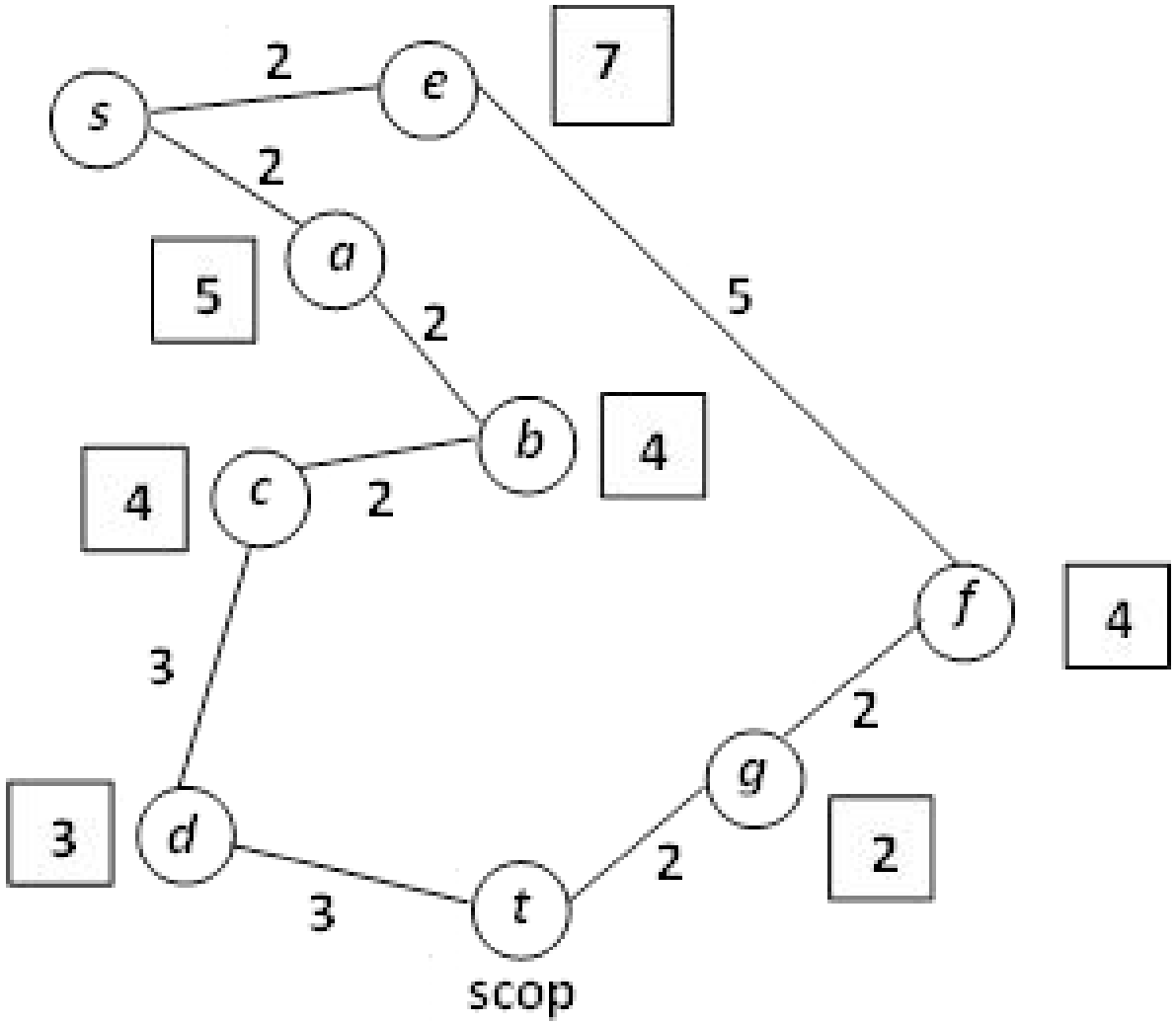
Exemplu

- **Considerăm oraşele *s, a, b, c, d, e, f, g, t* unite printr-o reţea de drumuri ca în figura care urmează. Aici fiecare drum direct între două oraşe este etichetat cu lungimea sa; numărul din căsuţa alăturată unui oraş reprezintă distanţa în linie dreaptă între oraşul respectiv şi oraşul *t*. Ne punem problema determinării celui mai scurt drum între oraşul *s* şi oraşul *t* utilizând strategia best-first.**
- **Definim, în acest scop, funcţia $\hat{h}$, bazându-ne pe distanţa în linie dreaptă între două oraşe. Astfel, pentru un oraş X, definim**

$$\hat{f}(X) = \hat{g}(X) + \hat{h}(X) = \hat{g}(X) + dist(X, t)$$

unde: $dist(X, t)$ reprezintă distanţa în linie dreaptă între X şi *t*.

- **În acest exemplu, căutarea de tip best-first este efectuată prin intermediul a două procese, P_1 si P_2, ce explorează fiecare câte una din cele două căi alternative. Calea de la *s* la *t* via nodul *a* corespunde procesului P_1, iar calea prin nodul *e* corespunde procesului P_2.**


s
e
a
b
c
d
t
f
g
2
2
2
2
3
3
5
7
5
4
4
3
2
2
scop

În stadiile inițiale, procesul P_1 este mai activ, deoarece $\hat{f}$ - valorile de-a lungul căii corespunzătoare lui sunt mai mici decât $\hat{f}$ - valorile de-a lungul celeilalte căi. Atunci când P_1 explorează c, iar procesul P_2 este încă la e, $\hat{f}(c) = \hat{g}(c) + \hat{h}(c) = 6 + 4 = 10$, $\hat{f}(e) = \hat{g}(e) + \hat{h}(e) = 2 + 7 = 9$ şi deci $\hat{f}(e) < \hat{f}(c)$. În acest moment, situaţia se schimbă: procesul P_2 devine activ, iar procesul P_1 intră în aşteptare. În continuare, $\hat{f}(c) = 10$, $\hat{f}(f) = 11$, $\hat{f}(c) < \hat{f}(f)$ şi deci P_1 devine activ şi P_2 intră în aşteptare. Pentru că $\hat{f}(d) = 12 > 11$, procesul P_1 va reintra în aşteptare, iar procesul P_2 va rămâne activ până când se va atinge starea scop *t*.

Căutarea schiţată mai sus porneşte din nodul iniţial şi este continuată cu generarea unor noduri noi, conform relaţiei de succesiune. În timpul acestui proces, este generat un arbore de căutare, a cărui rădăcină este nodul de start. Acest arbore este expandat în direcţia cea mai promiţătoare conform $\hat{f}$ - valorilor, până la găsirea unei soluţii.

JOCURILE CA PROBLEME DE CĂUTARE

- **Jocurile reprezintă o arie de aplicaţie pentru algoritmii euristici.**

- **Jocurile de două persoane sunt în general complicate datorită existenţei unui oponent ostil şi imprevizibil. De aceea ele sunt interesante din punctul de vedere al dezvoltării euristicilor, dar aduc multe dificultăţi în dezvoltarea şi aplicarea algoritmilor de căutare.**

- **Oponentul introduce *incertitudinea*, întrucât nu se ştie niciodată ce va face acesta la pasul următor.**

- **În esenţă, toate programele referitoare la jocuri trebuie să trateze aşa-numita *problemă de contingenţă*. (Aici *contingenţă* are sensul de *întâmplare*).**

- **Incertitudinea care intervine în cazul jocurilor nu este de aceeaşi natură cu cea introdusă, de pildă, prin aruncarea unui zar sau cu cea determinată de starea vremii. Oponentul va încerca, pe cât posibil, să facă mutarea cea mai puţin benignă, în timp ce zarul sau vremea sunt presupuse a nu lua în consideraţie scopurile agentului.**

- **Complexitatea jocurilor introduce *un tip de incertitudine complet nou*. Astfel, incertitudinea se naşte nu datorită faptului că există informaţie care lipseşte, ci datorită faptului că jucătorul nu are timp să calculeze consecinţele exacte ale oricărei mutări. Din acest punct de vedere, jocurile se aseamănă infinit mai mult cu lumea reală decât problemele de căutare standard.**

JOCURI DE DOUA PERSOANE CU INFORMATIE COMPLETA

Ne vom referi la tehnici de joc corespunzătoare unor *jocuri de două persoane cu informaţie completă*, cum ar fi şahul.

În cazul jocurilor interesante, arborii rezultaţi sunt mult prea complecşi pentru a se putea realiza o căutare exhaustivă, astfel încât sunt necesare abordări de o natură diferită. Una dintre metodele clasice se bazează pe „principiul minimax", implementat în mod eficient sub forma Algoritmului Alpha-Beta (bazat pe aşa-numita *tehnică de alpha-beta retezare*).

- **Întrucât, în cadrul unui joc, există, de regulă, limite de timp, jocurile *penalizează ineficienţa* extrem de sever. Astfel, dacă o implementare a căutării de tip A*, care este cu 10% mai puţin eficientă, este considerată satisfăcătoare, un program pentru jocul de şah care este cu 10% mai puţin eficient în folosirea timpului disponibil va duce la pierderea partidei. Din această cauză, studiul nostru se va concentra asupra *tehnicilor de alegere a unei bune mutări atunci când timpul este limitat.***

- ***Tehnica de "retezare"* ne va permite să ignorăm porţiuni ale arborelui de căutare care nu pot avea nici un rol în stabilirea alegerii finale, iar *funcţiile de evaluare euristice* ne vor permite să aproximăm utilitatea reală a unei stări fără a executa o căutare completă.**

O definire formală a jocurilor

Tipul de jocuri la care ne vom referi în continuare este acela al jocurilor de două persoane cu informaţie perfectă sau completă. În astfel de jocuri:

- **există doi jucători care efectuează mutări în mod alternativ;**
- **ambii jucători dispun de informaţia completă asupra situaţiei curente a jocului;**
- **jocul se încheie atunci când este atinsă o poziţie calificată ca fiind "terminală" de către regulile jocului - spre exemplu, "mat" în jocul de şah.**

Un asemenea joc poate fi reprezentat printr-un *arbore de joc* în care nodurile corespund situaţiilor (stărilor), iar arcele corespund mutărilor. Situaţia iniţială a jocului este reprezentată de nodul rădăcină, iar frunzele arborelui corespund poziţiilor terminale.

Un joc poate fi definit, în mod formal, ca fiind *un anumit tip de problemă de căutare având următoarele componente*:

- ***starea iniţială*, care include poziţia de pe tabla de joc şi o indicaţie referitoare la cine face prima mutare;**
- ***o mulţime de operatori*, care definesc mişcările permise ("legale") unui jucător;**
- ***un test terminal*, care determină momentul în care jocul ia sfârşit;**
- ***o funcţie de utilitate* (numită şi *funcţie de plată*), care acordă o valoare numerică rezultatului unui joc; în cazul jocului de şah, spre exemplu, rezultatul poate fi *câştig, pierdere* sau *remiză*, situaţii care pot fi reprezentate prin valorile 1, -1 sau 0.**

- **Vom lua în consideraţie cazul general al unui joc cu doi jucători, pe care îi vom numi MAX şi respectiv MIN.**

- **MAX va face prima mutare, după care jucătorii vor efectua mutări pe rând, până când jocul ia sfârşit.**

- **La finalul jocului vor fi acordate puncte jucătorului câştigător (sau vor fi acordate anumite penalizări celui care a pierdut).**

- **Dacă un joc ar reprezenta o problemă standard de căutare, atunci acţiunea jucătorului MAX ar consta din căutarea unei secvenţe de mutări care conduc la o stare terminală reprezentând o stare câştigătoare (conform funcţiei de utilitate) şi din efectuarea primei mutări aparţinând acestei secvenţe.**
- **Acţiunea lui MAX interacţionează însă cu cea a jucătorului MIN. Prin urmare, MAX trebuie să găsească *o strategie* care va conduce la o stare terminală câştigătoare, indiferent de acţiunea lui MIN. Această strategie include mutarea corectă a lui MAX corespunzătoare fiecărei mutări posibile a lui MIN.**
- **În cele ce urmează, vom arăta cum poate fi găsită strategia optimă (sau raţională), deşi în realitate nu vom dispune de timpul necesar pentru a o calcula.**

Algoritmul Minimax

- **JUCATORI: MIN şi MAX**

- **MAX reprezintă jucătorul care încearcă să câştige sau să îşi maximizeze avantajul avut.**
- **MIN este oponentul care încearcă să minimizeze scorul lui MAX.**
- **Se presupune că MIN foloseşte aceeaşi informaţie şi încearcă întotdeauna să se mute la acea stare care este cea mai nefavorabilă lui MAX.**

- **Algoritmul Minimax este conceput pentru a determina strategia optimă corespunzătoare lui MAX şi, în acest fel, pentru a decide care este cea mai bună primă mutare.**

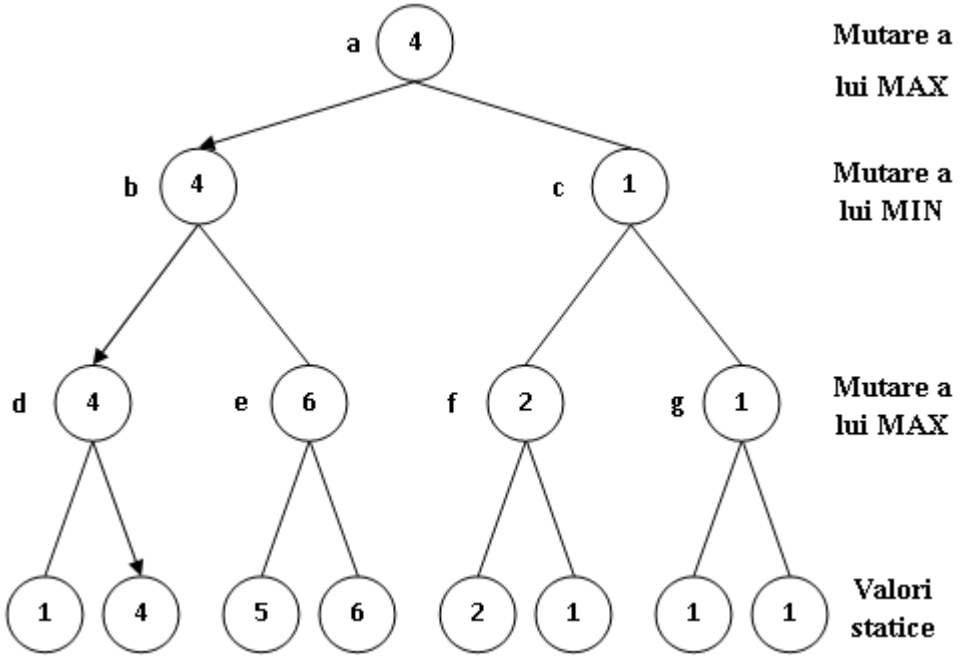
Algoritmul Minimax

1. **Generează întregul arbore de joc, până la stările terminale.**
2. **Aplică funcţia de utilitate fiecărei stări terminale pentru a obţine valoarea corespunzătoare stării.**
3. **Deplasează-te înapoi în arbore, de la nodurile-frunze spre nodul-rădăcină, determinând, corespunzător fiecărui nivel al arborelui, valorile care reprezintă utilitatea nodurilor aflate la acel nivel. Propagarea acestor valori la niveluri anterioare se face prin intermediul nodurilor- părinte succesive, conform următoarei reguli:**
   - **dacă starea-părinte este un nod de tip MAX, atribuie-i maximul dintre valorile avute de fiii săi;**
   - **dacă starea-părinte este un nod de tip MIN, atribuie-i minimul dintre valorile avute de fiii săi.**
4. **Ajuns în nodul-rădăcină, alege pentru MAX acea mutare care conduce la valoarea maximă.**

- **Observaţie:**

  Decizia luată la pasul 4 al algoritmului se numeşte *decizia minimax*, întrucât ea maximizează utilitatea, în ipoteza că oponentul joacă perfect cu scopul de a o minimiza.

Un arbore de căutare cu valori minimax determinate conform Algoritmului Minimax este cel din figura următoare:



Valorile poziţiilor de la ultimul nivel sunt determinate de către funcţia de utilitate şi se numesc *valori statice*. Valorile minimax ale nodurilor interne sunt calculate în mod dinamic, în manieră bottom-up, nivel cu nivel, până când este atins nodul-rădăcină. Valoarea rezultată, corespunzătoare acestuia, este 4 şi, prin urmare, cea mai bună mutare a lui MAX din poziţia *a* este *a-b*. Cel mai bun răspuns al lui MIN este *b-d*. Această secvenţă a jocului poartă denumirea de *variaţie principală*. Ea defineşte jocul optim de tip minimax pentru ambele părţi.

- **Se observă că valoarea poziţiilor de-a lungul variaţiei principale nu variază. Prin urmare, mutările corecte sunt cele care *conservă valoarea jocului*.**

- **Complexitate:**

Dacă adâncimea maximă a arborelui este *m* şi dacă există *b* "mutări legale" la fiecare punct, atunci complexitatea de timp a Algoritmului Minimax este $O(b^m)$. Algoritmul reprezintă o căutare de tip depth-first (deşi aici este sugerată o implementare bazată pe recursivitate şi nu una care foloseşte o coadă de noduri), astfel încât cerinţele sale de spaţiu sunt numai liniare în *m* şi *b*.

Observaţie:

În cazul jocurilor reale, cerinţele de timp ale algoritmului sunt total nepractice, dar acest algoritm stă la baza atât a unor metode mai realiste, cât şi a analizei matematice a jocurilor.

Observaţie:

Întrucât, pentru majoritatea jocurilor interesante, arborele de joc nu poate fi alcătuit în mod exhaustiv, au fost concepute diverse metode care se bazează pe căutarea efectuată numai într-o anumită porţiune a arborelui de joc. Printre acestea se numără şi tehnica Minimax, care, în majoritatea cazurilor, va căuta în arborele de joc numai *până la o anumită adâncime*, de obicei constând în numai câteva mutări.

- ✓ **Algoritmul general Minimax a fost amendat în două moduri: funcţia de utilitate a fost înlocuită cu o *funcţie de evaluare*, iar testul terminal a fost înlocuit de către un aşa-numit *test de tăiere*.**

- **Cea mai directă abordare a problemei deţinerii controlului asupra cantităţii de căutare care se efectuează este aceea de a fixa o limită a adâncimii, astfel încât testul de tăiere să aibă succes pentru toate nodurile aflate la sau sub adâncimea *d*. Limita de adâncime va fi aleasă astfel încât cantitatea de timp folosită să nu depăşească ceea ce permit regulile jocului. O abordare mai robustă a acestei probleme este aceea care aplică „iterative deepening”. În acest caz, atunci când timpul expiră, programul întoarce mutarea selectată de către cea mai adâncă căutare completă.**

Funcţii de evaluare

O *funcţie de evaluare* întoarce o *estimaţie*, realizată dintr-o poziţie dată, a utilităţii aşteptate a jocului. Ea are la bază evaluarea şanselor de câştigare a jocului de către fiecare dintre părţi, pe baza calculării caracteristicilor unei poziţii. Performanţa unui program referitor la jocuri este extrem de dependentă de calitatea funcţiei de evaluare utilizate.

- **Funcţia de evaluare trebuie să îndeplinească anumite condiţii evidente:**
  - ✓ **trebuie să concorde cu funcţia de utilitate în ceea ce priveşte stările terminale;**
  - ✓ **calculele efectuate nu trebuie să dureze prea mult;**
  - ✓ **trebuie să reflecte în mod corect şansele efective de câştig.**

- **O valoare a funcţiei de evaluare acoperă mai multe poziţii diferite, grupate laolaltă într-o *categorie* de poziţii etichetată cu o anumită valoare. (Spre exemplu, în jocul de şah, fiecare *pion* poate avea valoarea 1, un *nebun* poate avea valoarea 3 şamd.. În poziţia de deschidere evaluarea este 0 şi toate poziţiile până la prima captură vor avea aceeaşi evaluare. Dacă MAX reuşeşte să captureze un nebun fără a pierde o piesă, atunci poziţia rezultată va fi evaluată la valoarea 3. Toate poziţiile de acest fel ale lui MAX vor fi grupate într-o *categorie* etichetată cu "3").**

- **Funcţia de evaluare trebuie să reflecte şansa ca o poziţie aleasă la întâmplare dintr-o asemenea categorie să conducă la câştig (sau la pierdere sau la remiză) pe baza experienţei anterioare (VEZI tehnici de învăţare).**

- **Funcţia de evaluare cel mai frecvent utilizată presupune că valoarea unei piese poate fi stabilită *independent* de celelalte piese existente pe tablă. Un asemenea tip de funcţie de evaluare se numeşte *funcţie liniară ponderată*, întrucât are o expresie de forma**

$$w_1 f_1 + w_2 f_2 + \ldots + w_n f_n,$$

unde valorile w_i, $i = \overline{1,n}$ reprezintă ponderile, iar f_i, $i = \overline{1,n}$ sunt caracteristicile unei anumite poziţii. În cazul jocului de şah, spre exemplu w_i, $i = \overline{1,n}$ ar putea fi valorile pieselor (1 pentru pion, 3 pentru nebun etc.), iar f_i, $i = \overline{1,n}$ ar reprezenta numărul pieselor de un anumit tip aflate pe tabla de şah.

În construirea formulei liniare trebuie mai întâi alese caracteristicile, operaţie urmată de ajustarea ponderilor până în momentul în care programul joacă suficient de bine. Această a doua operaţie poate fi automatizată punând programul să joace multe partide cu el însuşi, dar alegerea unor caracteristici adecvate nu a fost încă realizată în mod automat.

(caracteristica ≡ feature)

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: 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;**
  - ***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.**

- ***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.

- **Alpha este scorul cel mai prost pe care îl poate obţine MAX, presupunând că MIN joacă perfect.**

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. 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, sunt:**

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

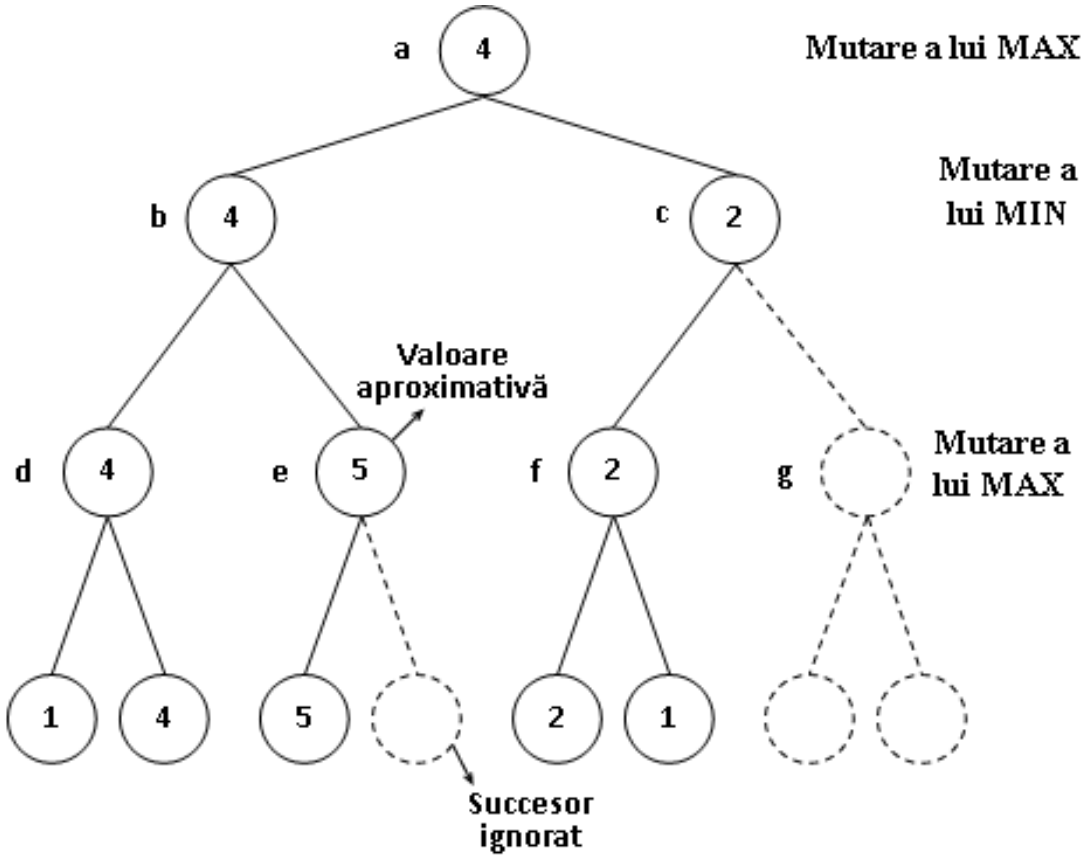
V(P, alpha, beta) > beta, dacă V (P) > beta,

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).

Figura următoare ilustrează acţiunea Algoritmului Alpha-Beta în cazul arborelui anterior. 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.



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.

Corespunzător arborelui din figură procesul de căutare decurge după cum urmează:

1. **Începe din poziţia *a*.**
2. **Mutare la *b*.**
3. **Mutare la *d*.**
4. **Alege valoarea maximă a succesorilor lui *d*, ceea ce conduce la V(*d*) = 4.**
5. **Întoarce-te în nodul *b* şi execută o mutare de aici la *e*.**
6. **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.**

Un alt exemplu

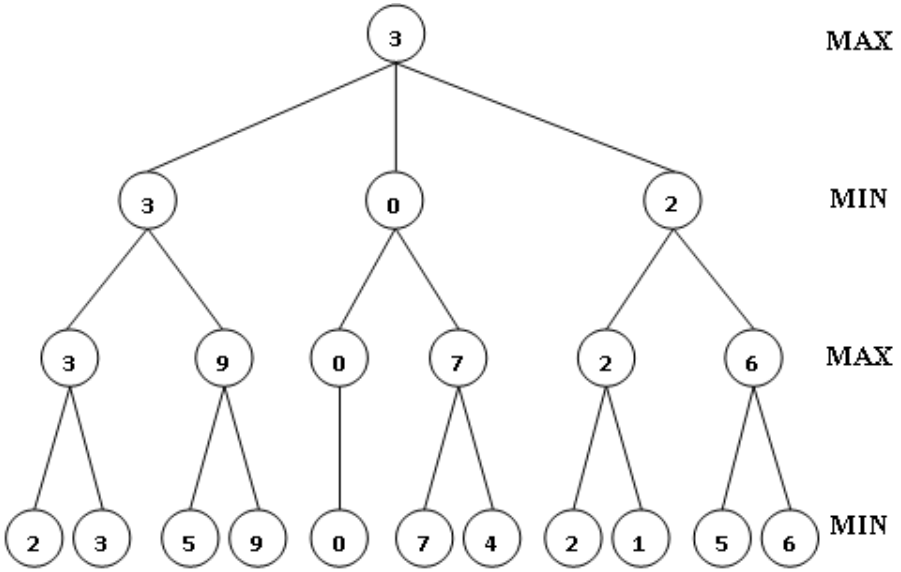


Fig. 1 (Alg. Minimax)

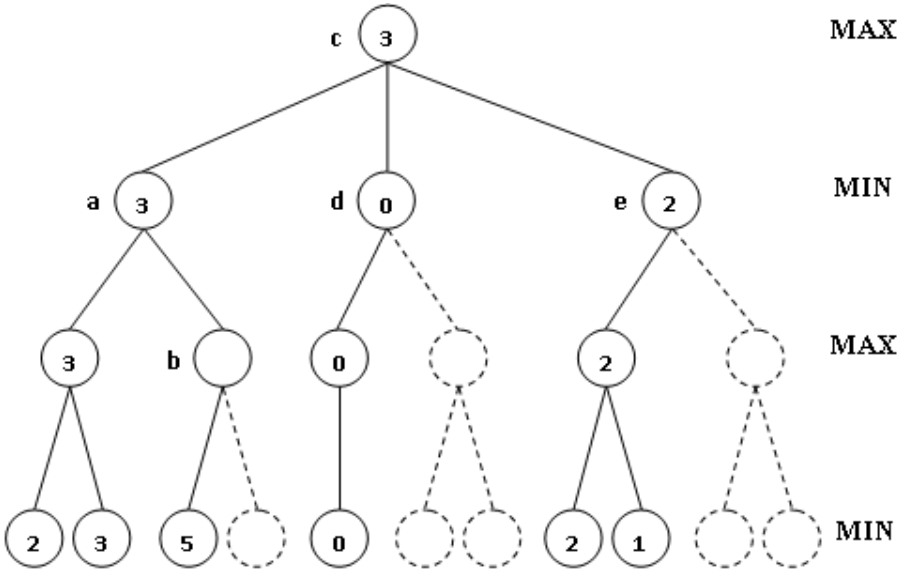


Fig. 2 (Alg. Alpha-Beta)

În Fig. 2: ***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.**

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.

Tipuri de cunoştinţe

Cunoştinţe relaţionale simple

- **Cea mai simplă modalitate de reprezentare a faptelor declarative constă în folosirea unei mulţimi de relaţii de acelaşi tip cu cele utilizate în sistemele de baze de date.**
- **Cunoştinţele relaţionale din acest tabel corespund unei mulţimi de atribute şi de valori asociate, care împreună descriu obiectele bazei de cunoştinţe.**

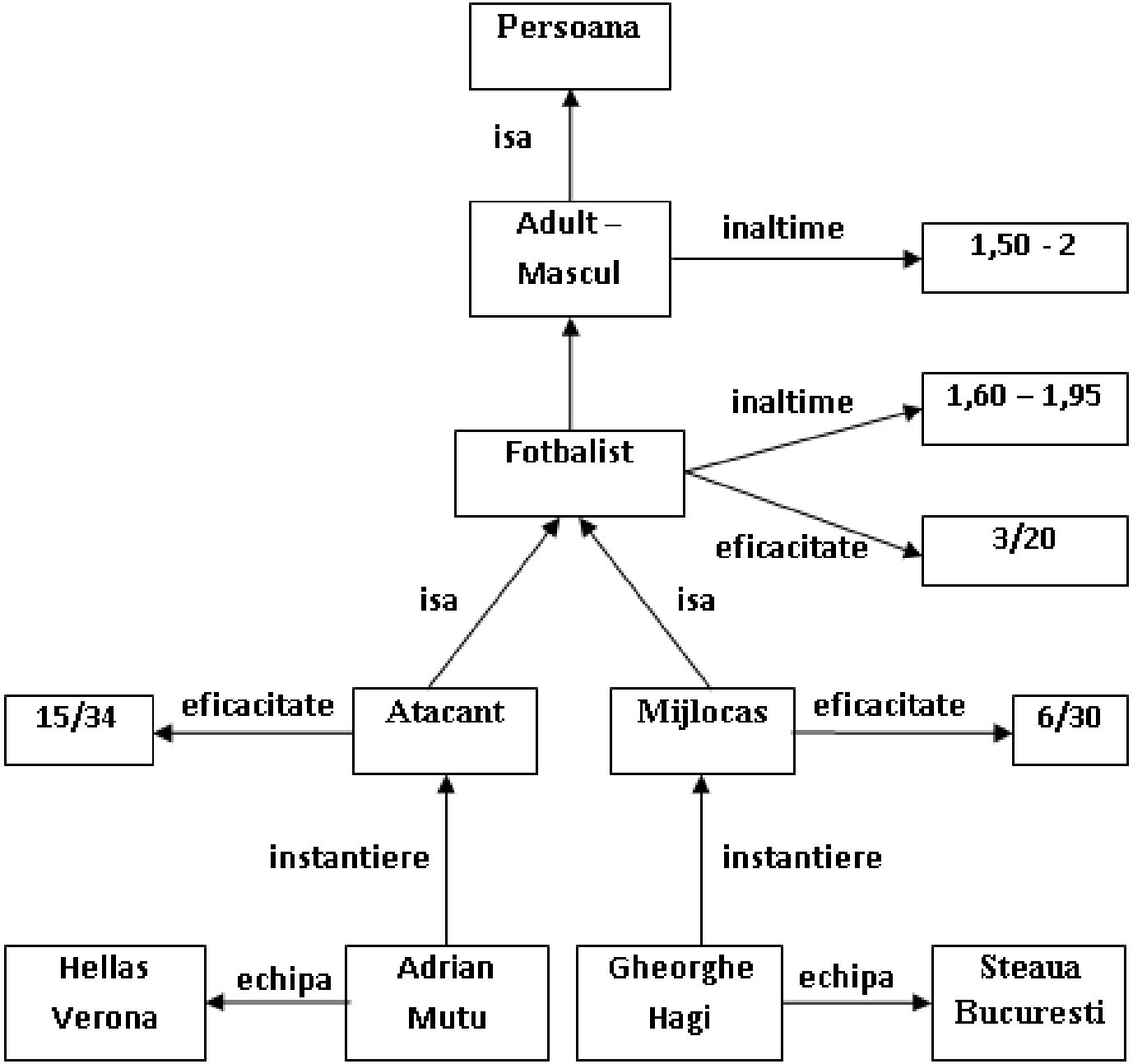
Student	Vârstă	An de studiu	Note la informatică
Popescu Andrei	18	I	8-9
Ionescu Maria	18	I	9-10
Hristea Oana	20	I	7-8
Pârvu Ana	19	II	8-9
Savu Andrei	19	II	7-8
Popescu Ana	20	III	9-10

Cunoştinţe care se moştenesc

- **Este posibil ca reprezentarea de bază să fie îmbogăţită cu mecanisme de inferenţă care operează asupra structurii reprezentării.**
- **Pentru ca această modalitate de reprezentare să fie eficientă, structura trebuie proiectată în aşa fel încât ea să corespundă mecanismelor de inferenţă dorite.**
- **Una dintre cele mai utilizate forme de inferenţă este *moştenirea proprietăţilor*, prin care elemente aparţinând anumitor *clase* *moştenesc* *atribute şi valori* provenite de la clase mai generale, în care sunt incluse.**
- **Pentru a admite moştenirea proprietăţilor, *obiectele* trebuie să fie organizate în *clase*, iar *clasele* trebuie să fie aranjate în cadrul unei *ierarhii*.**

- **În fig. care urmează, sunt reprezentate cunoştinţe legate de jocul de fotbal, cunoştinţe organizate într-o structură de acest tip. În această reprezentare, liniile desemnează atribute. Nodurile figurate prin dreptunghiuri reprezintă obiecte şi valori ale atributelor obiectelor.**
  - ✓ **Aceste valori pot fi, la rândul lor, privite ca obiecte având atribute şi valori ş.a.m.d..**
  - ✓ **Săgeţile corespunzătoare liniilor sunt orientate de la un obiect la valoarea lui (de-a lungul liniei desemnând atributul corespunzător).**

- **În exemplul din figura, toate obiectele şi majoritatea atributelor care intervin corespund domeniului sportiv al jocului de fotbal şi nu au o semnificaţie generală. Singurele două excepţii sunt atributul *isa*, utilizat pentru a desemna *incluziunea între clase* şi atributul *instanţiere*, folosit pentru a arăta *apartenenţa la o clasă*. Aceste două atribute, extrem de folosite, se află la baza *moştenirii proprietăţilor* ca tehnică de inferenţă.**


Persoana
isa
Adult – Mascul
inaltime
1,50 - 2
inaltime
1,60 – 1,95
Fotbalist
eficacitate
3/20
isa
isa
15/34
eficacitate
Atacant
Mijlocas
eficacitate
6/30
instantiere
instantiere
Hellas Verona
echipa
Adrian Mutu
Gheorghe Hagi
echipa
Steaua Bucuresti

Utilizând această tehnică de inferenţă, *baza de cunoştinţe* poate asigura atât regăsirea faptelor care au fost memorate în mod explicit, precum şi a faptelor care derivă din cele memorate în mod explicit, ca în următorul exemplu:

eficacitate(Adrian_Mutu) = 15/34

Este una dintre cele mai folosite tehnici de inferenţă!

- **În acest exemplu, structura corespunzătoare reprezintă o *structură de tip* "*slot-and-filler*". Ea mai poate fi privită şi ca o *reţea semantică* sau ca o *colecţie de cadre*.**
- **În cel din urmă caz (*colecţie de cadre)*, fiecare cadru individual reprezintă colecţia atributelor şi a valorilor asociate cu un anumit nod.**
- **O diferenţiere exactă a acestor tipuri de reprezentări este greu de făcut. În general, termenul de sistem de cadre implică existenţa unei mai mari structurări a atributelor şi a mecanismelor de inferenţă care le pot fi aplicate decât în cazul reţelelor semantice.**

Cunoştinţe inferenţiale

- **Puterea logicii tradiţionale este adesea utilă pentru a se descrie toate inferenţele necesare.**
- **Astfel de cunoştinţe nu sunt utile decât în prezenţa unei proceduri de inferenţă care să le poată exploata.**
- **Există multe asemenea proceduri, dintre care unele fac raţionamente de tipul "*înainte*", de la fapte date către concluzii, iar altele raţionează "*înapoi*", de la concluziile dorite la faptele date. Una dintre procedurile cele mai folosite de acest tip este *rezoluţia*, care foloseşte strategia contradicţiei.**
- **În general, logica furnizează o structură puternică în cadrul căreia sunt descrise legăturile dintre valori. Ea se combină adesea cu un alt limbaj puternic de descriere, cum ar fi o ierarhie de tip *isa*.**

Cunoştinţe procedurale

- **Reprezentarea cunoştinţelor descrisă până în prezent s-a concentrat asupra faptelor statice, declarative.**
- **Un alt tip de cunoştinţe extrem de utile sunt *cunoştinţele procedurale sau operaţionale*, care specifică ce anume trebuie făcut şi când.**
- **Cea mai simplă modalitate de reprezentare a cunoştinţelor procedurale este cea sub formă de cod, într-un anumit limbaj de programare.**
- **În acest caz, maşina foloseşte cunoştinţele atunci când execută codul pentru a efectua o anumită sarcină.**
- **Acest mod de reprezentare a cunoştinţelor procedurale nu este însă cel mai fericit din punctul de vedere al adecvării inferenţiale, precum şi al eficienţei în achiziţie.**

- **Cea mai folosită tehnică de reprezentare a cunoştinţelor procedurale în programele de inteligenţă artificială este aceea a utilizării *regulilor de producţie*.**
- **Atunci când sunt îmbogăţite cu informaţii asupra felului în care trebuie să fie folosite, regulile de producţie sunt mai procedurale decât alte metode existente de reprezentare a cunoştinţelor.**
- **Regulile de producţie, numite şi reguli de tip if-then, sunt instrucţiuni condiţionale, care pot avea diverse interpretări, cum ar fi:**
  - **if precondiţie P then concluzie C**
  - **if situaţie S then acţiune A**
  - **if condiţiile C1 şi C2 sunt verificate then condiţia C nu este verificată**
- **Regulile de producţie sunt foarte utilizate în proiectarea *sistemelor expert*.**

- **Regulile de tip if-then adesea definesc relaţii logice între conceptele aparţinând domeniului problemei. Relaţiile pur logice pot fi caracterizate ca aparţinând aşa-numitelor *cunoştinţe categorice*, adică acelor cunoştinţe care vor fi întotdeauna adevărate.**

- **În unele domenii, cum ar fi diagnosticarea în medicină, predomină cunoştinţele "*moi*" sau *probabiliste*. În cazul acestui tip de cunoştinţe, regularităţile empirice sunt valide numai până la un anumit punct (adesea, dar nu întotdeauna). În astfel de cazuri, regulile de producţie sunt modificate prin adăugarea la interpretarea lor logică a unei calificări de verosimilitate, obţinându-se reguli de forma:**

if conditie A then concluzie B cu certitudinea F

unde:

F = *factor de certitudine*, *măsură a încrederii* sau *certitudine subiectivă*

Pentru calculul lui F: statistica Bayesiana

Reprezentarea cunoştinţelor în sistemele expert

- **Un *sistem expert* este un program care se comportă ca un expert într-un domeniu relativ restrans.**
- **Caracteristica majoră a sistemelor expert, numite şi *sisteme bazate pe cunoştinţe*, este aceea că ele se bazează pe cunoştinţele unui expert uman în domeniul care este studiat.**
- **La baza sistemelor expert se află utilizarea în rezolvarea problemelor a unor mari cantităţi de cunoştinţe specifice domeniului.**

Alte caracteristici ale sistemului expert:

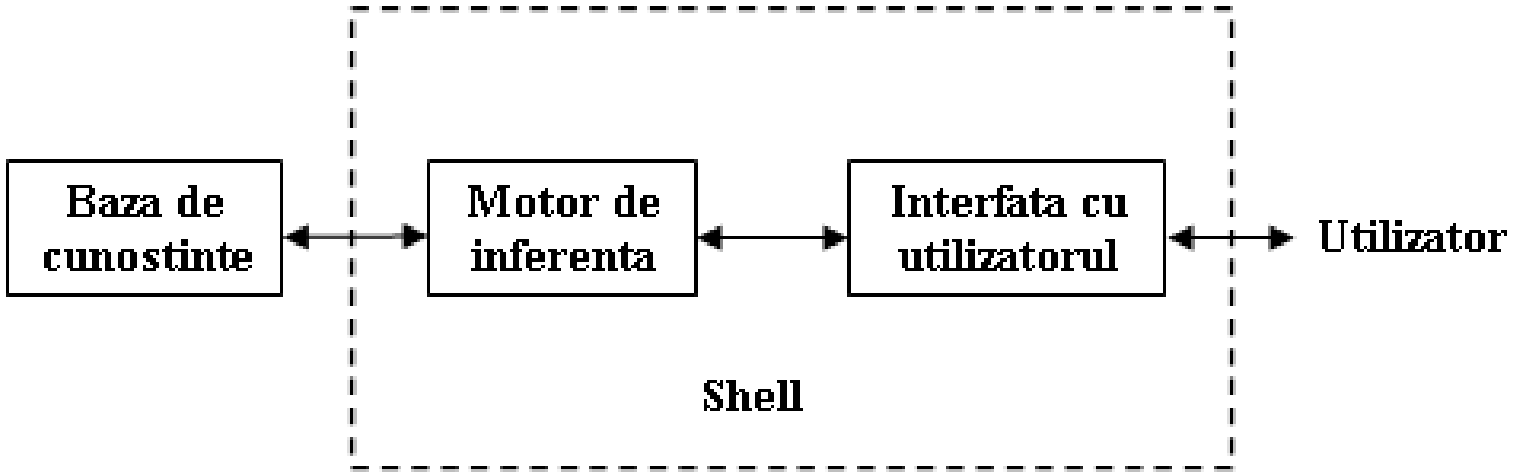
- **să fie capabil *să explice* comportamentul său şi deciziile luate - la fel cum o fac experţii umani - prin generarea răspunsului pentru două tipuri de întrebări ale utilizatorului:**
  - ✓ **întrebare de tipul "*cum*": *Cum* ai ajuns la această concluzie?**
  - ✓ **întrebare de tipul "*de ce*": *De ce* te interesează această informaţie?**

- **să lucreze cu *incertitudinea* sau *starea de a fi incomplet* - informaţii incerte, incomplete sau care lipsesc; relaţii aproximative în domeniul problemei (de ex. efectul unui medicament asupra stării pacientului).**

Structura de bază a unui sistem expert

Un sistem expert conţine trei module principale, şi anume:

- **o bază de cunoştinţe;**
- **un motor de inferenţă;**
- **o interfaţă cu utilizatorul.**



Concluzii

- **Regulile if-then formează lanţuri de forma**

  `informatie input →...→ informatie dedusa`

- **Informaţia de tip input mai poartă denumirea de *date* sau *manifestări*.**
- **Informaţia dedusă constituie *ipotezele* care trebuie demonstrate sau *cauzele manifestărilor* sau *diagnostice* sau *explicaţii*.**
- **Atât înlănţuirea înainte, cât şi cea înapoi (ca metode de inferenţă) presupun *căutare*, dar direcţia de căutare este diferită pentru fiecare în parte.**
- **Înlănţuirea înapoi execută o căutare de la scopuri înspre date, din care cauză se spune despre ea că este *orientată către scop*.**
- **Înlănţuirea înainte caută pornind de la date înspre scopuri, fiind *orientată către date*.**

Clase de metode pentru reprezentarea cunoştinţelor

Principalele tipuri de reprezentări ale cunoştinţelor sunt reprezentările *bazate pe logică* şi cele de tip "*slot-filler*" ("deschizătură-umplutură").

Reprezentările bazate pe logică **aparţin unor două mari categorii, în funcţie de instrumentele folosite în reprezentare, şi anume:**

- **Logica - mecanismul principal îl constituie inferenţa logică.**
- **Regulile (folosite, de pildă, în sistemele expert) - principalele mecanisme sunt "*înlănţuirea înainte*" şi "*înlănţuirea înapoi*". O regulă este similară unei implicaţii logice, dar nu are o valoare proprie (regulile sunt *aplicate*, ele nu au una dintre valorile „true" sau „false").**

Reprezentările de tip slot-filler **folosesc două categorii diferite de structuri:**

- **Reţele semantice şi grafuri conceptuale - o reprezentare distribuită (concepte legate între ele prin diverse relaţii). Principalul mecanism folosit este *căutarea*.**
- **Cadre şi scripturi - o reprezentare structurată (grupuri de concepte şi relaţii); sunt foarte utile în reprezentarea tipicităţii. Principalul mecanism folosit este *împerecherea* (potrivirea) *şabloanelor* (tiparelor) – „pattern matching”.**

REŢELE BAYESIENE

- **Ideea: În lumea reală majoritatea evenimentelor sunt independente de celelalte, astfel încât interacţiunile dintre ele nu trebuie luate în consideraţie. În schimb, se poate folosi *o reprezentare locală* având rolul de a descrie grupuri de evenimente care interacţionează (dependenta locala intre variabile).**

- **Relaţiile de independenţă condiţionată dintre variabile pot reduce substanţial numărul probabilităţilor condiţionate care trebuie specificate.**

- **Structura de date folosită cel mai frecvent pentru a reprezenta dependenţa dintre variabile şi pentru a da o specificaţie concisă a repartiţiei de probabilitate multidimensionale este *reţeaua Bayesiană*.**

- **Retelele Bayesiene sunt un tip de model grafic probabilist care utilizeaza inferenta Bayesiana pentru calculul probabilitatilor.**

- **Retelele Bayesiene isi propun sa modeleze dependenta conditionata si, prin urmare, cauzalitatea, prin reprezentarea dependentei conditionate sub forma muchiilor dintr-un graf directionat. Prin aceste relatii, se poate efectua inferenta asupra variabilelor aleatoare din graf.**

- **Reteaua Bayesiana constituie un graf directionat aciclic in care fiecare muchie corespunde unei dependente conditionate si fiecare nod corespunde unei variabile aleatoare unice.**

- **Pentru retelele Bayesiene s-au creat algoritmi performanti care efectueaza**
  - ✓ **Inferenta**

  sau
  - ✓ **Invatare**

  in reteaua Bayesiana.

- **Vom studia un algoritm de inferenta in retele Bayesiene.**

Definiţia 1

O *reţea Bayesiană* este un graf cu următoarele proprietăţi:

(1) Graful este direcţionat şi aciclic.

(2) O mulţime de variabile aleatoare constituie nodurile reţelei.

(3) O mulţime de arce direcţionate conectează perechi de noduri. În mod intuitiv, semnificaţia unui arc direcţionat de la nodul *X* la nodul *Y* este aceea că *X* are *o influenţă directă* asupra lui *Y*.

(4) Fiecărui nod îi corespunde un tabel de probabilităţi condiţionate care cuantifică efectele pe care părinţii le au asupra nodului respectiv. (Părinţii unui nod sunt toate acele noduri din care pleacă arce direcţionate înspre acesta).

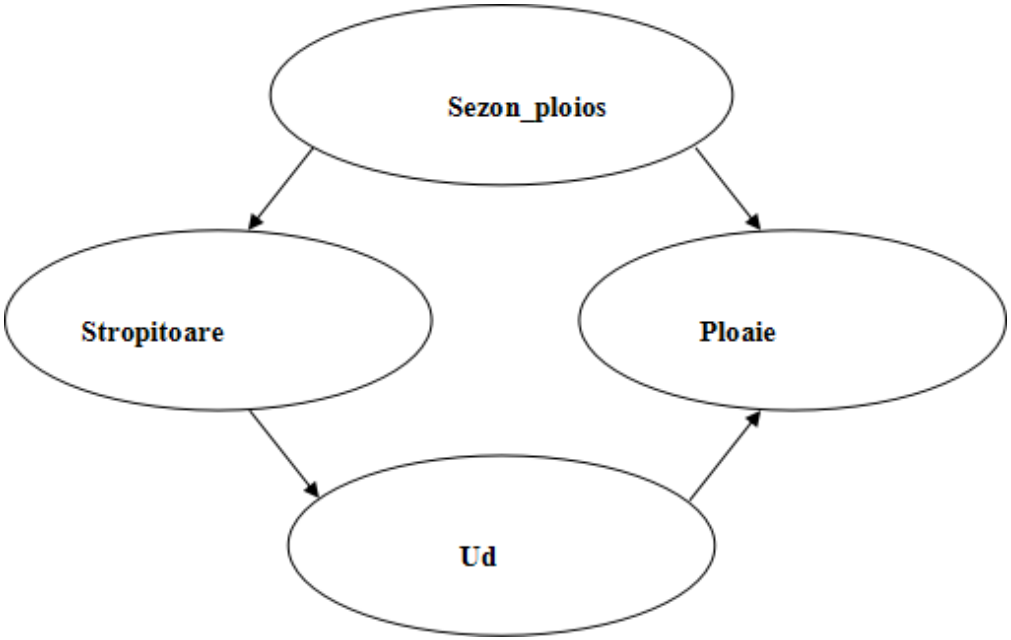
- **Un expert în domeniu decide relaţiile directe de dependenţă condiţionată care sunt valabile în domeniu.**

- **După stabilirea topologiei reţelei Bayesiene, nu mai este necesară decât specificarea probabilităţilor condiţionate ale acelor noduri care participă în dependenţe directe. Acestea vor fi folosite în calculul oricăror alte probabilităţi.**

- **Concret: se construieşte un graf direcţionat aciclic care reprezintă *relaţiile de cauzalitate* dintre variabile. Variabilele dintr-un astfel de graf pot fi propoziţionale (caz în care pot lua valorile TRUE sau FALSE) sau pot fi variabile care primesc valori de un alt tip (spre exemplu o temperatură a corpului sau măsurători făcute de către un dispozitiv de diagnosticare).**

- **Pentru a putea folosi graful cauzalităţii (reţea) ca bază a unui raţionament de tip probabilist sunt necesare însă mai multe informaţii.**

- **În particular, este necesar să cunoaştem, pentru o valoare a unui nod părinte, ce dovezi sunt furnizate referitor la valorile pe care le poate lua nodul fiu.**



Probabilităţile condiţionate corespunzătoare pot fi date sub forma unui tabel de tipul:

Atribut	Probabilitate
p(Ud|Stropitoare, Ploaie)	0.95
p(Ud|Stropitoare, ¬ Ploaie)	0.9
p(Ud| ¬ Stropitoare, Ploaie)	0.8
p(Ud| ¬ Stropitoare, ¬ Ploaie)	0.1
p(Stropitoare| Sezon ploios)	0.0
p(Stropitoare| ¬ Sezon ploios)	1.0
p(Ploaie|Sezon ploios)	0.9
p(Ploaie| ¬ Sezon ploios)	0.1
p(Sezon ploios)	0.5

Exemple:

- **probabilitatea a priori de a avea un sezon ploios este 0.5**
- **dacă suntem în timpul unui sezon ploios, probabilitatea de a avea ploaie într-o noapte dată este 0.9**

- **Pentru a putea folosi o asemenea reprezentare în rezolvarea problemelor este necesar un mecanism de calcul al influenţei *oricărui* nod arbitrar asupra *oricărui* alt nod.**

- **Pentru a obţine acest mecanism de calcul este necesar ca:**

  - **graful iniţial să fie convertit la un graf nedirecţionat, în care arcele să poată fi folosite pentru a se transmite probabilităţi în oricare dintre direcţii, în funcţie de locul din care provin dovezile;**

  - **să existe un mecanism de folosire a grafului care să garanteze transmiterea corectă a probabilităţilor (exemplu: iarba udă să nu constituie o dovadă a ploii, care să fie considerată apoi o dovadă a existenţei ierbii ude).**

- **Există trei mari clase de algoritmi care efectuează aceste calcule.**

- **Ideea care stă la baza tuturor acestor metode este aceea că nodurile au domenii limitate de influenţă.**

- **Metoda cea mai folosită este probabil cea a transmiterii mesajelor [Pearl, 1988], abordare care se bazează pe observaţia că, pentru a calcula probabilitatea unui nod *A* condiţionat de ceea ce se ştie despre celelalte noduri din reţea, este necesară cunoaşterea a trei tipuri de informaţii:**

  - **suportul total care soseşte în *A* de la nodurile sale părinte (reprezentând cauzele sale);**
  - **suportul total care soseşte în *A* de la fiii acestuia (reprezentând simptomele sale);**
  - **intrarea în matricea fixată de probabilităţi condiţionate care face legătura dintre nodul *A* şi cauzele sale.**

Relaţii de independenţă condiţionată în reţele Bayesiene

- **O reţea Bayesiană exprimă independenţa condiţionată a unui nod şi a predecesorilor săi, fiind daţi părinţii acestuia şi foloseşte această independenţă pentru a proiecta o metodă de construcţie a reţelelor.**

- **Pentru a stabili *algoritmi de inferenţă* trebuie însă să ştim dacă se verifică independenţe condiţionate mai generale, adica:**

  - ✓ **Fiind dată o reţea, dorim să putem "citi" dacă o mulţime de noduri *X* este independentă de o altă mulţime *Y* fiind dată o mulţime de "noduri dovezi" *E*.**

- **Metoda de stabilire a acestui fapt se bazează pe noţiunea de *separare dependentă de direcţie* sau *d-separare*.**

Definiţia 2

O mulţime de noduri *E* *d-separă* două mulţimi de noduri *X* şi *Y* dacă orice drum nedirecţionat de la un nod din *X* la un nod din *Y* este *blocat* condiţionat de *E*.

Definiţia 3

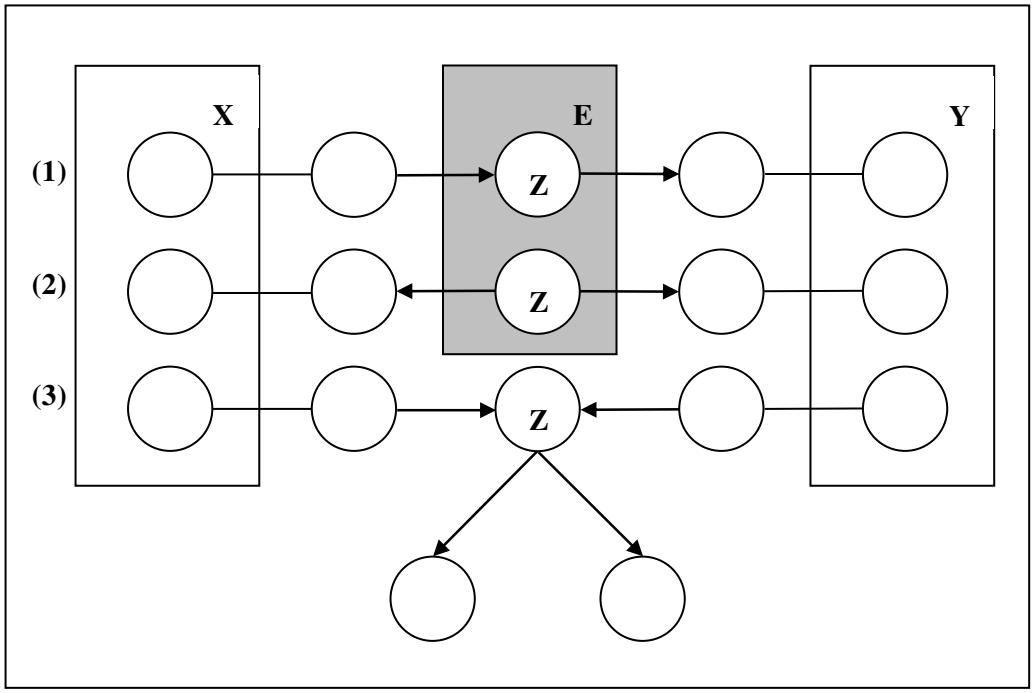
Fiind dată o mulţime de noduri *E*, spunem că un drum este *blocat* condiţionat de *E* dacă există un nod *Z* aparţinând drumului, pentru care una dintre următoarele trei condiţii se verifică:

(1) *Z* aparţine lui *E* şi *Z* are o săgeată a drumului intrând în *E* şi o săgeată a drumului ieşind din *E*.

(2) *Z* aparţine lui *E* şi *Z* are ambele săgeţi ale drumului ieşind din *E*.

(3) Nici *Z* şi nici vreunul dintre descendenţii săi nu aparţin lui *E*, iar ambele săgeţi ale drumului ţintesc înspre *Z*.

Figura următoare ilustrează cele trei cazuri posibile:



- **S-a demonstrat că (teorema fundamentală a reţelelor Bayesiene, demonstrată de Verma şi Pearl):**
  - ✓ ***dacă orice drum nedirecţionat de la un nod din X la un nod din Y este d-separat de E, atunci X şi Y sunt independente condiţionat de E.***

- **Procesul construirii şi folosirii reţelelor Bayesiene nu utilizează d-separarea.**

- **Noţiunea de d-separare este fundamentală în construcţia algoritmilor de inferenţă.**

Inferenţa în reţele Bayesiene

- **Sarcina principală a oricărui sistem probabilist de inferenţă este aceea de a calcula probabilităţi a posteriori de tipul**

  P(Interogare|Dovezi)

  corespunzător unei mulţimi de *variabile de interogare* condiţionat de valori exacte ale unor *variabile dovezi*.

  - ✓ **Exemplu: În exemplul considerat, *Ud* este o variabilă de interogare, iar *Stropitoare* şi *Sezon_ploios* ar putea fi variabile dovezi.**

- **Reţelele Bayesiene sunt suficient de flexibile pentru ca orice nod să poată servi fie ca o variabilă de interogare, fie ca o variabilă dovadă.**

- **Un agent primeşte valori ale variabilelor dovezi de la senzorii săi (sau în urma altor raţionamente) şi întreabă despre posibilele valori ale altor variabile astfel încât să poată decide ce acţiune trebuie întreprinsă.**

- **Sarcina cea mai frecventă şi mai utilă a reţelelor Bayesiene:**
  - ✓ **determinarea probabilităţilor condiţionate a posteriori ale variabilelor de interogare.**

Literatura de specialitate discută patru tipuri distincte de inferenţă, care poate fi realizată de reţelele Bayesiene:

- **inferenţă de tip diagnostic (de la efecte la cauze);**
- **inferenţă cauzală (de la cauze la efecte);**
- **inferenţă intercauzală (între cauze ale unui efect comun);**
- **inferenţe mixte (reprezentând combinaţii a două sau mai multe dintre inferenţele anterioare).**

***Exemplu*: Setând efectul *Stropitoare* la valoarea "adevărat" şi cauza *Sezon_ploios* la valoarea "fals", ne propunem să calculăm**

$$P(Ud|Stropitoare, \neg Sezon\_ploios).$$

Aceasta este o inferenţă mixtă, care reprezintă o utilizare simultană a inferenţei de tip diagnostic şi a celei cauzale.

Un algoritm pentru răspunsul la interogări
(algoritm de calcul al probabilităţilor condiţionate a posteriori ale variabilelor de interogare)

- **Algoritmul va fi de tip *înlănţuire înapoi* prin faptul că pleacă de la variabila de interogare şi urmează drumurile de la acel nod până la nodurile dovezi.**

- **Algoritmul se referă numai la *reţele unic conectate*, numite şi poli-arbori.**
  (In poli-arbori există cel mult un drum nedirecţionat între oricare două noduri ale reţelei).

- **Algoritmii pentru reţele generale vor folosi algoritmii referitori la poli-arbori ca principală subrutină.**

- **Figura care urmează prezintă o <u>reţea generică unic conectată</u>.**

- **În această reţea nodul *X* are <u>părinţii</u> $U = U_1 \dots U_m$ şi <u>fiii</u> $Y = Y_1 \dots Y_n$. Corespunzător fiecărui fiu şi fiecărui părinte a fost desenat un dreptunghi care include toţi <u>descendenţii</u> nodului şi toţi <u>strămoşii</u> lui (cu excepţia lui *X*).**

- **<u>Proprietatea de *unică conectare*</u> înseamnă că toate dreptunghiurile sunt disjuncte şi că nu există legături care să le conecteze între ele.**

- **Se presupune că *X* este <u>variabila de interogare</u> şi că există o mulţime *E* de <u>variabile dovezi</u>.**

- **Se urmăreşte calcularea probabilităţii condiţionate**

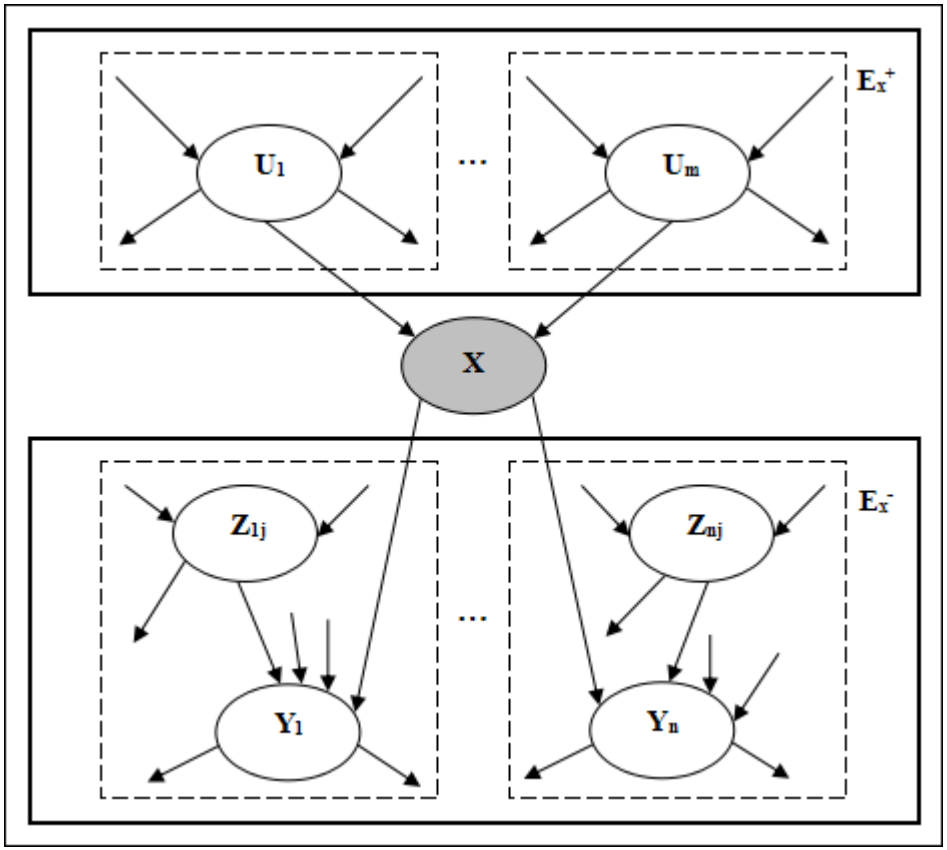
$$P(X \mid E)$$

- **Dacă însuşi *X* este o variabilă dovadă din *E*, atunci calcularea lui $P(X \mid E)$ este banală. Presupunem că *X* nu aparţine lui *E*.**

- **Reţeaua din figură este partiţionată în conformitate cu părinţii şi cu fiii variabilei de interogare *X*.**

- **Pentru a concepe un algoritm, va fi util să ne putem referi la diferite porţiuni ale dovezilor:**

  - E^+_X **reprezintă suportul cauzal pentru *X* - variabilele dovezi aflate "deasupra" lui *X*, care sunt conectate la *X* prin intermediul părinţilor săi.**
  - E^-_X **reprezintă suportul probatoriu pentru *X* - variabilele dovezi aflate "dedesubtul" lui *X* şi care sunt conectate la *X* prin intermediul fiilor săi.**



Fie reţeaua generică unic conectată din figura anterioară, partiţionată în conformitate cu părinţii şi cu fiii variabilei de interogare X. Nodul X are părinţii $\mathbf{U} = U_1 \dots U_m$ şi fiii $Y = Y_1 \dots Y_n$. Presupunem ca există o mulţime E de variabile dovezi, $E = \{E^-_X, E^+_X\} = \{U_1, \dots, U_m, Y_1, \dots, Y_n\}$, mulţime de variabile aleatoare discrete, unde:

- E^+_X **reprezintă suportul cauzal pentru X - variabilele dovezi aflate "deasupra" lui X, care sunt conectate la X prin intermediul părinţilor săi.**
- E^-_X **reprezintă suportul probatoriu pentru X - variabilele dovezi aflate "dedesubtul" lui X şi care sunt conectate la X prin intermediul fiilor săi.**

Se urmăreşte calcularea probabilităţii condiţionate $P(X \mid E)$.

Strategia generală pentru calculul lui $P(X \mid E)$ este următoarea:

- **Exprimă $P(X \mid E)$ în termenii contribuţiilor lui E^+_X şi E^-_X.**
- **Calculează contribuţia mulţimii E^+_X calculând efectul ei asupra părinţilor lui X şi apoi transmiţând acest efect lui X.**
- **Calculează contribuţia mulţimii E^-_X calculând efectul ei asupra fiilor lui X şi apoi transmiţând acest efect lui X.**

Exprimarea lui $P(X \mid E)$ în termenii contribuţiilor lui E^+_X şi E^-_X este următoarea:

$$P(X \mid E) = \alpha\, P(X \mid E^+_X)\, P(E^-_X \mid X) \quad \textbf{(1)}$$

În urma efectuării calculelor, se obţine contribuţia mulţimii $E^+{}_X$ ca fiind dată de

$$P(X \mid E^+{}_X) =$$
$$\sum_{(u_1,\ldots,u_m)} P(X|\{U_1 = u_1, \ldots, U_m = u_m\}) \prod_{i=1}^{m} P\left(\{U_i = u_i\}|E_{U_i \setminus X}\right) \textbf{(2)}$$

si, introducând (2) în (1), avem:

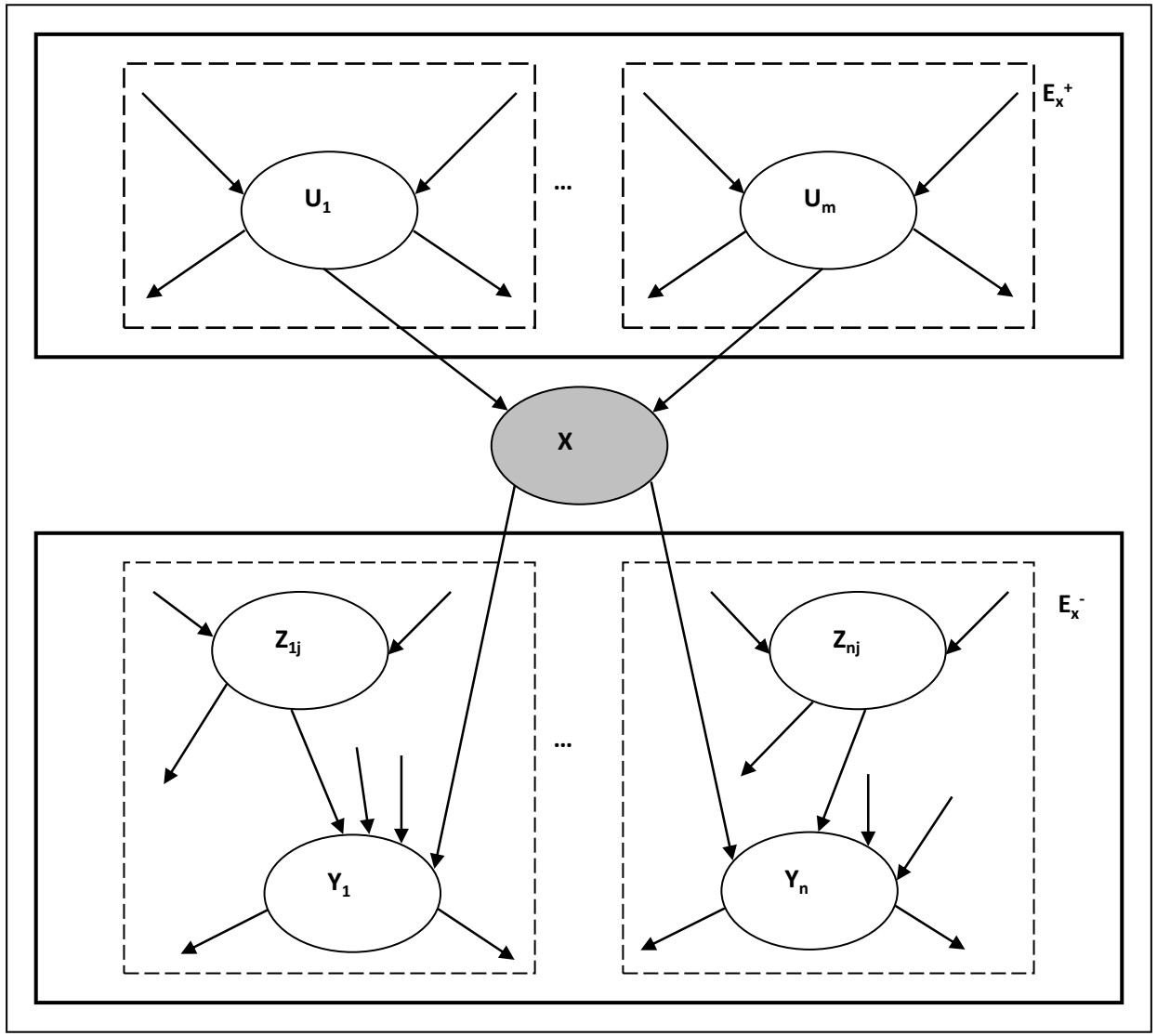
$$P(X \mid E) = \alpha\, P(E^-{}_X \mid X) \sum_{\mathbf{u}} P(X|\mathbf{u}) \prod_{i=1}^{m} P\left(\{U_i = u_i\}|E_{U_i \setminus X}\right)$$

Contribuţia mulţimii E^-_X se obţine ca fiind:

$$P(E^-_X|X) = \beta \prod_i \sum_{y_i} P(E^-_{Y_i}|y_i) \sum_{\mathbf{z}_i} P(y_i|X, \mathbf{z}_i) \prod_j P\left(z_{ij}|E_{Z_{ij}\setminus Y_i}\right)$$

Menţionăm că:

- **Notaţia $E_{U_i\setminus X}$ este folosită pentru a se face referire la toate dovezile conectate cu nodul U_i, mai puţin cele prin drumul de la X;**
- **U este vectorul părinţilor $U_1 \ldots U_m$ şi u reprezintă o atribuire de valori pentru aceştia;**
- **Y este vectorul fiilor $Y_1, \ldots, Y_n$ şi fie $\mathbf{y} = (y_1, \ldots, y_n)$ o realizare a acestuia;**
- **$\mathbf{Z_i}$ sunt părinţii lui Y_i, alţii decât X şi fie $\mathbf{z_i}$ o atribuire de valori ale acestor părinţi (vezi figura următoare).**


E_x^+
U_1
...
U_m
X
E_x^-
Z_{1j}
...
Z_{nj}
Y_1
Y_n

Folosind aceste notaţii, scriem, în pseudocod, algoritmul pentru răspunsul la interogări care calculează probabilitatea condiţionată a posteriori a variabilei de interogare X, adică $P(X \mid E)$.

Algoritmul

function INTEROGARE-REŢEA (X) **return**
o distribuţie de probabilitate a valorilor lui X
input: X, o variabilă aleatoare
SUPORT-EXCEPT (X, nul)
function SUPORT-EXCEPT (X,V) **return** $P(X|E_{X\setminus \mathrm{V}})$
if DOVEZI ? (X) **then return** distribuţia observată pentru X
else
calculează $P\left(E^{-}{}_{\boldsymbol{X}\setminus \mathrm{V}}|X\right)$ = DOVEZI-EXCEPT (X,V)
$\mathbf{U} \leftarrow$ PĂRINŢI $[X]$
if $\mathbf{U}$ este vid
then return $\alpha P\,(E^{-}{}_{X\setminus \mathrm{V}} \mid X)\, P(X)$
else
pentru fiecare $U_{\boldsymbol{i}}$ **in** U **do**
calculează şi memorează $P\left(U_i|E_{U_i\setminus X}\right) =$
= SUPORT-EXCEPT $(U_{\boldsymbol{i}}, X)$
return $\alpha P\,(E^{-}{}_{X\setminus \mathrm{V}} \mid X) \sum_{\mathbf{u}} P(X|\mathbf{u}) \prod_{\boldsymbol{i}} P\left(U_{\boldsymbol{i}}|E_{U_i\setminus X}\right)$
function DOVEZI-EXCEPT (X,V) **return** $P\,(E^{-}{}_{X\setminus \mathrm{V}} \mid X)$
$\mathbf{Y} \leftarrow$ FII $[X]$-V
if $\mathbf{Y}$ este vid
then return o repartiţie uniformă
else
pentru fiecare $Y_{\boldsymbol{i}}$ **in** $\mathbf{Y}$ **do**
calculează $P\,(E^{-}{}_{Y_{\boldsymbol{i}}} \mid y_{\boldsymbol{i}})$ = DOVEZI-EXCEPT $(Y_{\boldsymbol{i}}, \text{nul})$
$\mathbf{Z}_{\boldsymbol{i}} \leftarrow$ PĂRINŢI $[Y_{\boldsymbol{i}}]$ - X
pentru fiecare $Z_{\boldsymbol{ij}}$ **in** $\mathbf{Z}_{\boldsymbol{i}}$
calculează $P\left(Z_{\boldsymbol{ij}}|E_{Z_{ij}\setminus Y_i}\right) =$
= SUPORT-EXCEPT $\left(Z_{\boldsymbol{ij}}, Y_{\boldsymbol{i}}\right)$
return $\beta \prod_i \sum_{y_i} P\left(E^{-}{}_{Y_i}|y_i\right) \sum_{\mathbf{z}_i} P(y_i|X,\mathbf{z}_i) \prod_j P\left(z_{ij}|E_{Z_{ij}\setminus Y_i}\right)$

- **Problema:**

  Fiind dat sirul de intrare
  In aceasta familie masa se ia la ore fixe.
  sa se conceapa un soft care sa inteleaga sensul sirului de intrare.

- **Indicatie:**
  Pentru a se intelege sensul sirului de intrare, softul trebuie, mai intai, sa determine cu ce sens este folosit aici cuvantul polisemantic *masa*. (Trebuie sa se determine faptul ca *masa* este folosit cu sensul de mancare si nu cele de mobila, multime – masa de oameni, masa din fizica etc.)

- **Terminologie:**
  ***Masa* este un cuvant polisemantic (cu mai multe sensuri), deci ambiguu. Precizarea sensului cu care *masa* este folosit in sirul de intrare dat reprezinta operatia de dezambiguizare a sensului.**
- **WSD ≡ Word Sense Disambiguation (dezambiguizare automata a sensului cuvintelor polisemantice)**

O retea semantica pentru reprezentarea si procesarea limbajului natural
WordNet (WN)

- **WordNet (WN) este o baza de date lexicala a limbii engleze, conceputa si creata de Profesorul George Miller, la Universitatea Princeton, in anii '90. Ea este permanent actualizata de catre un colectiv de cercetatori.**

- **WN cuprinde toate "cuvintele cu continut" ale limbii engleze, adica toate substantivele, adjectivele, verbele si adverbele. Substantivele si verbele sunt organizate in ierarhii, iar adjectivele si adverbele in clustere.**

- **"Piatra de temelie" in WN o constituie *synset*-ul, sau multimea de sinonime.**

- **Aici cuvinte sinonime inseamna cuvinte care se refera la un acelasi *concept*. (De pilda, cuvintele *scaun* si *masa* sunt considerate aici sinonime, intrucat ambele se refera la conceptul de "mobila".)**

- **Fiecare *synset* din WN se refera la un anumit *concept*.**

- **Toate sinonimele care apar intr-un synset au aceeasi parte de vorbire (substantiv, adjectiv, verb sau adverb).**

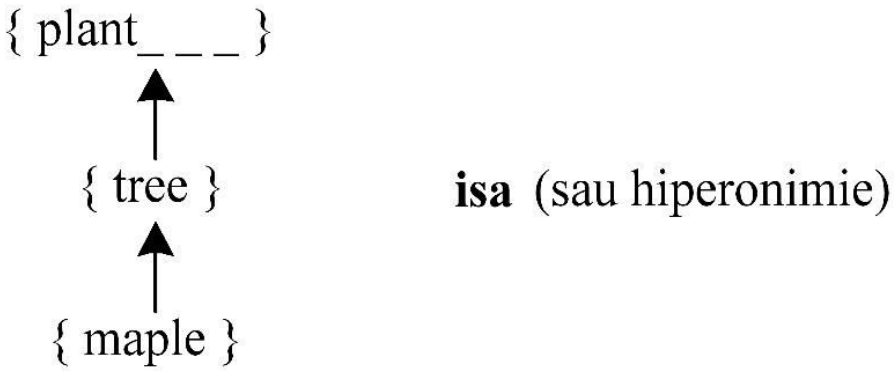
- **Un cuvant polisemantic (cu mai multe sensuri) se refera la un alt concept prin fiecare dintre sensurile lui. Prin urmare, un cuvant polisemantic va interveni in mai multe synset-uri WN. El apartine cate unui synset prin fiecare sens al lui. Spre exemplu, cuvantul polisemantic *masa* poate apartine fiecaruia dintre synset-urile care se refera la conceptele de *mancare, mobila, multime, masa din fizica* etc.**

- **Un synset WN contine toate sinonimele care lexicalizeaza conceptul la care synset-ul se refera si care au aceeasi parte de vorbire (substantiv, verb etc.), precum si un string numit *glosa*, care seamana cu o definitie clasica a sensului (cum sunt cele din dictionar). Glosa poate contine si un exemplu de utilizare a sinonimelor din synset.**

- **Synset-urile WN sunt legate intre ele prin *relatii semantice.* Aceste relatii leaga intre ele conceptele la care se refera synset-urile si care sunt aceleasi in toate limbile (conceptele sunt independente de limba).**

Synset-uri de substantive

Synset-urile substantivale sunt legate intre ele prin relatii semantice cum ar fi *hiperonimia* si inversa acestei relatii, *hiponimia*, care reprezinta relatia ISA din inteligenta artificiala in domeniul procesarii limbajului natural. Pe baza acestei relatii este construita ierarhia substantivala din WN:



Hiperonimul este conceptul parinte, iar hiponimul este conceptul fiu. Mecanismul de inferenta este mostenirea proprietatilor. (Hiponimele mostenesc toate proprietatile hiperonimului).

Synset-uri substantivale – continuare:

- **alta relatie semantica importanta in cazul substantivelor este *meronimia* sau relatia *parte-din* (“the *part-of* relation”). Inversa acestei relatii este *holonimia*.**

Exemplu: substantivele *clanta* si *usa*.

(Clanta este parte din usa. Substantivul *usa* este holonim al substantivului *clanta*. Iar *clanta* este meronim al lui *usa*).

WordNet

- **Structura de retea semantica a WN este data de relatiile semantice care leaga synset-urile intre ele.**

- **Prin intermediul relatiei de hiperonimie conceptele mostenesc toate proprietatile conceptelor parinte. Mostenirea proprietatilor transforma WN intr-o baza de cunostinte.**

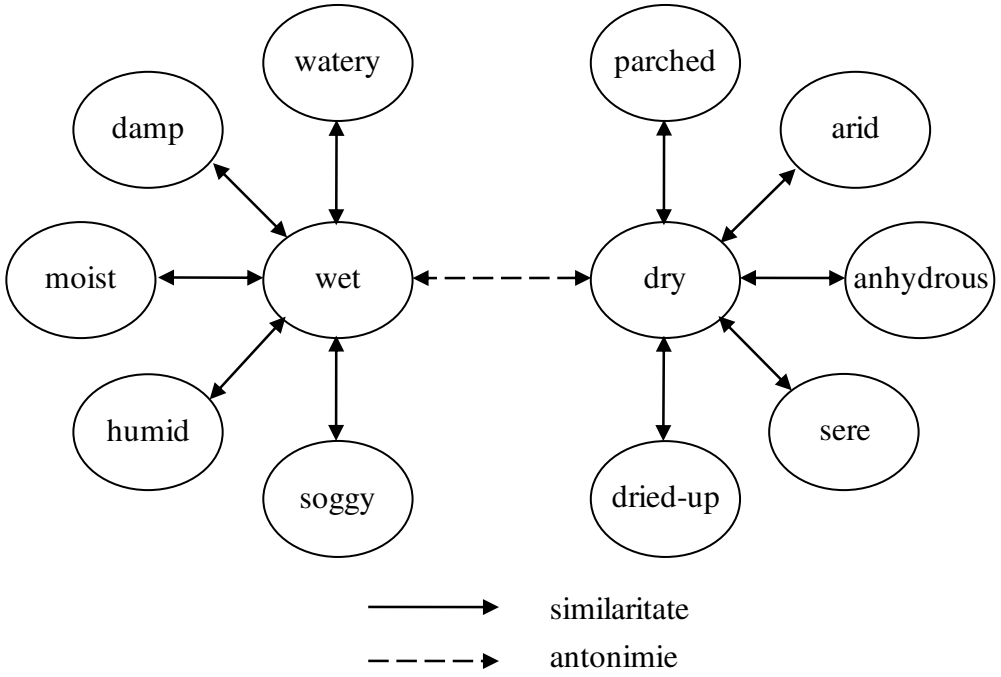
- **Concluzie: WN este o baza de date lexicala a limbii engleze, o retea semantica si o baza de cunostinte. Faptul ca WN reprezinta o baza de cunostinte o face extrem de utila in diverse aplicatii din IA.**

SYNSET-uri de adjective in WN

Relatii semantice de o alta natura leaga intre ele synset-urile de adjective din WN. Principala relatie semantica este considerata a fi *antonimia*.

Prin intermediul relatiei de antonimie synset-urile de adjective sunt organizate in *cluster*-e. Mai jos este dat ca exemplu cluster-ul format in jurul antonimelor

***wet / dry*:**



Synset-uri de adjective

Relatia de *similaritate* exista in WN numai intre synset-uri de adjective. Ea aduce, in plus, antonime indirecte. (In exemplul dat, *wet* si *dry* sunt *antonime directe*, iar *wet* si *arid* sunt *antonime indirecte*. Adjectivul *moist* nu are un antonim direct, dar un antonim indirect al lui poate fi gasit urmand drumul *moist* $\rightarrow$ *wet* $\rightarrow$ *dry*).

S-a dovedit ca relatia de *antonimie* furnizeaza informatie importanta procesului de dezambiguizare automata a sensului.

Alte relatii semantice importante in cazul adjectivelor sunt: *also-see*, *pertaining-to*, *attribute*.

SYNSET-URI DE VERBE SI ADVERBE

- **Synset-urile de verbe sunt organizate in ierarhii, ca si cele de substantive, prin intermediul relatiei *entailment* (to entail = a atrage dupa sine).**

- **Alta relatie semantica importanta pentru verbe este relatia *cauzala.***

- **Synset-urile de adverbe sunt organizate in cluster-e, ca si cele de adjective.**

Similaritate si inrudire
(intre concepte si cuvintele care le lexicalizeaza)

- **Inrudirea semantica este un concept mai general decat similaritatea semantica. Similaritatea este un caz particular de inrudire.**
  Exemplu: *Masina* si *anvelopa* nu sunt similare, dar sunt inrudite (anvelopa este o "parte din" masina). *Doctorii* si *spitalele* nu sunt concepte similare, dar sunt inrudite.

- **In literatura exista diverse masuri pentru calcularea similaritatii si/sau inrudirii. Vom vedea o asemenea masura in algoritmul care urmeaza.**

- **S-a demonstrat ca, pentru a realiza dezambiguizarea automata a sensului cuvintelor, conceptul de inrudire este mai util decat cel de similaritate.**

- **Studiem, in continuare, un algoritm de dezambiguizare a sensului cuvintelor bazat pe conceptul de inrudire si care foloseste cunostinte furnizate de reteaua semantica WordNet.**

- **Problema:**
  Fiind dat sirul de intrare
  In aceasta familie masa se ia la ore fixe.
  sa se conceapa un soft care sa inteleaga sensul sirului de intrare.

- **Indicatie:**
  Pentru a se intelege sensul sirului de intrare, softul trebuie, mai intai, sa determine cu ce sens este folosit aici cuvantul polisemantic *masa*. (Trebuie sa se determine faptul ca *masa* este folosit cu sensul de mancare si nu cele de mobila, multime – masa de oameni, masa din fizica etc.) Algoritmul care urmeaza dezambiguizeaza cuvantul ambiguu *masa*.
  Cuvantul de dezambiguizat ≡ cuvant tinta

Algoritmul Lesk extins

- **Banerjee si Pedersen (2003) prezinta o noua masura a inrudirii semantice intre concepte, care se bazeaza pe numarul de cuvinte pe care le au in comun definitiile lor (suprapuneri de glose – "gloss overlaps").**

- **Aceasta masura primeste ca input doua concepte (reprezentate de doua synset-uri WN) si ofera ca output o valoare numerica, care cuantifica gradul lor de inrudire semantica. Aceasta valoare numerica este apoi folosita pentru a realiza dezambiguizarea sensului.**

- **Masura de inrudire si algoritmul de dezambiguizare corespunzator reprezinta o varianta a Algoritmului lui Lesk clasic (care se bazeaza pe suprapuneri de glose). Aceasta varianta extinde glosele conceptelor luate in consideratie.**

- **Extinderea gloselor se face astfel incat sa fie luate in consideratie si glosele altor concepte, inrudite cu cele date prin intermediul relatiilor semantice furnizate de WN.**

Algoritmul lui Lesk

Suprapunerile de glose (definitii) au fost introduse de Lesk (1986) cu scopul de a servi in dezambiguizarea sensului.

Algoritmul lui Lesk atribuie un sens unui cuvant tinta (de dezambiguizat), intr-un context dat, prin compararea gloselor diverselor sensuri ale cuvantului cu cele ale celorlalte cuvinte din context. Sensul cuvantului tinta a carui glosa are cele mai multe cuvinte in comun cu glosele cuvintelor invecinate este cel atribuit cuvantului tinta.

Exemplu: consideram urmatoarele glose ale cuvintelor *car* si *tire*

car*: *four wheel motor vehicle usually propelled by an internal combustion engine

tire*: *hoop that covers a wheel, usually made of rubber and filled with compressed air

- ✓ **glosele acestor concepte au in comun cuvantul *wheel***

Limitare: Algoritmul Lesk original ia in consideratie numai suprapunerile dintre glosele cuvantului tinta si cele ale cuvintelor din jurul lui, in contextul dat. Definitiile de dictionar sunt insa scurte si nu furnizeaza un vocabular suficient de mare (vor fi putine cuvinte comune).

- **Masura suprapunerii extinse de glose introdusa de Banerjee si Pedersen (2003) extinde glosele conceptelor luate in consideratie prin includerea gloselor conceptelor inrudite cu acestea. Conceptele inrudite sunt alese prin intermediul relatiilor semantice din WN.**

Masura suprapunerii extinse de glose

- **Atunci cand masuram inrudirea dintre doua synset-uri de input, cautam suprapuneri nu numai intre glosele lor, dar si intre glosele synset-urilor hiperonime, hiponime, meronime, holonime etc. ale lor.**

- **Alte relatii WN de luat in consideratie sunt attribute, similar-to, also-see, antonymy etc.**

- **Observatie: NU toate aceste relatii au aceeasi importanta in procesul de dezambiguizare. Alegerea relatiilor depinde de partea de vorbire a cuvantului tinta (substantiv, adjectiv, verb, adverb), dar si de tipul de aplicatie pentru care se studiaza inrudirea.**

- **Vom aplica aici aceasta masura a inrudirii in procesul de dezambiguizare.**

Exemplu de utilizare a relatiilor semantice din WN:

In cazul substantivelor testele empirice au demonstrat ca este util in dezambiguizare sa folosim glosele hiponimelor si meronimelor synset-urilor de input. Observati ca se recomanda folosirea hiponimelor (conceptul fiu) si nu a hiperonimelor (conceptul parinte) sau a ambelor. Cu alte cuvinte, s-a demonstrat ca hiponimia furnizeaza mai multa informatie decat hiperonimia, desi ambele formeaza relatia ISA.

Mecanism de acordare a scorurilor

- **Algoritmul Lesk iniţial compară glosele unei perechi de concepte şi calculează un scor prin numărarea cuvintelor pe care acestea le au în comun. Acest mecanism de acordare a scorurilor nu diferenţiază între suprapuneri de cuvinte unice şi suprapuneri de grupuri de cuvinte, ci tratează fiecare glosă ca pe un „sac de cuvinte". Spre exemplu, acorda scorul 3 conceptelor *drawing paper* si *decal*, care au urmatoarele glose:**

  drawing paper: paper that is specially prepared for use in drafting
  decal: the art of transferring designs from specially prepared paper to a wood or glass or metal surface.

  Aici exista 3 suprapuneri, cuvantul *paper* si grupul de cuvinte *specially prepared*.

- **Banerjee si Pedersen (2003) atribuie unei suprapuneri de *n* cuvinte scorul n^2 (pentru ca o suprapunere de *n* cuvinte consecutive este mult mai rara decat suprapunerile de cuvinte unice). Pentru perechea anterioara de glose se atribuie scorul 5 (in loc de 3).**

Algoritm de acordare a scorurilor

1. **Fiind date două şiruri, se detectează cea mai lungă suprapunere dintre acestea. Dacă două sau mai multe astfel de suprapuneri au aceeaşi lungime maximă, atunci este raportată acea suprapunere care intervine prima în primul şir care se compară. Cea mai lungă suprapunere detectată este înlăturată, iar în locul ei se plasează un marcaj în fiecare dintre cele două şiruri constituind input-ul. Cele două şiruri astfel obţinute sunt apoi din nou verificate pentru suprapuneri, iar acest proces continuă până când nu mai există suprapuneri între ele.**

2. **Se atribuie scoruri tuturor suprapunerilor găsite (dimensiunile suprapunerilor detectate sunt ridicate la pătrat) şi aceste scoruri sunt adunate pentru a se determina *scorul perechii de glose date*.**

Calcularea scorului de inrudire dintre doua synset-uri A si B

1. **Definim mulţimea RELS ca pe o mulţime nevidă de relaţii constând din una sau mai multe dintre relaţiile puse la dispoziţie de WordNet:**

$$RELS \subset \{r \mid r \text{ este o relaţie definită în WordNet}\}.$$

Presupunem că funcţia fiecărei relaţii *r* (*r* ∈ RELS) este cea dată de numele relaţiei, care acceptă un synset de input şi întoarce glosa synset-ului (sau synset-urilor) înrudite cu cel de input prin relaţia desemnată. Spre exemplu, dacă *r* ∈ RELS reprezintă relaţia de hiperonimie, atunci *r*(*A*) întoarce glosa unui synset hiperonim al lui *A*. *r* mai poate reprezenta „relaţia glosă”, caz în care *r*(*A*) întoarce glosa synset-ului A.

OBS.: Dacă mai multe synset-uri sunt înrudite cu cel de input prin intermediul aceleiaşi relaţii, glosele acestora vor fi concatenate şi returnate ca un unic şir de caractere. Această concatenare este efectuată deoarece nu se doreşte să se facă o diferenţiere între synset-uri care sunt înrudite cu cel de input printr-o aceeaşi relaţie, de interes fiind doar glosele acestora. Atunci când niciun synset nu este înrudit cu cel de input prin relaţia dată, va fi returnat şirul vid.

Definim o nouă mulţime nevidă, de *perechi de relaţii*, selectate din mulţimea RELS. Singura constrângere în formarea acestor perechi de relaţii va fi aceea că, dacă este aleasă perechea (r_1, r_2), atunci va trebui aleasă şi perechea (r_2, r_1), astfel încât să se asigure reflexivitatea măsurii de înrudire (r_1, $r_2 \in$ RELS). Cu alte cuvinte, trebuie să se verifice *înrudire* (A, B) = *înrudire* (B, A). Vom numi această nouă mulţime RELPAIRS. Ea se defineşte după cum urmează:

2. RELPAIRS = {(R_1, R_2) $\mid R_1, R_2 \in$ RELS; dacă (R_1, R_2) $\in$ RELPAIRS, atunci (R_2, R_1) $\in$ RELPAIRS}.

3. Presupunem că *scor ( )* este o funcţie care acceptă ca input două glose, găseşte grupurile de cuvinte care se suprapun ale acestora şi întoarce un scor calculat conform Algoritmului de acordare a scorurilor descris anterior.

- **Folosind toate aceste elemente, calculam scorul de inrudire dintre synset-urile A si B dupa cum urmeaza:**

$$\textit{înrudire}\ (A, B) = \sum_{\forall (R_1, R_2) \in \mathrm{RELPAIRS}} \textit{scor}\ (R_1(A), R_2(B)).$$

Această măsură de înrudire se bazează pe mulţimea tuturor perechilor posibile de relaţii furnizate de reţeaua semantică WordNet. În practică, vor fi alese acele relaţii care s-au dovedit a fi cele mai utile corespunzător fiecărei părţi de vorbire.

<u>Exemplu:</u>

Vom presupune că mulţimea de relaţii aleasă este RELS = {gloss, hype, hypo} (unde *hype* şi *hypo* reprezintă relaţiile de hiperonimie şi, respectiv, hiponimie). În continuare, vom presupune că mulţimea perechilor de relaţii RELPAIRS este dată de RELPAIRS = {(gloss, gloss), (hype, hype), (hypo, hypo), (hype, gloss), (gloss, hype)}. Atunci, înrudirea dintre synset-urile *A* şi *B* poate fi calculată după cum urmează:

$$\textit{înrudire}\ (A,B) = \textit{scor}\ (\textit{gloss}(A), \textit{gloss}(B)) + \textit{scor}\ (\textit{hype}(A), \textit{hype}(B)) + \textit{scor}\ (\textit{hypo}(A), \textit{hypo}(B)) + \textit{scor}\ (\textit{hype}(A), \textit{gloss}(B)) + \textit{scor}\ (\textit{gloss}(A), \textit{hype}(B)).$$

Se observă că alegerea acestor perechi de relaţii asigură egalitatea *înrudire* (*A*,*B*) = *înrudire* (*B*,*A*).

Aplicatie in dezambiguizarea sensului cuvintelor

Vom selecta o fereastră de context în jurul cuvântului ţintă (de dezambiguizat) şi vom presupune că această fereastră de context constă din $2n + 1$ cuvinte notate w_i, $-n \leq i \leq +n$, unde cuvântul ţintă este w_0. Pentru fiecare cuvânt cu conţinut din fereastra de context este identificată o mulţime de sensuri candidate. Fie $|w_i|$ numărul de sensuri candidate ale cuvântului w_i. Aceste sensuri vor fi notate prin $s_{i,j}$, $1 \leq j \leq |w_i|$. Fiecărui sens k posibil al cuvântului ţintă îi vom atribui un scor notat $SenseScore_k$ şi calculat prin adunarea scorurilor de înrudire obţinute la compararea sensului analizat al cuvântului ţintă cu fiecare sens al fiecărui cuvânt cu conţinut care nu este cuvânt ţintă din fereastra de context. Scorul sensului $s_{0,k}$ este calculat după cum urmează:

$$SenseScore_k = \sum_{i=-n}^{n} \sum_{j=1}^{|w_i|} \hat{i}nrudire(s_{0,k}, s_{i,j}), \quad i \neq 0$$

Acel sens care are scorul cel mai ridicat este considerat a fi cel mai adecvat pentru cuvântul ţintă. Dacă mai multe sensuri au scor egal, atunci, prin convenţie, se alege acel sens care este considerat cel mai important (uzual) conform WordNet. O combinaţie candidată de glose care nu prezintă nicio suprapunere va primi scorul zero.

Complexitate

Dacă, în medie, există α sensuri pentru fiecare cuvânt, iar fereastra de context conţine N cuvinte, vom avea $\alpha^2 \times (N - 1)$ perechi de mulţimi de synset-uri care trebuie comparate, ceea ce reprezintă o creştere liniară în raport cu N.