UMJETNA INTELIGENCIJA

„AI-complete" problemi:
- ↳ problemi lagani za čovjeka, a teški za računalo
- ↳ računalni problemi čija je složenost istovjetna rješavanju **temeljnog problema umjetne inteligencije:**
  - izgraditi računalo inteligentno kao i čovjek

⇒ **INTELIGENCIJA:**
- ↳ ne postoj usklađena definicija
- ↳ postoj nekolika pojmova, al najčešće se spominju **samosvijest** i **prilagodavanje neformatoj situaciji**

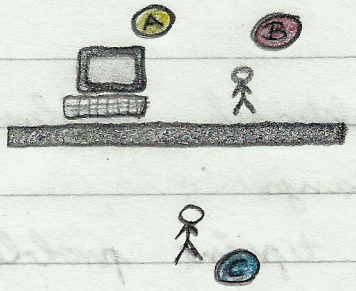
⇒ inženjerski pristup inteligenciji:
UNDERSTANDING-BY-BUILDING

⇒ **UMJETNA INTELIGENCIJA:**
- ↳ grana računarske znanosti
- ↳ sastoj se od mnogo podgrana
- ↳ znanstvena disciplina koja se bavi izgradnjom računala čije je ponašanje moguće promatrati kao inteligentno

⇒ TURINGOV TEST:

↳ igra „Pogodi osobu"

↳ cilj igrača „C" je da pogodi tko od igrača „A" , „B" je računala, a tko je čovjek



↳ pretpostavka da se ovim testom ispituje da li se stroj ponaša ljudski

↳ neke od sposobnosti da stroj prođe Turingov test je:

- razumijevanje jezika
- dosljednost
- kreativno razmišljanje
- pojam laži i istine

⇒ postoje četiri različite dimenzije definicije inteligencije:

a) RAZMIŠLJATI LJUDSKI

b) RAZMIŠLJATI RACIONALNO

c) PONAŠATI SE LJUDSKI

d) PONAŠATI SE RACIONALNO

⇒ pretpostavka da Turingov test ispituje samo ljudsko ponašanje, pa se danas ne pretpostavlja za dokaz inteligencije

PRETRAŽIVANJE PROSTORA STANJA

⇒ mnoga analitčkih problema se može riješit pretraživanjem prostora stanja

↳ neki tipični problemi:

a) rješavanje sudoku tablice

b) nalaženje puta od A do B (GPS)

c) rubrickova kocka

⇒ **FORMALAN OPIS:**

↳ neka je S skup stanja

↳ problem se sastoj od početnog stanja, prijelaza izmedu stanja i ciljnih stanja

$$(s_0, succ, goal)$$

1) $s_0 \in S$
2) $succ : S \to \wp(S)$ ⇒ funkcija sljedbenika
3) $goal : S \to \{T, \bot\}$ ⇒ ispitni predikat

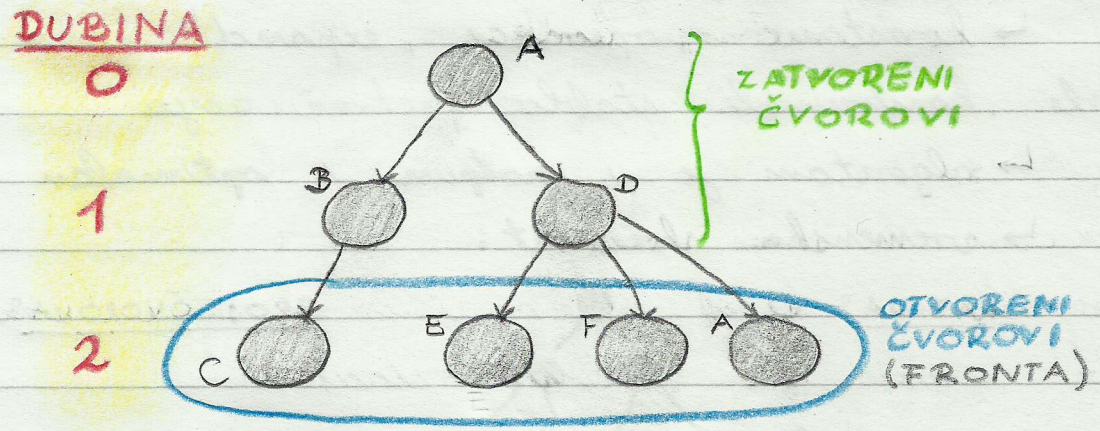
⇒ ispitni predikat daje istinu „T" samo za ciljna stanja, a neistinu za ostala stanja //

⇒ primjetimo da ovakvim prikazom svodimo problem na problem USMJERENOG TEŽINSKOG GRAFA, no to ne moramo koristiti uvijek

⇒ STABLO PRETRAŽIVANJA:

↳ ra razliku od grafa, ovdje možemo imati ponavljanje čvorova, ali ne možemo imati ciklus u grafu

↳ najprije prikazuje put gdanja kroz graf



↳ primjetimo da ukoliko graf sadrži ciklus, stablo će biti beskonačno

↳ u stablu govorimo o čvorovima, a u grafu o stanjima

↳ čvor pohranjuje stanje

⇒ SVOJSTVA ALGORITAMA:

a) POTPUNOST - uvijek nalazi rješenje

b) OPTIMALNOST - uvijek nalazi najbolje rješenje

c) VREMENSKA SLOŽENOST

d) PROSTORNA SLOŽENOST

⇒ STRATEGIJE PRETRAŽIVANJA:

a) slijepo pretraživanje

b) usmjereno pretraživanje

⇒ sada ćemo se bavit sa slijepim pretraživanjem i pogledat jednostavne algoritme slijepog pretraživanja

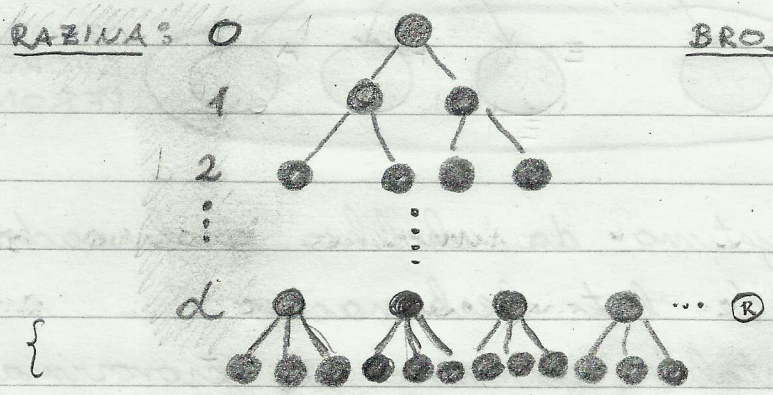
⇒ SLIJEPO PRETRAŽIVANJE:

1) pretraživanje u širinu

↳ koristimo „open" i „expand" kako bi konstruirali stablo pretraživanja

↳ algoritam je i potpun i optimalan

↳ vremenska složenost:



$b^{d+1} - b$ ekspanzija

b ⇒ „branching factor"

d ⇒ „depth of optimal solution"

↳ ekspandirali smo b^{d+1} čvorova u najgorem slučaju pa je to vremenska i prostorna složenost ovog algoritma: $O(b^{d+1})$

↳ algoritam prihvatljiv samo za male probleme

2) pretraživanje s jednolikom cijenom

↳ ako nisu sve akcije jednako skupe (npr. duljina ceste između mjesta) koristimo ovaj algoritam (ima nezgodno ime ☹)

↳ ideja je prvo ekspandirati jeftinije čvorove

↳ to je realizirano tako da se na glavu liste uvijek stavlja čvor sa najmanjom cijenom, tj. kod umetanja u listu moramo paziti!

↳ algoritam je i potpun i optimalan

↳ vremenska i prostorna složenost je i dalje eksponencijalna:

$$O\left(b^{1+\lfloor C^*/\varepsilon \rfloor}\right)$$

$d = \lfloor \frac{C^*}{\varepsilon} \rfloor$ $C^* \Rightarrow$ optimalna dubina rješenja

$\varepsilon \Rightarrow$ najmanja težina grafa

3) pretraživanje u dubinu

↳ sada stablo pretraživanja obilazimo „inorder" postupkom

↳ pretraživanje u dubinu je manje memorijski zahtjevno:

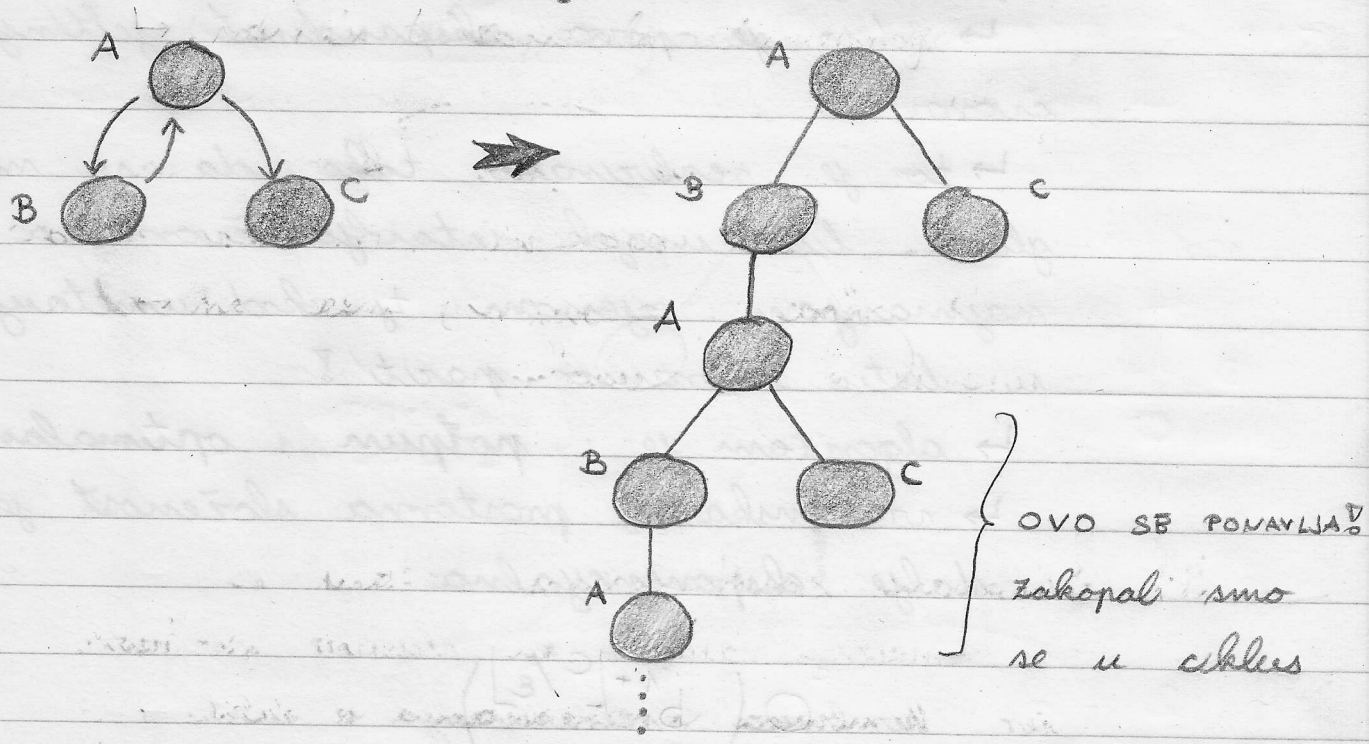
$$O(bm)$$

$m \Rightarrow$ maksimalna dubina stabla

↳ vremenska složenost: $O(b^m)$

↳ neporavna složenost, ako je $m \gg d$

↳ primjetimo da ovaj algoritam nije potpun jer možemo zaglavit u ciklus



↳ ovaj algoritam nije optimalan jer nije potpun, a i kad nađe rješenje ono ne mora biti najbolje jer algoritam pretražuje u dubinu, a ne širinu //

4) ograničeno pretraživanje u dubinu

↳ ograničen broj razina rekapanja u dubinu „k"

↳ prostorna složenost: $O(bk)$

↳ vremenska složenost: $O(b^k)$

↳ ovaj algoritam nije potpun jer ako je rješenje ispod „k"-te razine, nećemo ga pronaći (tj. ako je $d > k$)

↳ također, algoritam nije optimalan jer pretražuje u dubinu

5) iterativno pretraživanje u dubinu

↳ najbolji algoritam slijepog pretraživanja jer kombinira pretraživanja u dubinu i širinu

↳ svaki put krećemo ispočetka i povećavamo granicu dubine za jedan („k++")

↳ time simuliramo pretraživanje u širinu

↳ vremenska složenost: $O(b^d)$

↳ prostorna složenost: $O(bd)$

↳ algoritam je potpun

↳ algoritam je optimalan, ako „k" povećavamo za jedan

↳ čvorove ponavljamo, ali je asimptotska složenost ista kao i kod pretraživanja u širinu

⇒ pretpostavimo da su svi algoritmi slijepog pretraživanja **eksponencijalne vremenske složenosti**

↳ usmjereno pretraživanje će imati istu asimptotsku složenost, ali će u praksi složenost biti znatno bolje od slijepog pretraživanja

⇒ PROBLEM PONAVLJANJA STANJA:

↳ ne želimo da opet ekspandiramo stanja koja smo već ekspandirali

↳ konstit ćemo listu „visited" koja sadrži sva do sad posjećena stanja

↳ tu listu najčešće implementiramo nekom hash-listom

↳ takav način rješenja problema ponavljanja stanja može znatno smanjiti vremensku i prostornu složenost

⇒ KREATIVNI ZADATAK PROGRAMERA:

- dobro apstrahirat problem na:
  a) stanja
  b) prijelaze
  c) cijene prijelaza

HEURISTIČKO PRETRAŽIVANJE

⇒ slijepa pretraga raspolaže isključivo egzaktnim podacima
↳ ne koriste nikakvu **dodatnu informaciju** o prirodi problema

⇒ HEURISTIKA:
↳ iskustvena pravila o prirodi problema

$h: S \to R^+$ ⇒ heuristička funkcija pridjeljuje svakom stanju $s \in S$ procjenu udaljenosti od tog stanja do ciljnog stanja

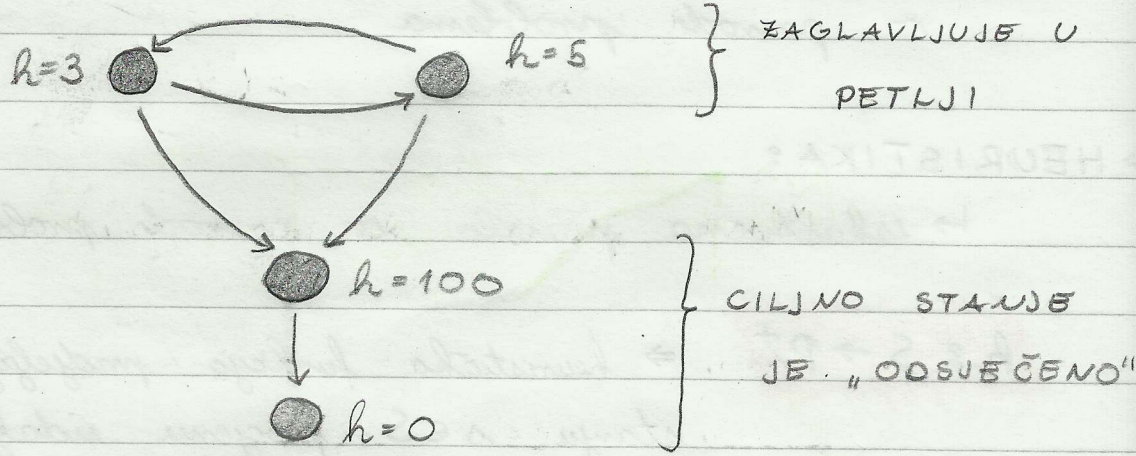
⇒ dodatan nadatak programera kod heurističkog pretraživanja je dobro i jeftino definiranje i izračunavanje heurističke funkcije
⇒ dogovorno, na ciljno stanje, heuristička funkcija treba vraćat nulu

⇒ promotrit ćemo tri algoritma:
a) „najbolji - prvi"
b) „A*"
c) „uspon - na - vrh"

1) najbolji prvi

↳ uvijek idemo u smjeru gdje heuristika kaže da je najbolje

↳ skladno tome, pretpostavimo da nemamo dođ do rješenja:



↳ ovaj algoritam nema vraćanje natrag i u slučaju loše heuristike je **nepotpun** i **neoptimalan**

↳ on spada u POHLEPNE ALGORITME jer donosi odluku na temelju lokalne proizvoljno dobre heuristike

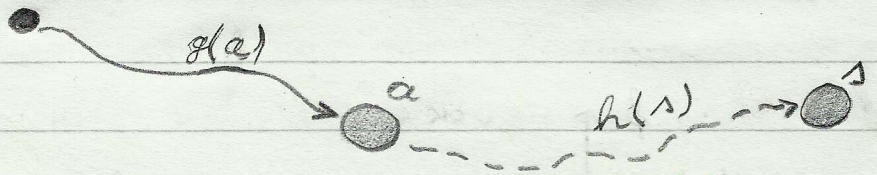
↳ vremenska i prostorna složenost:

- $O(b^m)$ jer je ovo normalno pretraživanje u širinu na manjim iznimama

BITAN ALGORITAM 8.

2) algoritam A*

↳ kombinira algoritme:
- najbolji prvi
- pretraživanje s jednolikom cijenom



$$f(n) = g(n) + h(state(n))$$

↓

ukupna cijena se računa na temelju stvarne cijene trenutnog čvora i procjene cijene traženog čvora pomoću heurističke funkcije

↳ ovaj algoritam se brine o tome da ako nađemo neki put do nekog čvora, a već postoji jeftiniji put do tog čvora, onda taj put ne uzimamo u obzir

↳ algoritam je **potpun**

↳ algoritam je optimalan, ali ako je heuristika OPTIMISTIČNA - za svako stanje daje manju ili jednaku cijenu nego što ona stvarno jeste ($h(s) \leq h^*(s)$)

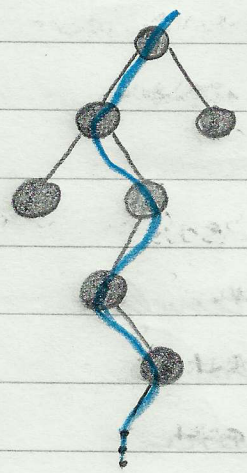
↳ u suprotnom, možemo zaobići najbolji put zbog precjenjivanja heuristike

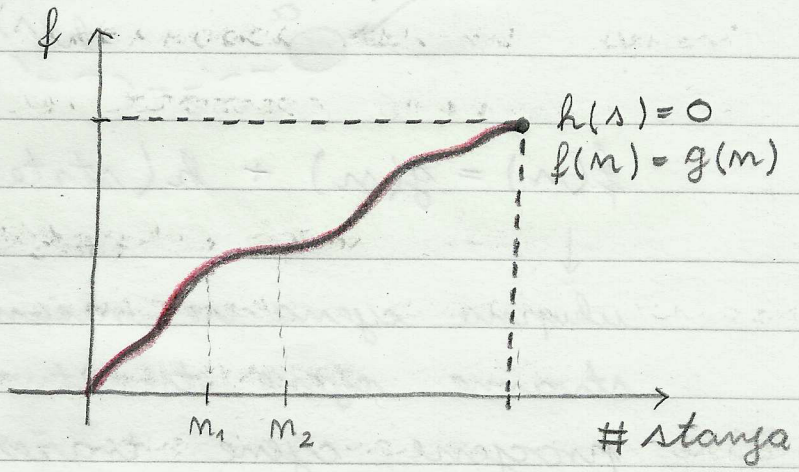
↳ **složenost**: O

⇒ **koje su heuristike dobre?**

↳ poželjno je da $f(n)$ monotono raste

$$\forall n_2 \in expand(n_1) \Rightarrow f(n_2) \geq f(n_1)$$





↳ takva heuristika je KONZISTENTNA

↳ kod ponovljenih stanja, ne moramo provjeravati cijenu već razvorenih čvorova (ta cijena će biti sigurno manja il jednaka)

⇒ **koristan zaključak:**

$$f(n_1) \leq f(n_2)$$

$$g(n_1) + h(state(n_1)) \leq g(n_2) + h(state(n_2))$$

$$g(n_1) + h(s_1) \leq g(n_1) + c + h(s_2)$$

$$h(s_1) \leq h(s_2) + c$$

$c \rightarrow$ put od „n_1" do „n_2" (stvarni put)

⇒ primijetimo da nije svaka optimistična heuristika konzistentna:

- ako imamo optimističnu nekonzistentnu heuristiku, onda ćemo dobit neko rješenje prije optimalnog
- no još uvijek ćemo dobit najbolje rješenje jer se algoritam A* vrti sve dok imamo čvorove u listi **open**

⇒ izbor heuristike može drastično utjecat na ponašanje našeg algoritma te smanjiti složenosti (želimo OPTIMISTIČNU, NAJOBAVJEŠTENIJU i heuristiku sa malom složenosti izračunavanja)

⇒ VARIJANTE ALGORITMA A*:

a) bez liste **closed**

b) bez otvaranja već otvorenih čvorova uz uvjet da je heuristika konzistentna

c) sa listom closed i bez otvaranja otvorenih čvorova, ali uz **pathmax**-korekciju

⇒ SVOJSTVO DOMINACIJE:

↳ ako su oba algoritma optimalna (s optimističnim heuristikama), algoritam A_1^* dominira nad algoritmom A_2^* ako i samo ako:

$$\forall s \in S \quad h_1(s) \geq h_2(s)$$

↳ A_1^* ima više informacija od A_2^* (kažemo da je A_1^* OBAVJEŠTENIJI)

3) uspon na vrh

↳ kao pohlepno pretraživanje „najbolji-prvi" sa varijkom da ne pohranjuje čvorove

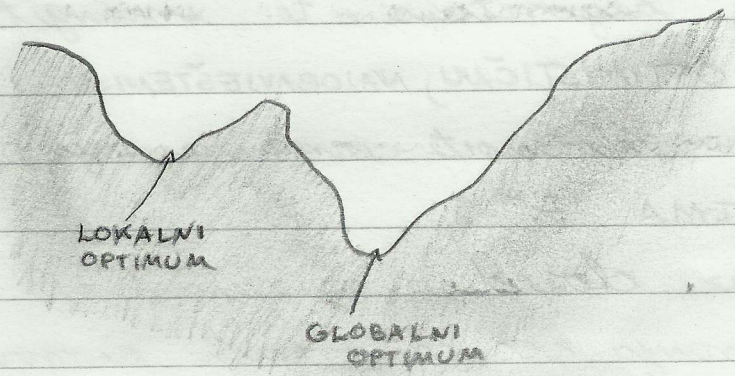
↳ jeftin, ali može raglavit u lokalnom optimumu

↳ složenost:

a) VREMENSKA: $O(m)$

b) PROSTORNA: $O(1)$

↳ samo hivorski ide u bolj čvor dok god bolj čvor postoj



IGRANJE IGARA

⇒ problem pretraživanja stanja, al' sada postoj protivnik u igranju igre

⇒ u svakom stanju treba donijeti optimalnu odluku, tj. treba pronaći optimalnu strategiju

⇒ mi se bavimo deterministickim igrama sa dva igrača, potpunom informacijom i sumom nula

⇒ ovdje nemamo stablo pretraživanja, već imamo **stablo igre**

↳ problem je drugačije definiran VIDI SLAJDOVE!

⇒ <u>igrače ćemo nazvati:</u>

MAX → računalo (max!)

MIN → protivnik

↳ „min" nastoj minimizirat dobitak igrača „max"

↳ „max" nastoj maksimizirat svoj dobitak

↳ prvi igrač je „max" i igrači se nakon svakog poteza izmjenjuju

↳ ova metoda rješavanja problema igranja igre naziva se:

MINIMAKS

↳ oba igrača koriste optimalnu strategiju

⇒ MINIMAKS VRIJEDNOST:

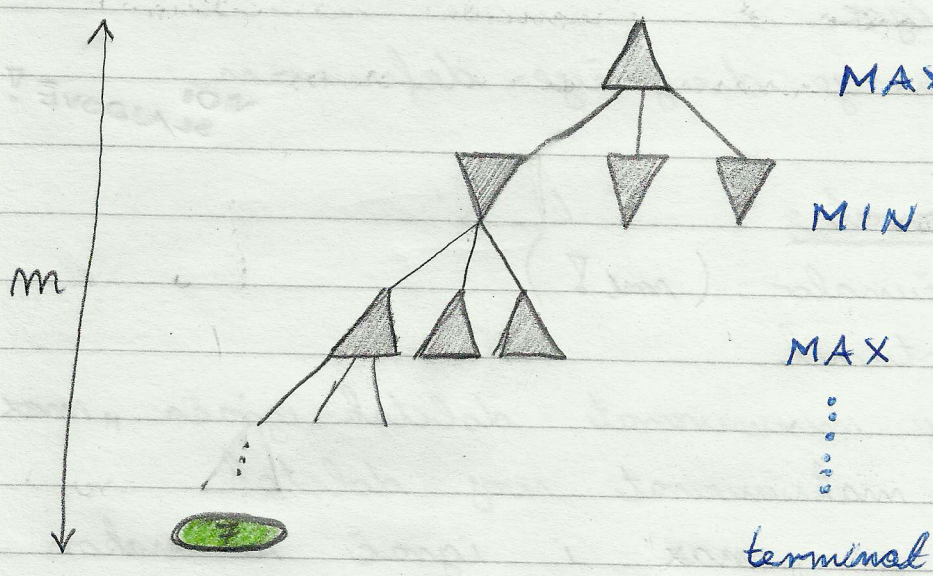
$$m(s) = \begin{cases} utility(s) & terminal(s) \\ \max_{t \in succ(s)} m(t) & "s" \text{ je MAX} \\ \min_{t \in succ(s)} m(t) & "s" \text{ je MIN} \end{cases}$$

↳ vidimo da je minimaks vrijednost čvora definirana rekurzivno

⇒ REALIZACIJA ALGORITMA:

↳ ostvarena dvojena međusobno rekurzivnom funkcijama

↳ time dobivamo pretraživanje u dubinu



"b" → faktor grananja

↳ vremenska složenost: $O(b^m)$

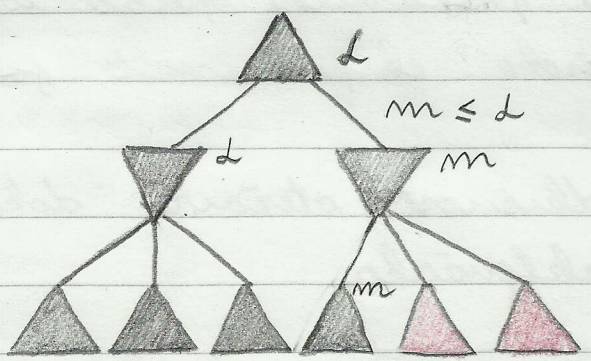
↳ prostorna složenost: $O(m)$ → ne ekspandiramo prije zakopa u rekurziji, pa nije $O(bm)$

⇒ NESAVRŠENE ODLUKE:

- rijetko možemo pretražit cijelo stablo odluke, pa pozivamo se na heuristikama
- IDEJA:
  - ↳ idemo do dubine „d" i onda pozivamo heuristiku

⇒ PODREZIVANJE ALFA-BETA: ISPIT!

- broj stanja igre eksponencijalno raste sa brojem poteza
- primjenom podrezivanja taj broj možemo prepolovit
- ideja ⇒ odredit minimaks vrijednost bez obilaženja svih čvorova



MAX

MIN

MAX

α-podrez (podrezujemo ispod MIN čvora)

NE TREBAMO ISTRAŽITI!

O LOGIKAMA

⇒ logika nam treba za **prikaz znanja** te za modeliranje **zaključivanja**

⇒ izvođenje zaključaka svodi se na dokazivanje teorema

⇒ svaki sustav logike sastoji se od 3 komponente:

a) sintaksa
b) semantika
c) teorija dokaza

⇒ **ekspresivnost logika**: neke logike su više izražajne, a neke manje - ovisno o tome što možemo prikazati / opisati logikom

↳ nedostatak visoke ekspresivnost je **gubitak odlučivosti**:

- mala odlučivost otežava dokazivanje i izvođenje zaključaka

⇒ MODEL ⇒ jedna istinita interpretacija formule

⇒ model stvarnog svijeta

↳ npr. formula A V B ima 3 modela!

A	B	A V B
0	0	0
0	1	1
1	0	1
1	1	1

} MODELI FORMULE AVB

⇒ LOGIČKA POSLJEDICA:

↳ formula G je logička posljedica formula $F_1, \ldots, F_n$ (premisa) ako i samo ako svaka interpretacija koja zadovoljava formulu $F_1 \wedge \ldots \wedge F_n$ također zadovoljava formulu G

↳ pišemo: $F_1, \ldots, F_n \models G$

⇒ formula koja je valjana (tautologija) označava se sa: $\models F$

⇒ dosad smo opisali PROPOZICIJSKU LOGIKU

↳ no, njome ne možemo opisati riječi poput: „Svaki", „Neki", ...

↳ propozicijska logika nije dovoljno ekspresivna za takve probleme

↳ stoga, uvodimo LOGIKU PRVOG REDA koja je ekspresivnija, ali u njoj nećemo moći sve dokazati

⇒ za detaljno objašnjenje propozicijske logike i logike prvog reda pogledaj prezentaciju i službenu skriptu !

LOGIKA PRVOG REDA

⇒ ATOM: predikat primijenjen na neke varijable

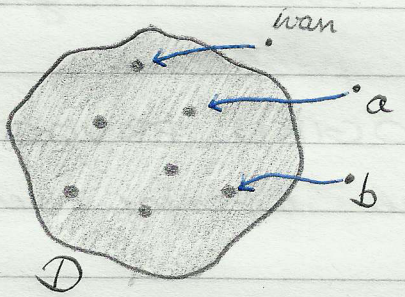
$$P(f(x), g(x,y), a, x)$$

PREDIKATNI SIMBOL — FUNKCIJSKI SIMBOL — IZRAZ — KONSTANTA — VARIJABLA

PREDIKAT

⇒ interpretacija logike prvog reda

- domena D
- preslikavanje svake konstante u element iz D
- dodatna dva preslikavanja višeg reda



⇒ NAPOMENA: o implikaciji razmišljaj kao o filtru



↳ samo ako vrijedi A, onda je želim nešto reći dodatno o B

↳ zato je implikacija istina čim je A lažan

⇒ NEODLUČIVOST LOGIKE PRVOG REDA:

↳ svaka formula logike prvog reda ima beskonačno mnogo interpretacija

↳ samim time, ne možemo raditi iscrpno pretraživanje svih mogućih interpretacija (tablica istinitost ima beskonačno mnogo redaka)

PRESLIKAVANJE PRIRODNOG JEZIKA U LOGIKU PRVOG REDA

PR: Preslikaj rečenicu: „Svi studenti su pametni" u logiku prvog reda!

↳ TOČNO: $\forall x \, (S(x) \rightarrow P(x))$

- „za sve što je student dodatno kažemo da je i pametno"

↳ NETOČNO: $\forall x \, (S(x) \wedge P(x))$

- „u mojem svijetu (domeni) je student i jest pametno, al nitko više ne postoj"

↓

isključujemo mogućnost drugih objekata, pa ovakve formule ne smijemo pisati

⇒ <u>neke specifičnosti:</u>

- uvijek univerzalni, unutra implikacija ($\rightarrow$)
- uvijek specifični, unutra operator $\wedge$ („i")

↓

NE VRIJEDI UVIJEK!

LOGIČKO ZAKLJUČIVANJE

⇒ TEORIJA DOKAZA:

↳ ljudi pokušavaju izvesti logičku posljedicu iz zadanih premisa koje vrijede

↳ teorija dokaza je **učinkovita**, ali je i dalje neodlučiva

⇒ DEDUKCIJA: $F_1, \ldots, F_n \vdash G$

↳ dedukcija je formula ako i samo ako je G moguće izvesti iz niza formula $F_1, \ldots, F_n$ (premisa) pomoću pravila zaključivanja

$$F \vdash G$$

⇒ čitamo: F uvod G

F dedukcijski povlači G

⇒ vrlo jako područje umjetne inteligencije je PODRUČJE AUTOMATSKOG ZAKLJUČIVANJA:

↳ engl. „ATP" ⇒ automated theorem proving

↳ ono služi za dobivanje sljedećeg koraka iz neke bare znanja

⇒ NAPOMENA: semantičko povlačenje (relacija $\models$) smo dokazivali pomoću tablica istinitosti - ovo više ne možemo, ali će biti korisno

PRAVILO ZAKLJUČIVANJA:

- mora biti ispravno
- mora biti potpuno

↳ ispravnost i potpunost povezuju semantiku i teoriju dokaza

VRLO BITNO ZA RAČUNARSTVO!

⇒ METODA REZOLUCIJE:

↳ sastoji se od samo jednog pravila zaključivanja:

$$A \vee F,\ \neg A \vee G \vdash F \vee G$$

⇒ konjuktivna normalna forma: (CNF)

$$(A \vee B \vee C) \wedge (D \vee E) \wedge H$$

KLAUZULA

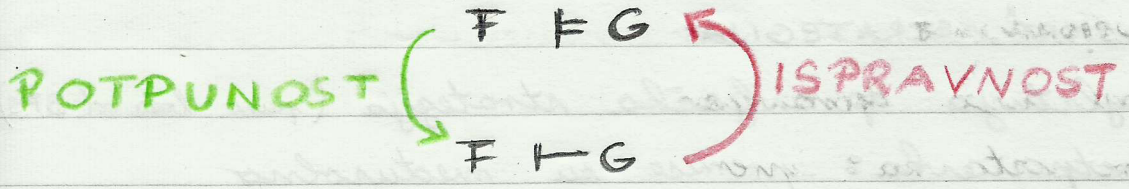
↳ svaka formula se može dovesti u konjuktivnu normalnu formu

↳ za tu pretvorbu postoji algoritam!

⇒ klauzalni oblik:

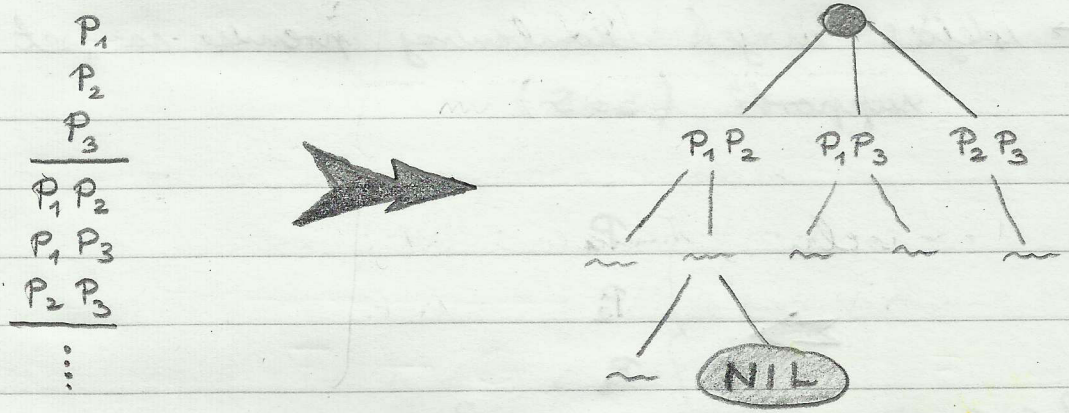
- skup skupova
- unutarnji razred su „ili", a vanjski su „i"
- klauzalni oblik je „bolji" zapis CNF-a

$$\{\{A, B, C\}, \{D, E\}, \{H\}\}$$



⇒ *kako najbolje izračunati klauzule ir skupa klauzula?*

↳ *u algoritmu ih trebamo odabirati par po par te je to naš prostor pretraživanja*



↳ *želimo potpuni algoritam pretraživanja ovog prostora - to može biti jako isplyujuć i složena, pa postoje* STRATEGIJE POJEDNOSTAVLJENJA *i* STRATEGIJE UPRAVLJANJA

VIDI SLAJDOVE!

⇒ „SET OF SUPPORT" STRATEGIJA:

↳ najbrža upravljačka strategija (puno brža od BFS)

↳ pretpatavka: premise su međusobno konzistentne jer inače uopće nema smisla ovo izvođenje zbog svojstva logičke posljedice

↳ nama je cilj dobit NIL, a pošto su premise konzistentne, nema ih smisla kombinirati jer između njih nećemo dobit NIL

↳ ideja: uvijek kombiniraj premise sa „set of support" (SoS)

P_1
P_2
P_3
$\neg G$ } SoS

⇒

P_1
P_2
P_3
$\neg G$ } SoS
$P_1\ \neg G$

ZAKLJUČIVANJE UPORABOM PREDIKATNE LOGIKE

⇒ postoj postupak kojim se ovo radi:

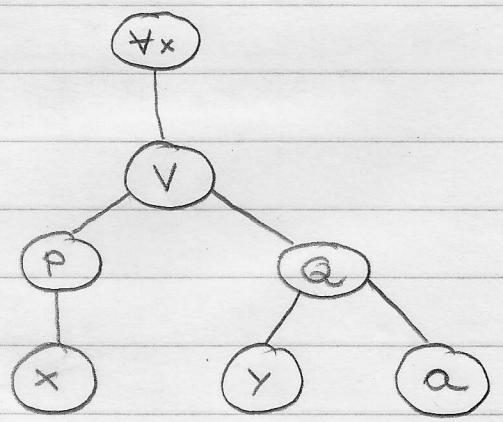
1) pretvorba u CNF
2) supstitucija
3) unifikacija
4) rezolucija

⇒ SKOLEMIZACIJA:

- postupak pretvorbe u CNF kojim se uklanjaju egzistencijalni varijabl
- umjesto tih varijabl uvodimo proizvoljne konstante i funkcije, ovisno o klauzuli
- to smijemo radit jer za samo dokazivanje NIL-a na kraju nije bitno koja je stvarna vrijednost tih varijabl

⇒ formulu u računalu prikazujemo kao stablo:

$\forall x \, (P(x) \vee Q(y, a))$ ⇒



⇒ **primjer supstitucije:**

↳ imamo $P(x)$

↳ imamo i dane supstitucije:

$$\alpha = \{ f(z)/x \}$$

$$\beta = \{ a/z \}$$

↳ primjenimo kompoziciju supstitucija:

$$(P(x)\,\alpha)\,\beta = P(f(a))$$

$$\downarrow$$

$$\alpha \circ \beta = \{ f(a)/x,\ a/z \}$$

kompozicija

u matematici je obratno

⇓

postupak je rigorozan, prvo trebamo izračunati skupove λ_1, λ_2 i λ_3 te izračunati razliku:

$$\lambda_1 \setminus \lambda_2 \setminus \lambda_3$$

↳ ta razlika je ona supstitucija koju moramo izvršit

↳ to moramo izračunati da nebi ponavljali supstitucije iz „α" i „β"

⇒ **algoritmus unifikacije:**

↳ pogledajmo primjer:

$$\left.\begin{array}{l} P(x, f(b)) \\ P(a, y) \end{array}\right\} \quad S = \{a/x, f(b)/y\}$$

↳ formule prikazujemo kao liste:

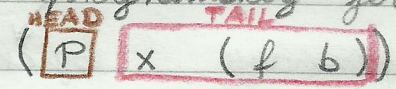
[P, x, [f, b]]

[P, a, y]

⇓

druga sintaksa koristi oble zagrade

zbog programskog jezika LISP:



(P a y)

LOGIČKO PROGRAMIRANJE U PROLOGU

⇒ algoritam:
↳ ima logiku i upravljanje

DEKLARATIVNO PROGRAMIRANJE

* programiranje s teorema

- funkcijsko programiranje
  ↳ npr. Heskell
- logičko programiranje
  ↳ npr. Prolog

⇒ ovo su jezici visokog stupnja apstrakcije, pa su prilično neučinkoviti

⇒ **PROLOG:**
↳ „Programming in Logic"
↳ alat za obradu prirodnog jezika
↳ **koncepti:**
a) rekurzija
b) unifikacija
c) postupak povratka („backtrack")

⇒ Prolog koristi podskup FOL-a:
Hornova klauzalna logika

⇒ Hornova klauzula je disjunkcija literala sa najviše jednim pozitivnim literalom:

$$\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_m \vee Q$$

↳ zapisano drugačije:

$$(P_1 \wedge P_2 \wedge \dots \wedge P_m) \rightarrow Q$$

⇒ svaka definitna klauzula definira neko pravilo zaključivanja

⇒ Hornova klauzula koja sadrži samo negativne literale je ciljna klauzula

⇒ primjetimo da su Hornove klauzule ograničene ekspresivnost, ali nam omogućuju korištenje rezolucije, pa smo spremni na taj kompromis
↳ također, efikasnost zaključivanja je puno veća (linearna složenost)

⇒ ULANČAVANJE UNAZAD:
↳ metoda dokazivanja vrlo slična linearnoj rezoluciji - rezolventa se koristi za daljnju rezoluciju i to se ponavlja

⇒ Hornove klauzule su ratvorene s obzirom na skup Hornovih klauzula za metodu rezolucije

⇒ ulančavanje unazad u Prologu:

VIDI SLAJDOVE!

- poredak klauzula definira programer
  - ↳ prvo se navode specifičnije, a zatim općenitije klauzule (odstupanje od logike jer u logici redoslijed nije bitan)

⇒ KAKO PROLOG DOKAZUJE?

↳ generira stablo dokaza

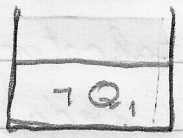
PR: Neka imamo klauzule:

$$P_1 \wedge P_2 \rightarrow Q_1$$
$$P_3 \wedge P_4 \rightarrow P_1$$

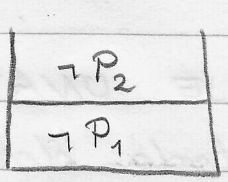
Pokažimo tijek izvođenja i stog u Prologu!

a) 1) $\neg P_1 \vee \neg P_2 \vee Q_1$

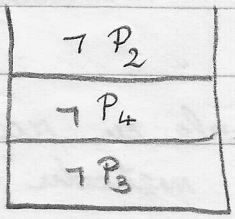
2) $\neg P_3 \vee \neg P_4 \vee P_1$



b) „kombiniraj klauzule i puni stog"



c) „kombiniraj klauzule i puni stog"



STOG

$\neg Q_1$

⇓

$\neg P_2$
$\neg P_1$

⇓

$\neg P_2$
$\neg P_4$
$\neg P_3$

cilj je isprazniti stog!

STOG JE CRTAN OBRATNO!

Zad: Prikaži rad prologa za sljedeća pravila i činjenice:

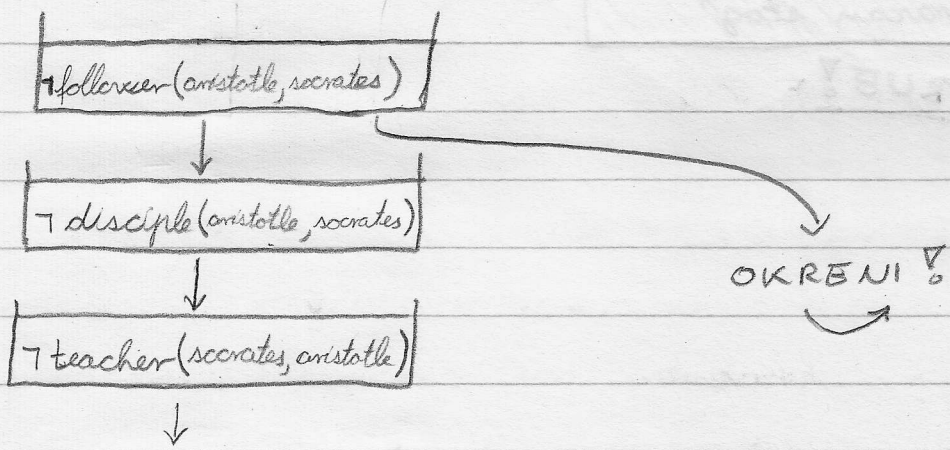
```
follower(X, Y) :-
    disciple(X, Y).
follower(X, Y) :-
    disciple(X, Z),
    follower(Z, Y).

disciple(X, Y) :-
    teacher(Y, X)
```

```
teacher(socrates, plato)
teacher(cratylus, plato)
teacher(plato, aristotle)
```

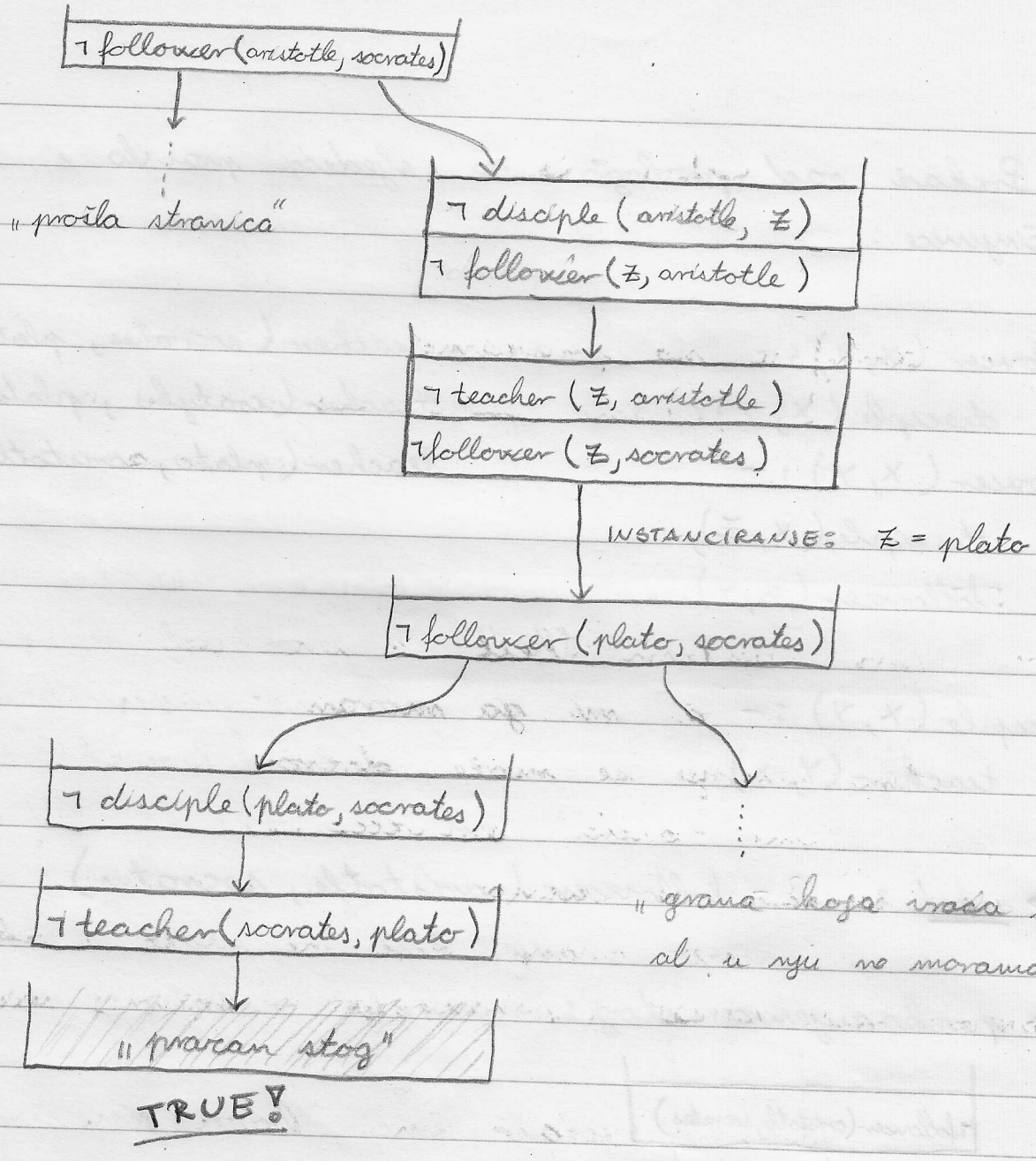
↳ upit: ?- follower(aristotle, socrates)

↳ prikazujemo stog:



nemamo dalje!

↳ ova grana vraća False, pa sada radimo „backtrack" na mjesto grananja (odmah prvo stanje stoga!)


¬ follower(aristotle, socrates)
„prošla stranica"
¬ disciple(aristotle, Z)
¬ follower(Z, aristotle)
¬ teacher(Z, aristotle)
¬ follower(Z, socrates)
INSTANCIRANJE: Z = plato
¬ follower(plato, socrates)
¬ disciple(plato, socrates)
¬ teacher(socrates, plato)
„pravan stog"
TRUE!
„grana koja vraća false, al u nju ne moramo ić"

CLIPS

⇒ sustav temeljen na pravilima

⇒ „expert system" – sustav jako dobar u nekom uskom području znanja

⇒ CLIPS:

↳ „Expert System Shell" → on radi zaključivanje, a mi ga moramo napuniti znanjem

↳ ljudsko znanje se može dobro formalizirati pomoću „ako - onda" pravila („chunk - of - knowledge")

↳ ovi sustavi se puno koriste u medicini jer je tamo jako puno znanja koje ne može svaki doktor sve znati (ponajprije u internoj medicini!)

⇒ ukratko, **ekspertni sustav** ima bazu znanja i radi zaključivanje pomoću te baze

↳ danas se više radi zaključivanje i učenje pomoću STROJNOG UČENJA, ali ne bi smjeli zanemariti važnost ekspertnih sustava!

AKO ..... ONDA .....

UVJET AKCIJA

(akciju nemamo u logici)

⇒ shema rada produkcijskog sustava:

↳ vidi na slajdovima!

↳ ideja je naći pravila koja su dobra, uraditi jedno koje će „upaliti", i ponovi postupak

⇒ PRODUKCIJSKI SUSTAVI:

a) su nemonotoni

↳ ako nešto novo zaključim, moguće je da to opovrgava neko staro pravilo („truth maintainance")

↳ ovoga u logici nema!

(logika je monotona)

⇒ pogledaj primjer ULANČAVANJA PRAVILA UNATRAG na slajdovima!

MODELIRANJE NEIZVJESNOSTI

⇒ inteligentno rješavanje problema nesigurnih, nepouzdanih, nepotpunih i nejasnih podataka

⇒ NUMERIČKI ORIJENTIRANI OKVIRI:

a) Bayesova shema
b) Neuronske skupovi i neuronska
c) Faktori izvjesnosti
d) Dempster - Shaferova teorija

} OVIME SE MI BAVIMO

⇒ **BAYESOVA SHEMA:**

↳ prikazat ćemo ga na Monty Hall problemu (vid slajdove 5.)

↳ temelji se na teoriji vjerojatnosti

↳ imamo slučajan pokus i prostor mogućih događaja

$$X = \{x_1, x_2, \dots, x_M\}$$

↓ skup mogućih događaja

↳ definicija vjerojatnosti: $p: P(X) \to [0, 1]$

↳ neovisnost događaja:

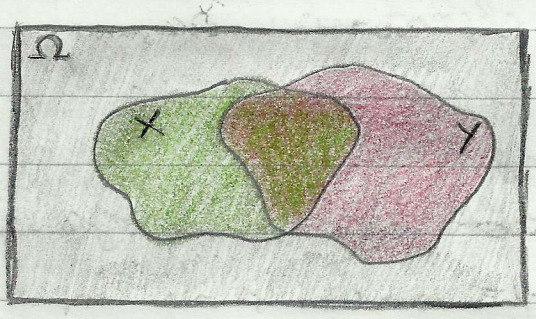
• događaji su neovisni ako vrijedi:

$$p(x_1 \wedge x_2) = p(x_1) \cdot p(x_2)$$

↳ za Bayesove sheme je bitna UVJETNA VJEROJATNOST:

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

↳ ovo možemo dobro prikazati preko dijagrama:



↳ „izvod" Bayesovog pravila:

$$P(X|Y) = \frac{P(X \wedge Y)}{P(Y)} \qquad P(Y|X) = \frac{P(X \wedge Y)}{P(X)}$$

$$P(Y \wedge X) = \underbrace{P(Y|X) \cdot P(X) = P(X|Y) \cdot P(Y)}$$

iz ovoga zaključujemo:

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

↑

ovaj oblik Bayesovog pravila znamo, no u umjetnoj inteligenciji se koristi drugi oblik

↳ Bayesova shema nam omogućuje da iz dokaza E izvedemo hipotezu H s nekom vjerojatnošću:

- omogućuje nam da raklyucivanje ABDUKTIVNO - suprotno od implikacije (za to smo u logici pokarali da nije valjano pravilo)

↳ no, ova jednostavna Bayesova shema za stvarne probleme nije dovoljna, pa imamo sljedeća proširenja:

a) imamo više hipoteza („m" - hipoteza)
b) imamo više dokaza („n" - dokaza)

$$p(H_i \mid E_1 E_2 \dots E_n) = \frac{p(E_1 E_2 \dots E_n \mid H_i)\, p(H_i)}{p(E_1 E_2 \dots E_n)}$$

↑

imamo kombinatornu eksploziju, pa trebamo <u>pretpostavit nezavisnost</u> dokaza, te dobivamo:

$$p(H_i \mid E_1 E_2 \dots E_n) = \frac{p(E_1 \mid H_i)\, p(E_2 \mid H_i) \dots p(E_n \mid H_i)\, p(H_i)}{\sum_{k=1}^{m} p(E_1 \mid H_k)\, p(E_2 \mid H_k) \dots p(E_n \mid H_2) \cdot p(H_2)}$$

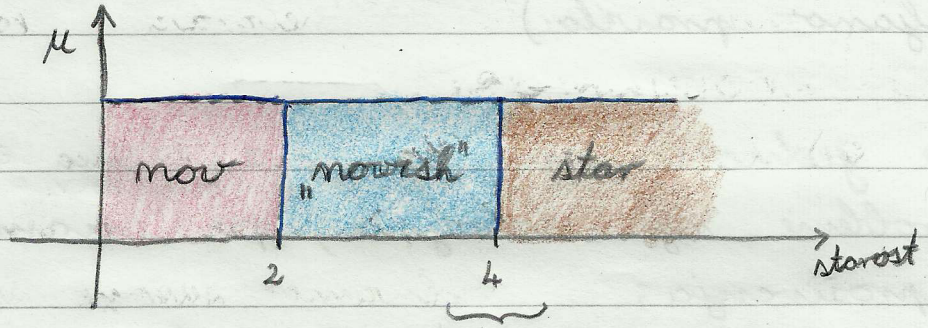
BITNO!!

⇒ NEIZRAZITA LOGIKA:

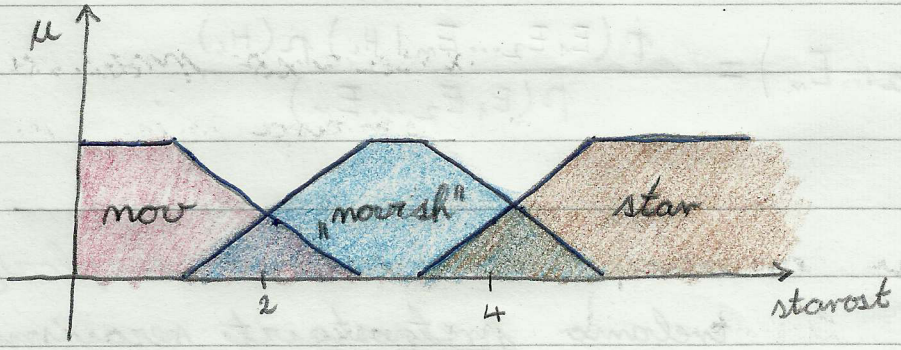
↳ ideja je „računanje s riječima"

↳ stoga, drugi naziv je i „kvazi" - logika

PR: Modeliranje starosti akumulatora:



previše diskretan prijelaz,
pa trebamo malo bolje prijelaze



⇓

ovo su „KVAZI" SKUPOVI jer su subjektivni i neformalni, služe za oblikovanje ljudskih riječi i ljudskog znanja //

↳ JEZIČNA VARIJABLA:

- varijabla koja umjesto standardne numeričke vrijednosti poprima vrijednost riječi il rečenice
- predstavlja poveznicu između prirodnog jezika i neizrazite logike

↳ JEZIČNI MODIFIKATORI:

a) **koncentracija** - kvadriranje funkcije pripadnosti i time dobivamo sužење skupa
- odgovara riječi „VRLO"

b) **dilatacija** - korjenovanje funkcije pripadnosti i time dobivamo proširenje skupa
- odgovara riječi „MANJE-VIŠE"

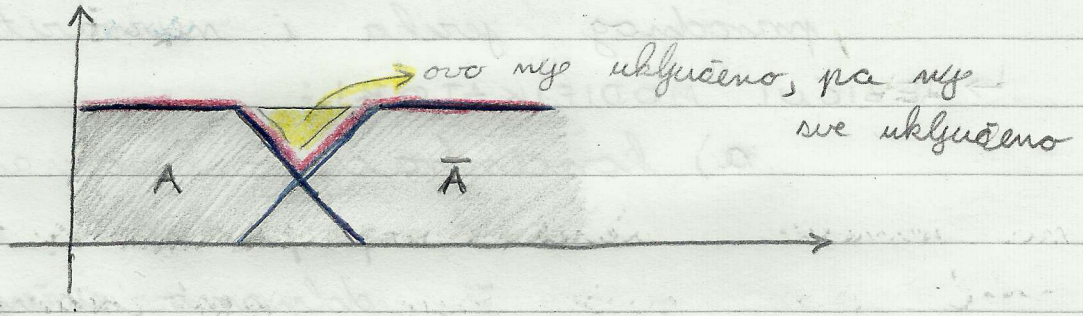
↳ dodatne operacije neizrazite logike:

- **logičko „ILI"** ⇒ unija dva skupa koju dobivamo ako uzmemo „max" od funkcija pripadnosti
- **logičko „I"** ⇒ presjek dva skupa koj dobivamo uzimanjem „min" od funkcija pripadnost
- **negacija** ⇒ razlika konstante 1 i funkcije pripadnosti

↳ primijetimo da nije sve „crno-bijelo" jer ne vrijedi zakon isključenja trećega, tj. ne vrijedi: $A \vee \bar{A} \equiv T$

$$\Downarrow$$

pogledajmo sliku:

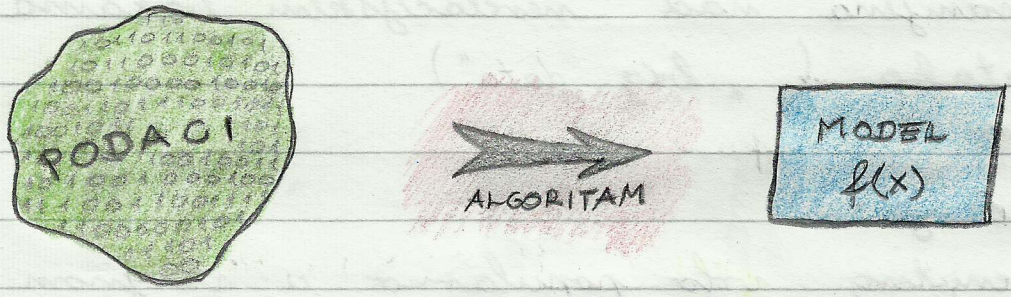


↳ slično možemo pokazati da ne vrijedi:

$$A \wedge \bar{A} \equiv \perp$$

↳ bitna nam je IMPLIKACIJA (vid slajdove!)

STROJNO UČENJE



↳ na temelju viđenih podataka izgrađujemo model koji može raditi predviđanje za nove podatke

⇓

MODEL MORA DOBRO GENERALIZIRATI!

⇒ **strojno učenje koristimo za:**

a) probleme presložene za algoritamsko rješavanje, tj. na način na koje smo do sada rješavali probleme

↳ UI - potpuni problem

b) ogromne količine podataka („data mining")

c) sustave koji se dinamički mijenjaju

PODACI → INFORMACIJE → ZNANJE → ODLUKA

⇒ dubinska analiza podataka:

- „data mining"
- primjena nad nerelacijskim bazama podataka („big data")

⇒ duboko učenje:

↳ trenutno vrlo popularno u strojnom učenju i u umjetnoj inteligenciji

↳ koristi se nad velikom količinom podataka

↳ npr. vid se perspektiva u polju autonomnih vozila

⇒ VRSTE STROJNOG UČENJA:

a) NADZIRANO UČENJE:

↳ podaci u obliku (ulaz, izlaz) = (X, Y)

↳ treba naći preslikavanje $\hat{Y} = f(X)$

↳ diskretan Y ⇒ klasifikacija

↳ kontinuiran Y ⇒ regresija

b) NENADZIRANO UČENJE:

↳ dani su podaci, al ne i ciljne vrijednosti (nemamo Y)

↳ treba naći pravilnost u podacima

c) PODRŽANO / OJAČANO UČENJE:

↳ učenje optimalne strategije na temelju pokušaja

NAIVAN BAYESOV KLASIFIKATOR

⇒ metoda strojnog učenja (algoritam)

⇒ bitna nam je Bayesova shema poopćena na „m" hipotera i „n" dokaza

⇒ par na pojam vjerodostojnost ?

$p(A \mid h)$

↳ ovo je vjerodostojnost hipotere „h" uz zadani „A"

↳ do sada smo čitali:

- vjerojatnost od („A") uz uvjet hipotere „h"

⇒ <u>PRETPOSTAVKA KLASIFIKATORA</u>: BITNO!

$$p(h_i \mid \vec{x}) = \frac{p(\vec{x} \mid h_i) \cdot p(h_i)}{\sum_{j=1}^{k} p(\vec{x} \mid h_j) \cdot p(h_j)} \qquad \vec{x} = (x_1 \dots x_n)$$

↳ pretpostavimo neovisnost vektora značajki $\vec{x}$, uz neku zadanu hipotezu „h_i": $x_i \perp x_j \mid h_i$

(naivna pretpostavka, al radi dobro ?)

↳ dobivamo:

$$p(h_i \mid \vec{x}) = \frac{p(x_1 \mid h_i) \cdot p(x_2 \mid h_i) \cdot \dots \cdot p(x_n \mid h_i) \cdot p(h_i)}{\sum_{j=1}^{k} p(x_1 \mid h_j) \cdot \dots \cdot p(x_n \mid h_j) \cdot p(h_j)}$$

⇒ možemo imati pojednostavljenje, ako nas ne zanima vjerojatnost u nazivniku:

$$h_{MAP} = \underset{h_i}{\operatorname{argmax}}\, p(h_i \mid \vec{x})$$

$$h_{NAP} = \underset{h_i}{\operatorname{argmax}} \left(p(h_i) \prod_{j=1}^{n} p(x_j \mid h_i) \right)$$

PRIMJER: Je li dan za igrat odbojku? (VIDI SLAJDOVE!)

$p(da) = 9/14$

$p(ne) = 5/14$

DA	NE
$p(sunčano \mid da) = 2/9$	$p(sunčano \mid ne) =$
$p(oblačno \mid da) = 4/9$	$p(oblačno \mid ne) =$
$p(kišno \mid da) = 3/9$	$p(kišno \mid ne) =$

↳ ako negdje imamo vjerojatnost 0 u brojniku, to nam „ubija" sve ostale vjerojatnosti

↳ to ne želimo pa koristimo metode ZAGLAĐIVANJA (engl. smoothing):

$$p(x_j \mid h_i) = \frac{N(x_j \mid h_i) + \lambda}{N(h_i) + \lambda K}$$

⇓

La Place-ovo zaglađivanje

↳ drugi trik koji moramo primijeniti je napisati formulu za h_{MAP} sa logaritmom kako bi izbjegli množenje s nulom

↳ logaritmiranje neće izmijeniti položaj jer je logaritam monotona funkcija

$$h_{MAP} = \underset{h_i}{\operatorname{argmax}} \left(p(h_i) \prod_{j=1}^{n} p(x_j | h_i) \right) \Big| \ln$$

$$h'_{MAP} = \underset{h_i}{\operatorname{argmax}} \left(\ln \left(p(h_i) \prod_{j=1}^{n} p(x_j | h_i) \right) \right)$$

$$h'_{MAP} = \underset{h_i}{\operatorname{argmax}} \left(\ln(p(h_i)) + \sum_{j=1}^{n} \ln(p(x_j | h_i)) \right)$$

dobili smo sumu, pa nam jedna vjerojatnost jednaka nuli neće poništiti sve ostale

STABLA ODLUKE

⇒ metoda strojnog učenja (algoritam)
⇒ algoritam generira stablo
  ↳ u svakom čvoru je **test** atributa
⇒ rezultat je odgovor, a ne vjerojatnost kao kod Bayesovog naivnog klasifikatora

⇒ **ALGORITAM ID3:**
  ↳ engl. Induction of Decision Trees
  ↳ on gradi stablo odluke na način da pametno bira atribut za testiranje
  ↳ spada u POHLEPNE ALGORITME jer jednom kada izabere atribut / čvor, algoritam se ne vraća i ne preispituje odluku
  ↳ **informacijska dobit:**
    • koliko informacije dobivamo izborom nekog atributa
    • radi pomoću **entropije**, tj. nesređenosti podataka
      ↳ **sjeti se:** entropija je maks/maksimalna kada su vjerojatnosti događaja međusobno jednake

$$\text{InfGain}(D, A) = \text{Entropija}(D) - \sum_{v \in \text{Value}(A)} \frac{|D_v|}{|D|} \cdot \text{Entropija}(D_v)$$

↳ želimo što veći pad entropije jer tako znamo da će se naše stablo moći razgranati (više informacije primamo vrlorom atributa koj uzrokuje veći pad entropije) BITNO!

⇒ odgovor na prvi „Q" iz algoritma se slijedova:

↳ ako imamo sljedeći skup:

VJETAR	VLAGA	
jak	visoka	DA
jak	visoka	NE

↳ tada nam potencijalno fali atributa po kojima bi dalje podijelil podatke

⇒ PROBLEM PRENAUČENOSTI:

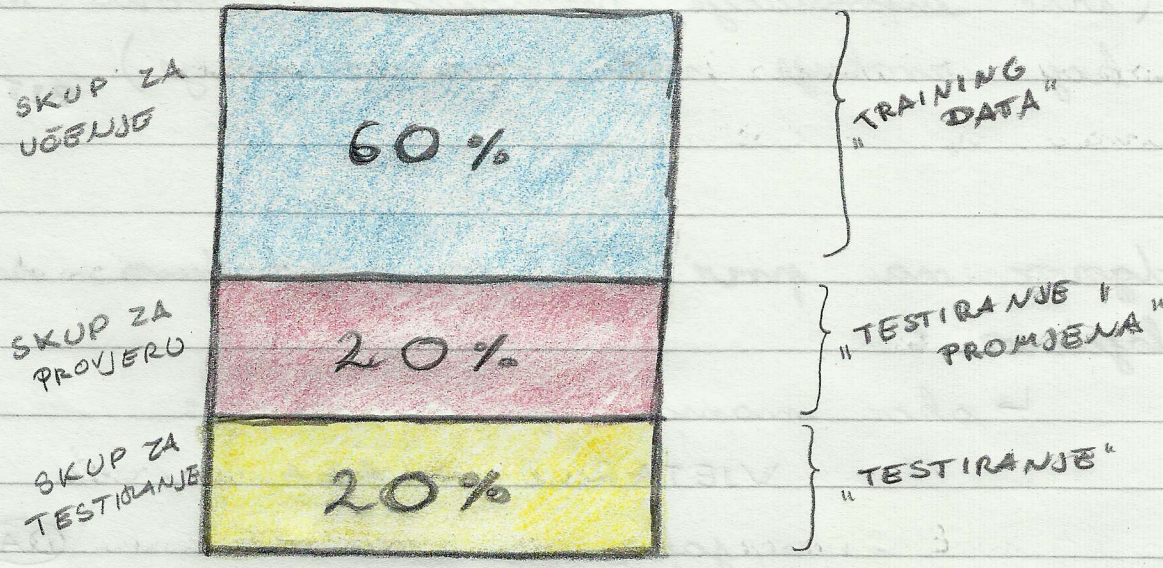
⇒ model dobro radi nad skupom za učenje, ali ne može generalizirat

↳ loš je za ostale primjere ☹

⇒ manifestira se nabrajanjem velikog stabla, osobito ako postoj šum u podacima (npr. pogreške u podacima (netko je krivo unio u sustav))

⇒ RJEŠENJE: podrezivanje, a ratom učenje na 80% podataka i testiranje na preostalih 20% podataka

⇒ ponekad je potrebno uvesti 3 skupa za rješavanje prenaučenosti:



↳ varijante modela treniramo na skupu za učenje, zatim njihovu kompetenciju ispitujemo na skupu za provjeru

↳ nakon odabira najboljeg modela, njegovu točnost ispitujemo na skupu za testiranje

POTPORNO UČENJE

⇒ koristi se najviše za igranje igara i za upravljanje robotima

⇒ koncepte ćemo objašnjavati na problemu Grid World (vidi slajdove)

⇒ ŽIVUĆA NAGRADA: agent dobiva nagradu za svaki potez koji je preživio (može biti negativna)

⇒ ŠUM: ako smo odabrali akciju, ishod ne mora biti 100% vjerojatan očekivanom ishodu akcije
↳ npr. ako smo odabrali lijevo, možda agent krene desno

⇒ POLITIKA: iskustvo stečeno kretanjem svijetom

⇒ MARKOVLJEV PROCES ODLUČIVANJA:
↳ formaliziran problem opisan gornjim pojmovima
↳ to je uređena petorka (vidi slajdove!)

⇒ mi tražimo optimalan put kroz mapu, a taj optimalan put ne mora imati kraj
↳ IDEJA: 1) svakom stanju damo vrijednost
2) krenemo onome sa najvećim očekivanjem
(no, time moramo propagirat informaciju od konačnog do svakog stanja)

⇒ PROCES ODLUČIVANJA U DVIJE ODLUKE:

a) naša odluka: u stanju „s", koju akciju „a" ćemo odabrati?

b) odluka drugog igrača (okolina): za stanje „s" i akciju „a", koje je sljedeće stanje „s'"

⇒ ODABIR PUTA:

a) više il manje

↳ npr. 1, 2, 2 vs 1, 2, 3

b) sad il kasnije

↳ npr. 3, 2, 1 vs 0, 0, 6

• mi želimo VIŠE i SADA, pa uvodimo dodatni pojam:

⇒ **propadanje nagrada:**

- za svaki korak koj napravimo, nagrada se množi sa $\gamma < 1$

⇒ **kako riješiti beskonačne igre** (npr. tetris)?

↳ mi učimo na kraju igre, pa za takvih ne možemo ništa naučiti

↳ znači, morat ćemo igru negdje raustavit kako bi naučil neke stvari (trebamo Markovljev proces odlučivanja)

⇒ **podsjetnik:** mi za svako stanje želimo doznati akciju i tako izgraditi optimalnu politiku

⇒ VRIJEDNOSTI STANJA:

↳ očekivanje: $E(X) = \sum_{x \in X} P(x) \cdot x$

↳ vrijednost stanja jest upravo Q vrijednost najbolje akcije

↳ <u>prvi pokušaj</u>: BITNO!

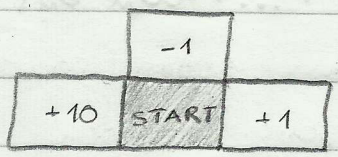
- iteracija vrijednosti
- dobar, ali kvadratno složen

⇒ potporno učenje se svod na pokušajeve kako bi istražili vrijednosti vjerojatnosti iz svijeta.

↳ to nam treba za rješavanje Markovljevog problema optimalne politike

⇒ UČENJE S VREMENSKOM RAZLIKOM:

↳ pogledajmo sljedeću sliku:



$\alpha = 0.5$

(„learning rate")

1) uzmemo neki uzorak (maknemo se npr. prema gore) start, ↑, -1

$V_1(s) = 0 + \alpha[-1 + 0 - 0] = -0.5$

2) uzmemo drugi uzorak (maknemo se npr. prema gore (ali desno)) start, ↑, 1

$V_2(s) = -0.5 + \alpha[1 + 0.5] = 0.25$

kolika novo opažanje utječe

$$V^{\pi}(s) = V^{\pi}(s) + \alpha \left[R(s, \pi(s), s'') + \gamma V^{\pi}(s') - V^{\pi}(s) \right]$$

novo očekivanje — stara očekivanje — novo opažanje — oduzimamo staro očekivanje od novog opažanja

⇒ Q-UČENJE:

↳ modifikacija iterativnog učenja

↳ malo izmijenjene formule (vidi slajdove)

⇒ kako bi osigurali istraživanje, radimo nasumično istraživanje ⇒ faktor ε (vidi slajdove)

⇒ PRIBLIŽNO Q-UČENJE:

↳ opisivanje stanja pomoću vektora značajki

↳ prostor želimo opisat u manje informacija

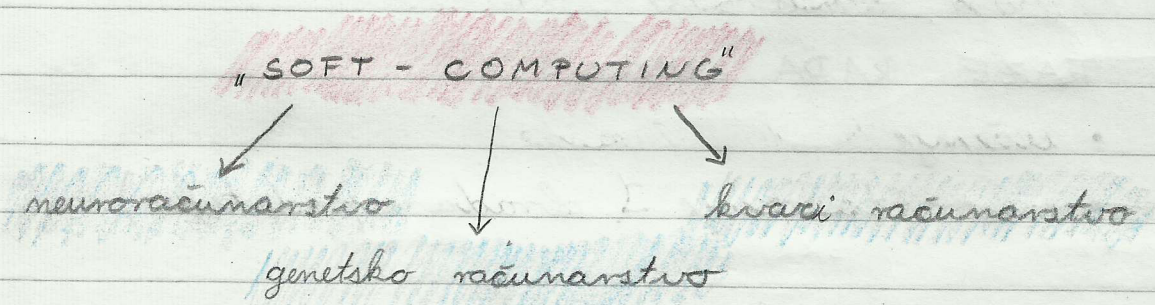
↳ značajke računamo pomoću funkcija udaljenosti

$$Q(s, a) = w_1 \cdot f_1(s, a) + \ldots + w_n \cdot f_n(s, a)$$

UMJETNE NEURONSKE MREŽE

⇒ IDEJA: obradivat informacije na način na koj to radi mozak (živčane stanice mozga)

⇒ spadaju u granu neuroračunarstva



⇒ „soft - computing" je grana računarstva koja se bavi podacima koji nisu uredeni i nisu lagani ra deterministički obradit

⇒ DRUGI PRISTUP PROGRAMIRANJU:

- ne programiraj algoritam rješavanja problema, nego **nauči** računalo da riješi problem

↳ standardni pristup programiranju:

- SIMBOLIČKI PRISTUP

↳ novi pristup programiranju:

- KONEKTIVISTIČKI PRISTUP

⇒ UMJETNA NEURONSKA MREŽA:

- umjetna replika ljudskog mozga kojom se nastoj simulirat postupak učenja
- koristi se za:
  a) KLASIFIKACIJU
  b) PREDVIĐANJE
- dobra je u procjeni nelinearnost i jako je velike robusnosti

⇒ DVIJE FAZE RADA NN-a:

- učenje
- iskorištavanje (obrada podataka)

⇒ o učenju:

a) pojedinačno učenje
   - na temelju svakog učenja radimo korekcije

b) grupno učenje
   - na temelju epohe u jednoj iteraciji radimo korekcije

⇒ UČENJE: iterativan postupak predočavanja ulaznih primjera i očekivanih izlaza pri čemu dolazi do postupnog podešavanja težina veza neurona

⇒ znanje neuronske mreže je skup težina neurona

⇒ TIPOVI UČENJA:

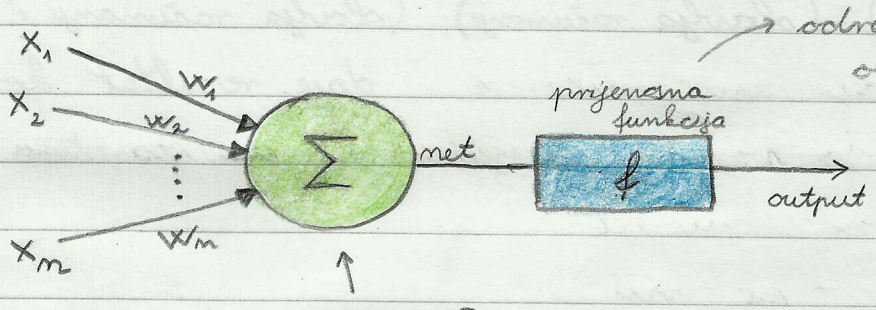
a) s učiteljem
b) bez učitelja
c) potporno učenje

⇒ SKUPOVI PRIMJERA ZA UČENJE:

↳ podjela 60%, 20%, 20% kao kod stabla odluke

↳ to je potrebno kako bi spriječili **pretreniranost** mreže na danim primjerima za učenje

⇒ MODEL NEURONA:



↳ prag paljenja: ako je suma manja od njega izlaz će biti nizak, a inače će biti visok (za TLU neuron)

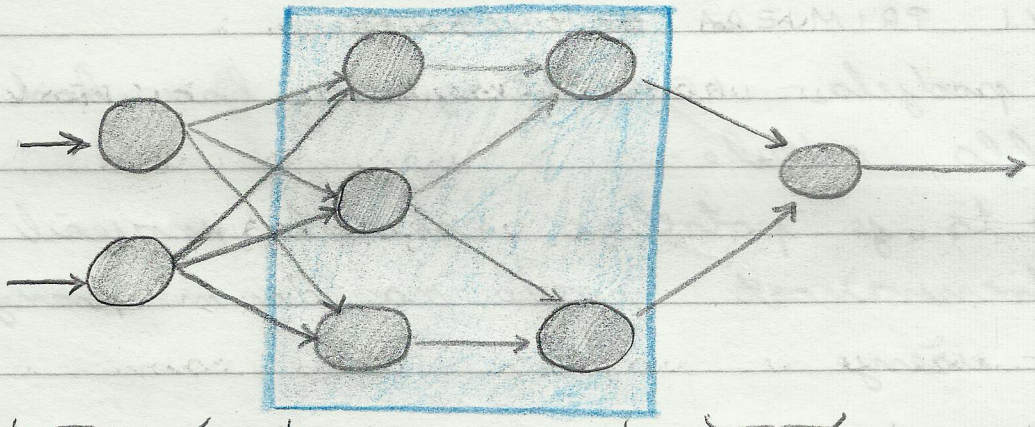
$$net = w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots + w_m x_m - w_0$$

$$net = \sum_{i=0}^{m} w_i x_i$$

⇒ time dobivamo skalarni umnožak (dobro jer brzo možemo računati množenjem matrica)

⇒ mi ćemo se baviti **slojevitom** arhitekturom neuronske mreže:

↳ mreža će biti **unaprijedna** - informacija se propagira samo prema naprijed



(samo prosljeđuje ulaze) (obavlja računanje) (obavlja računanje i daje rezultat korisniku)

↳ ovo je osnovna i najjednostavnija struktura mreže

⇒ **primjetimo sljedeću sponu:**

$$w_1 x_1 + w_2 x_2 + \ldots + w_m x_m + w_0 = 0$$

↳ ovo je jednadžba ravnine (hiperravnine u dimenziji višoj od 3D)

↳ stoga, postoji pravac koji dijeli tu ravninu na dva dijela i tako obavlja KLASIFIKACIJU uzoraka (aktivacijska funkcija je **step**)

↳ kažemo da je takav razred **linearno razdvojiv**

⇒ HEBB, 1949: „uči tako mijenjat jakost veza"

↳ Rosenblatt je 1958. ovu činjenicu iskoristio

⇒ PRAVILO PERCEPTRONA

BITNO?

1) ukoliko se uzorak klasificira ISPRAVNO:

↳ ne radi korekciju!

2) ukoliko se uzorak klasificira NEISPRAVNO:

↳ radi korekciju težina!

$$w_i(k+1) = w_i(k) + \eta(t-o)\cdot x_i(k)$$

3) ciklički uzimaj sve uzorke redom, a postupak raustavi kada su svi uzorci redom ispravno klasificirani

↳ oznake:

„k" ⇒ korak

„η" ⇒ stopa učenja

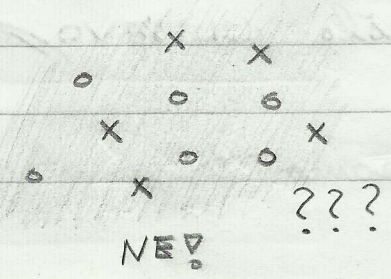
„w_i" ⇒ težina koju korigiramo

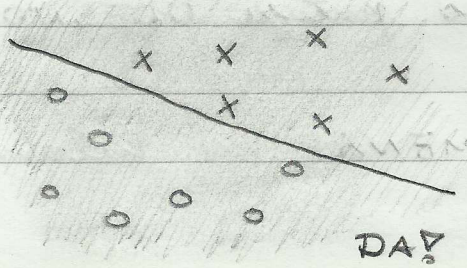
„t" ⇒ „true" (očekivana) vrijednost

„o" ⇒ dobiveni izlaz iz perceptrona

„x_i" ⇒ ulaz na težinu koju korigiramo

⇒ REZIME: perceptron ima TLU (step) funkciju, pa može riješiti samo linearno razdvojive probleme





BACKPROPAGATION

⇒ zbog problema TLU perceptrona da se primjenjuju samo na linearno razdvojive probleme, umjetna inteligencija sa konekcionističkim pristupom je pala u zaborav na čak tridesetak godina

↳ nedostajalo je:

- pametnije mijenjanje težina (onaj koji je više kriv za grešku se treba više izmijeniti)

⇒ hiperravnina greške predstavlja sve moguće verzije naše neuronske mreže

↳ to je zato jer svaka težina mreže daje jednu dimenziju hiperravnine greške

⇒ IDEJA: tehnikom gradijentnog spusta po hiperravnini greške se spuštamo od početne („random" početak) točke do globalnog minimuma greške (nadamo se)

⇒ gradijentno se spuštamo izmjenom prvih težina

⇒ podaci putuju UDESNO, a greške ULIJEVO

⇒ NAPOMENA: mi trebamo znati samo ideju algoritma

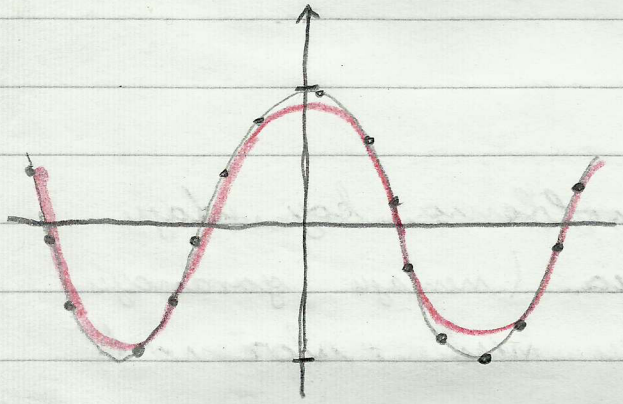
⇒ NAPOMENA:

↳ Veći broj slojeva mreže daje mreži veću ekspresivnost, ali i veću mogućnost prenaučenosti

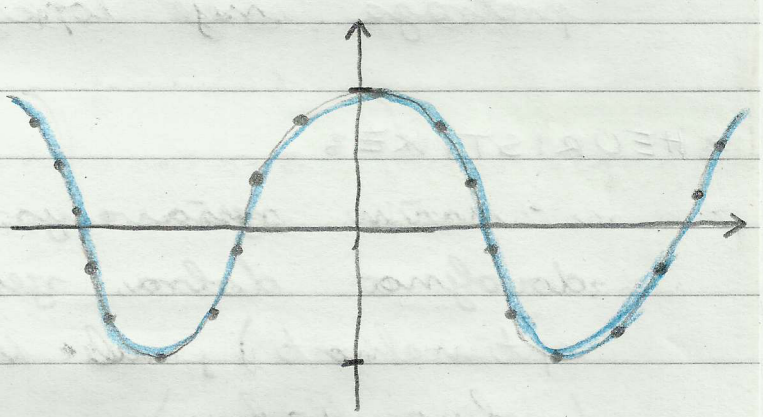
↳ ne znamo koja struktura mreže je optimalna!

⇒ NAPOMENA:

↳ ako imamo sigmoidalnu izlaznu funkciju, mreža teško generalizira rubne slučajeve



SIGMOIDALNA FUNKCIJA U IZLAZNOM SLOJU



LINEARNA FUNKCIJA U IZLAZNOM SLOJU

↳ ovo je rato jer sigmoidalna funkcija ide u krajnosti za „∞" i „-∞", a to se nikad ne događa

↳ ovo je bolje jer linearna izlazna funkcija samo prosljeđuje rezultate, pa još krajnosti nisu problem

EVOLUCIJSKO PROGRAMIRANJE

⇒ „ good connection of dumb parts yield smart results"

⇒ „ the xchole is greater than the sum of parts"

⇒ koriste se **genetski algoritmi**

⇒ PRIMJENA:

↳ optimiracijski problem gdje iscrpna pretraga nije opcija

⇒ HEURISTIKE:

- naći rješavanja problema koj daju dovolno dobra rješenja (nemaju garanciju optimalnosti), ali imaju nisku složenost (polinomijalnu)
- dijele se na:

a) **konstrukcyske** - grade rješenje segment po segment

b) **algoritmi lokalne pretrage** - kreću od početnog „random" rješenja i inkrementalno ga poboljšavaju

⇒ METAHEURISTIKE:

- skup algoritamskih koncepata koji koristimo na raznim problemima
- to su na primjer:
  - ↳ mravlji algoritmi
  - ↳ evolucijski algoritmi
  - ↳ simulirano kaljenje
  - ⋮

⇒ „NO FREE LUNCH", 1995.:

- „ako algoritam A nadmašuje algoritam B na nekim funkcijama cilja, tada mora postojati toliko drugih funkcija cilja gdje algoritam B nadmašuje algoritam A"
- drugim riječima, ne postoji algoritam koji je najbolji na sve probleme
- ovo je matematički dokazano!

O GENETSKIM ALGORITMIMA

⇒ ideja potiče još od Charlesa Darwina

⇒ PRETPOSTAVKE:

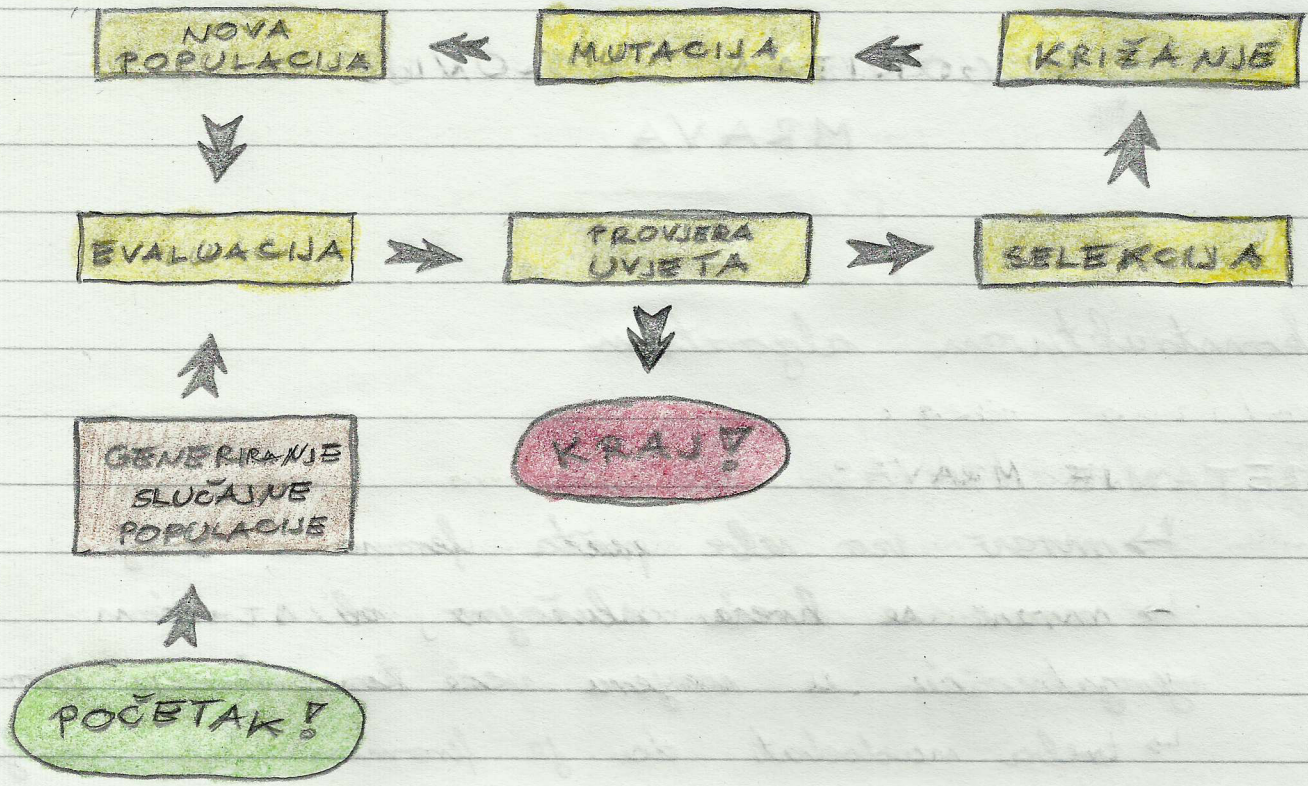
- plodnost vrsta - potomka je uvijek više nego što je potrebno
- veličina populacije je približno stalna
- dvije jedinke nastale križanjem gotovo nikad nisu identične

⇒ KAKO RADI GENETSKI ALGORITAM?

↳ radimo s populacijom kromosoma (jedan kromosom je jedno rješenje problema)

↳ svako rješenje ima dobrotu (fitness) koja se nastoji maksimizirati

↳ IDEJA: najbolji roditelji grade novu djecu križanjem, no treba osigurati da i ne-najbolji imaju šanse sudjelovati



⇒ **ULOGE OPERACIJA:**

a) SELEKCIJA ⇒ radi selekcijski pritisak te tako utječe na brzinu konvergencije
⇒ veći pritisak znači da smo jako strogi u nastojanju da najbolji roditelji imaju šansu za križanje

b) KRIŽANJE ⇒ utječe na pretraživanje okoline roditelja
⇒ selekcija + križanje jako sužavaju populaciju i tako daju konvergenciju

c) MUTACIJA ⇒ djeluje kao protuteža križanju i selekciji kako bi spriječila prerano konvergenciju

ALGORITAM KOLONIJE MRAVA

⇒ konstruktivan algoritam

⇒ KRETANJE MRAVA:

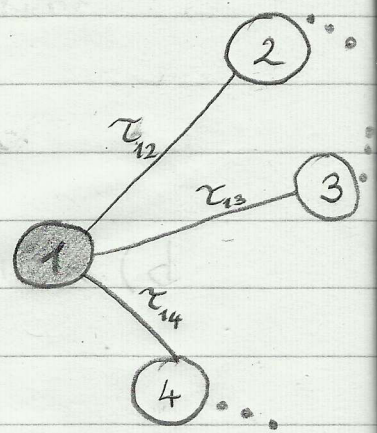
↳ mrav iza sebe pušta feromonski trag

↳ mrav se kreće slučajno, ali s većom vjerojatnošću u smjeru veće koncentracije feromona

↳ treba nadodati da je feromon lako hlapljiv i ne ostaje dugo na remlji

⇒ u računalnom sustavu imamo, ono predstavljamo GRAFOM, a kretanje po grafu je analogija liranja puta mrava kroz svijet ☺

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{t}}{\sum_{l \in N_i^k} \tau_{il}^{t}}, & \text{ako je } j \in N_i^k \\ 0, & \text{ako } j \notin N_i^k \end{cases}$$



↳ napomena: početno sve feromonske tragove stavimo na neku istu konstantu ($\tau_0 = const$)

⇒ *izvod navedene formule:*

• želimo da nam vjerojatnost bude proporcionalna količini feromonskih tragova:

$$p_{12} = k\,\tau_{12}$$
$$p_{13} = k\,\tau_{13}$$
$$p_{14} = k\,\tau_{14}$$

• pošto mrav može iz stanja 1 jedino u stanja 2, 3, 4 mora vrijediti:

$$p_{12} + p_{13} + p_{14} = 1$$

• sada raspisujemo dalje:

$$k\,\tau_{12} + k\,\tau_{13} + k\,\tau_{14} = 1$$

$$k = \frac{1}{\tau_{12} + \tau_{13} + \tau_{14}}$$

• vratimo se u početne vjerojatnosti:

$$p_{12} = \frac{\tau_{12}}{\tau_{12} + \tau_{13} + \tau_{14}}$$

$$p_{13} = \frac{\tau_{13}}{\tau_{12} + \tau_{13} + \tau_{14}}$$

$$p_{14} = \frac{\tau_{14}}{\tau_{12} + \tau_{13} + \tau_{14}}$$

⋮ „poopćenje"

$$p_{ij} = \frac{\tau_{ij}}{\sum_{l \in N_i} \tau_{il}}$$

⇒ dodajemo eksponent „α" radi manipulacije:

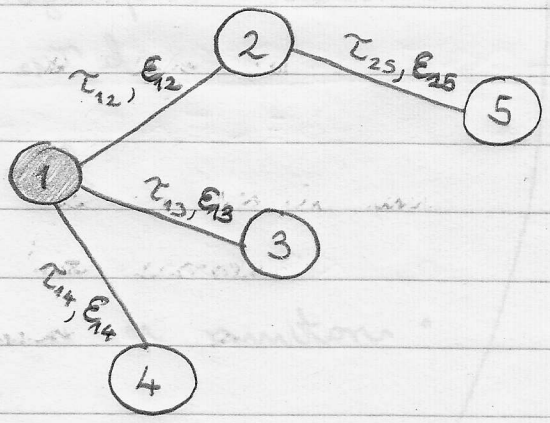
$$P_{ij} = \frac{\tau_{ij}^{\alpha}}{\sum_{l \in N_i} \tau_{il}^{\alpha}}$$

⇒ dodajemo „**decay**" (propadanje) kako bi emulirali propadanje / isparavanje feromona u prirodi

⇒ obično dozvolimo samo nekim najboljim rješenjima populacije da deponiraju feromonski trag

⇒ DODAVANJE HEURISTIKE:

↳ ideja je ugraditi iskustveno znanje da dalje poboljšamo rad algoritma



↳ heuristika će dati informaciju koja će utjecati na izbor puta

• na primjer, za TSP će biti dobra heuristika:

$$\varepsilon_{12} = \frac{1}{d_{12}}$$

„d_{12}" je udaljenost između gradova 1 i 2

BROJ MRAVA

⇒ NOVA FORMULA:

↳ inicijalizacija: $\tau_0 = \frac{m}{C^{nn}}$ ("nearest neighbour")

↳ vjerojatnost puta:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{l \in N_i^k} (\tau_{il}^{\alpha} \cdot \eta_{il}^{\beta})}, & \text{ako } j \in N_i^k \\ 0, & \text{ako } j \notin N_i^k \end{cases}$$

⇒ NAPOMENA:

↳ za $L=0$: isključivanje utjecaja feromonskih tragova te samo heuristika utječe na raspodjelu vjerojatnosti izbora puta

↳ za $\beta=0$: isključivanje heuristike, pa se vjerojatnost izbora putova računaju samo na temelju jakosti feromonskih tragova

FILOZOFSKI POGLED NA UMJETNU INTELIGENCIJU

⇒ „STRONG AI": bavi se pitanjem može li stroj imati svjestan um?

⇒ „WEAK AI": bavi se inženjerskim pitanjima te dizajnira računala tako da se naizgled ponaša inteligentno kako bi riješila neki problem

⇒ JOHN SEARLE:

↳ argument protiv jake umjetne inteligencije, funkcionalizma

↳ funkcionalizam → ideja da se stanja uma mogu ostvariti bez biološke podloge te da je sama bitna funkcionalnost dijelova da se ostvari cijelina

↳ misaoni eksperiment: KINESKA SOBA

VIDI SLAJDOVE!