Uvod

1. **Šta je Inteligencija a šta Veštačka inteligencija?**
   Inteligencija je sposobnost da se brzo pronađe adekvatno rešenje u velikom programskom prostoru.
   „Deo računarskih nauka koji se bavi projektovanjem i izgradnjom inteligentnih računarskih sistema, tj. sistema koji pokazuju karakteristike koje povezujemo sa inteligencijom u ljudskom rešavanje problema,..."
2. **Gruba podela definicija (sistemi koji razmišljaju/ponašaju se kao ljudi / racionalno)**

Inteligencija = sposobnost za rešavanje problema

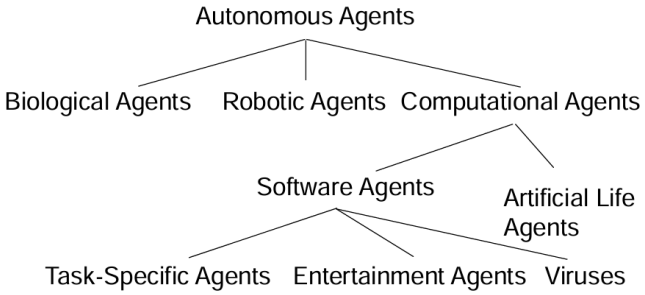
Sistemi koji razmišljaju kao ljudi (thinking humanly), napr. Cog	**Sistemi koji razmišljaju racionalno** (thinking rationally), napr. Theorem provers
Sistemi koji se ponašaju kao ljudi (acting humanly) napr. Turing's test	**Sistemi koji se ponašaju racionalno** (acting rationally) napr. Deep blue

Inteligencija = sposobnost da se ponašaju kao ljudi

3. **Turingov test**
   Da li mašine mogu da misle? Da li mašine mogu da se ponašaju inteligentno?
   Test za inteligentno ponašanje
   Turing je predvideo da ce do 2000 mašine imati 30% šanse da prođu test u trajanjuod 5 minuta
   Predvideo je sve argumente protiv VI u sledećih 50 godina
   Predložio je glavne komponente VI: znanje, zaključivanje, razumevanja jezika, učenje
   Problemi: ne može se reprodukovati, ne može se konstruisati, nije podložan matematičkoj analizi
   Osnovni nedostaci Turing-ovog testa:
   više je test sudije nego mašine, isključuje fizički/telesniaspekt
   Totalni Tjuringov test:
   Uključuje interakciju sa okruženjem, Prepoznavanje govora, Computer vision, Robotika
4. **Razlika izmeđi jake (Strong AI) i slabe (Weak AI) veštačke inteligecije**
   Slabe veštačke inteligencije su specijalizovane za određene zadatke i nemaju sposobnost opšteg razumevanja.
   Primeri uključuju prepoznavanje govora, prepoznavanje slika, preporučivanje proizvoda na osnovu pretrage, i slično.
   Sposobnost slabe veštačke inteligencije ograničena je na konkretne zadatke za koje je programirana.
   Sposobnost **jake veštačke inteligencije** obuhvata opšte razumevanje sveta, rešavanje različitih vrsta problema, učenje iz iskustava i prilagođavanje novim situacijama.
   Postizanje jake veštačke inteligencije predstavlja izazov jer zahteva razumevanje kompleksnih aspekata ljudske kognicije, uključujući apstraktno razmišljanje, svest, i druge visoke nivoe inteligencije.
5. **Osnovni koncepti veštačke inteligencije i poređenje sa konvencionalnim sistemima**
   •Simboličko umesto numeričkog izračunavanja.
   •Nealgoritamski pristup rešavanju problema.
   •Zaključivanje zasnovano na znanju.
   •Primenljivost kod loše struktuiranih problema i podataka

Inteligentni agenti

1. **Šta je agent (inteligentni agent)?**
   Agent je entitet koji saznajeo svom okruženju i izvršava razne akcije. Agent je bilo koji sistem (sw, hw ili kombinacija), koji je sposoban da prima informacije iz okruženja preko senzorai da deluje na okruženje preko efektora.
   Inteligentni agent (ili jednostavno agent) na Internetu, je program koji uzima informacije ili izvršava neke druge servise bez vašeg prisustva odnosno intervencije, na osnovu prethodno definisanog rasporeda.
   Inteligentni agent je računarski sistem koji može da izvrši autonomno fleksibilnu akciju u nekom okruženju za dostizanje definisanog cilja Pod fleksibilnom akcijom podrazumevamo:
   - Reaktivnost–postoji interakcijasa okruženjem i sposoban je da odgovori na promene koje se dešavaju u njemu
   - Pro-aktivnost–sposoban je za ponašanje definisano ciljem,
   - Socijalni aspekt –sposoban je da interagujesa drugim agentima (i ljudima)
2. **Klasifikacija agenata**



3. **Mera performansi i idealni racionalni agent**
   Mera performansi: objektivni kriterijum kojim se meri uspešnost agenta; određena je ciljevima koji su definisani za agenta
   Primer: mera performasi zaVacuum-cleaner agenta može biti očišćena količina smeća, potrošeno vreme za čišćenje, količina energije, količina generisane buke, ...
   Merenje performansi (zavisi od okruženja)
   - Eksterno merenje performansi (kako agent deluje na okolinu)
   - Samo-evaluacija–agent je sposoban da sam vrši procenu svojih akcija
   - kontinualno, periodično ili jedna evaluacija

   Specifikacija mera performansi:

   - mere performansi su eksterne
   - okruženje obezbeđuje povratnu informaciju agentu u obliku funkcija za preslikavanje iz mogućih stanja u realne brojeve
   - povratne informacije se mogu obezbediti nakon svake akcije, periodično ili na kraju delovanja agenta

   Idealni racionalni agent: Za svaku sekvencu opažanja, idealni racionalni agent izvršava akciju koja maksimizira performanse, na osnovu ulaza i ugrađenog znanja koje agent poseduje. Problem: perfektno racionalno ponašanje je limitirano hardverskim mogućnostima

4. Osnovni koncepti

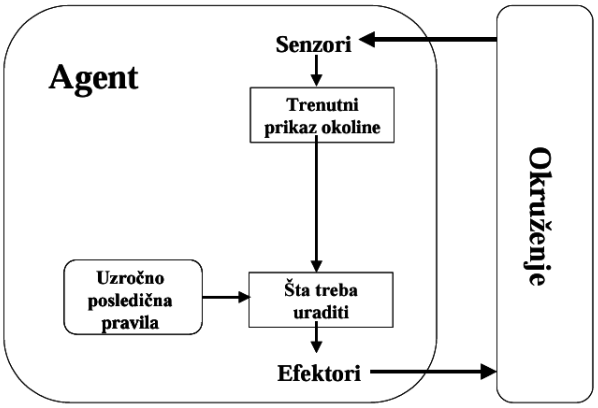
- Opažanja(ulaz)/ Senzori (ulazni uredjaji)
- Akcije(izlaz) / Efektori (izlazni uredjaji)
- Cilj / Mera performansi
- Okruženje

Percepts,
Actions,
Goals,
Environment

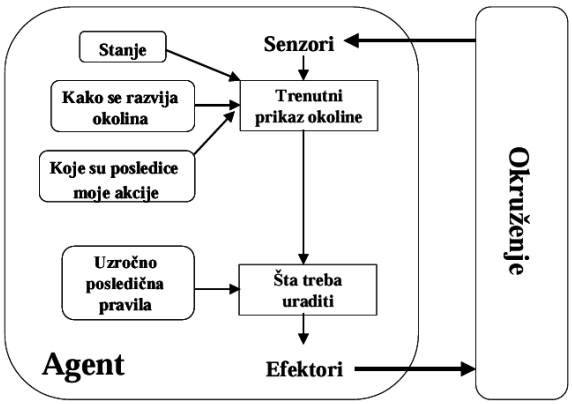
5. Arhitekture inteligentnih agenata

0.(Table-driven agenti)

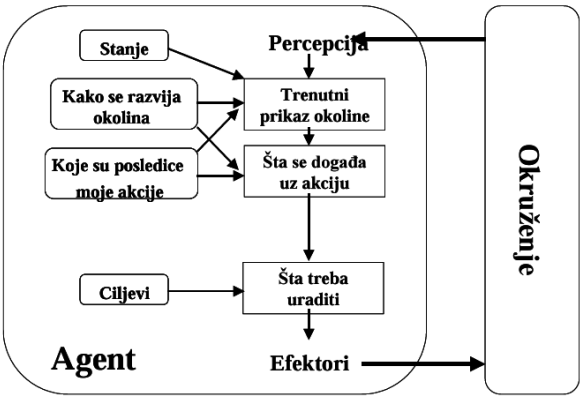
1. Jednostavniagent baziran na refleksima



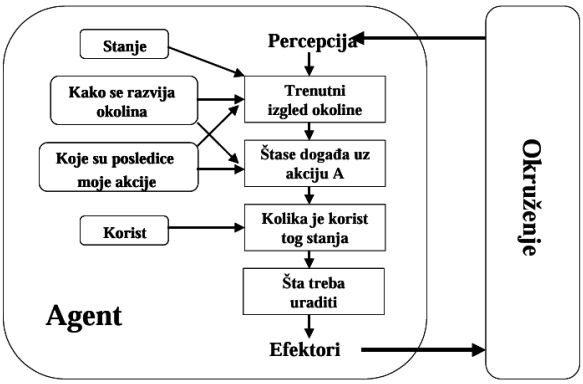
2. Refleks agenti koji pamte stanja



3. Agenti bazirani na cilju



4. Agenti bazirani na korisnosti



6. Tipovi okruženja

Dostupnost: Potpuno ili delimično dostupna:

- da li agent ima podatke o celokupnom okruženju ili samo o nekom delu okruženja.

Determinisanost: Deterministička ili stohastička:

- determisnitičko, ako sledeće stanje okruženja kompletno zavisi od tekućeg stanja i izabrane akcije od strane agenta,
- inače stohastičko.

Diskretnost:Diskretnaili kontinualna:

- da li su stanja(i akcije) diskretna ili kontinualna.
- Šah je diskretno, vožnja taksija je kontinualno okruženje.
- Kod diskretnih okruženja postoji konačan broj jasno definisanih koncepata i akcija.

Epizodnost: Bazirana na epizodama ili sekvencijalna:

- u okruženju baziranom na epizodama iskustvo agenta je podeljeno na epizode.

- Kvalitet akcije zavisi od samo od epizode; naredne epizode ne zavise od prethodnih.
- Kod sekvencijalnih to nije slučaj, tj. “epizode” su međusobno zavisne.

Statička ili dinamička:

- okruženje se ne menja (ili menja u drugom slučaju) dok agent “razmišlja”.

Jedan ili više agenata u okruženju:

- konkurentna(suparnička) ili
- komparativna(uporediva) okruženja (u originalu: competitive ilicomparative).

Definišu da li agenti međusobno sarađuju ili se nadmeću; u oba slučaja postoji komunikacija između agenata.

Traženje

1. **Formulacija problema, prostor stanja**

   Prostor stanjaje skup svih stanja koji se mogu dostići iz polaznog stanja.
   Prostor stanja ima formu grafa u kome su čvorovi stanja, a potezi akcije odnosno funkcije sledbenika.
   Put u prostoru stanja je sekvenca stanja povezanih sekvencom akcija. Rešenje problema je jedan put u grafu/prostoru stanja.

2. **Komponente za opis problema**

   Komponente:

   - Prostor stanja(definisan eksplicitno ili implicitno)
   - Inicijalno stanje
   - Ciljno stanje (ili uslovi koje treba zadovoljiti)
   - Dostupne/dozvoljene akcije (operatori koji menjaju stanja)
   - Ograničenja(napr.cena)
   - Elementikoji se odnose na znanje o domenu koje je relevantno za konkretan problem

   Tip rešenja:

   - Sekvenca operatora ili ciljno stanje
   - Neko, optimalno (potrebna definicija cene puta), sva

3. **Rešavanje problema kao traženje**

   - Stanje –diskretno stanje«sveta»
   - Prostor stanja:-skup svih stanja problema
   - Početno stanje
   - Ciljno stanje(ili stanja!!)
   - Operatori: -prelaz iz stanja u stanje; funkcije koje odredjuju sledeće stanje na osnovu tekućeg stanja (ili NIL)
   - Sledbenici (potomci): sva stanja u koja se može preći iz tekućeg stanja
   - Traženje sledbenika:primena svih operatora na tekuće stanje da bi se dobili potomci
   - Uslovi/Test za ciljno stanje: predikat koji vraća T ili F za zadato stanje
   - Cena puta(opciono): suma cena individualnih operatora
   - Heuristika –informacija o tome zbog čega je izbor nekog čvora (stanja) bolji od izbora ostalih
   - Rešavanje problema –odrediti niz operatora koji prevodi sistem iz početnog stanja u ciljno stanje; put do cilja (rešenje problema) je niz čvorova u grafu koje treba obići da bi se došlo do cilja.

4. Opšti algoritam traženja i stablo traženja

□ Polazno stanje □ Operatori □ Sledbenici □ Krajnje stanje (cilj) □ Test □ Cena puta	**Obilazak prostora stanja** •Pronalazi jednu putanju (ili sve putanje) od početnog do ciljnog stanja • Izračunava cenu primene niza operatora (put) koji vodi od početnog stanja to krajnjeg stanja • Neki algoritmi nalaze put sa minimalnom cenom
• Osnovna ideja: offline, simulirani obilazak prostora stanja generisanjem sledbenika za već posećena stanja • Stablo traženja je efikasan način da se predstavi kako algoritam traženja ispituje prostor traženja. Dinamički se kreira, počev od početnog stanja • Važno: prostor traženja i stablo traženja se razlikuju!	

5. Traženje po širini / Traženje po dubini i modifikacije

Algoritam traženja po širini (Breadth-first search)

Čvorovi se obilaze s leva udesno

Da bi se izbegle petlje, ne generisu se sledbenici koji su jednaki roditelju tekuceg čvora (postoje i drugi načini da se izbegnu petlje)

Nalazi najplići cilj

Kompletan i optimalan ako je cena puta neopadajuca funkcija dubine čvora

Implementacija: **red(FIFO)**

1. Formirati listu čvorova koja inicijalno sadrži samo startni čvor.
2. Dok se lista čvorova ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element liste ciljni čvor
   a) Ako je prvi element liste ciljni čvor, ne raditi ništa.
   b) Ako prvi element liste nije ciljni čvor, ukloniti ga iz liste i dodati sve njegove sledbenike iz stabla pretrage (ako ih ima i ako nisu već posećeni) na kraj liste.
3. Ako je pronađen ciljni čvor, pretraga je uspešno završena; u suprotnom pretraga je neuspešna.

Kompletnost? Da(ako jebkonačno)

Vreme?1+b+b2+b3+… +bd+ b(bd-1) = O(bd+1) Prostor?O(bd+1)(čuva svaki čvor u memoriji)

Optimalnost? Da(akoje cena= 1 po koraku)

Prostor–problem (u odnosu na vreme)

Prednosti:

- Uvek nalazi cilj (rešenje)
- Dobar za plitka stabla

Nedostaci:

- Veliki zahtevi za memorijom
- Može biti spor za stabla sa velikom dubinom

Modifikacija traženja po širini:

- Iz reda se bira čvor sa najnižom cenom g(n)
- Nalazi optimalno rešenje ako cena puta nikada ne opada sa dubinom
- Implementacija: red(FIFO)uređen po ceni puta

Algoritam traženje po dubini (Depth-first search)
Čvorovi se obilaze s leva udesno
Implementacija: **magacin(LIFO)**

Kompletnost?Ne: ne nalazi rešenje u beskonačnim prostorima stanja (infinite-depth spaces), kod stanja sa petljama

- Modifikuj ili izbegavaj stanja koja se ponavljaju!
- kompletan sa konačnim stanjima

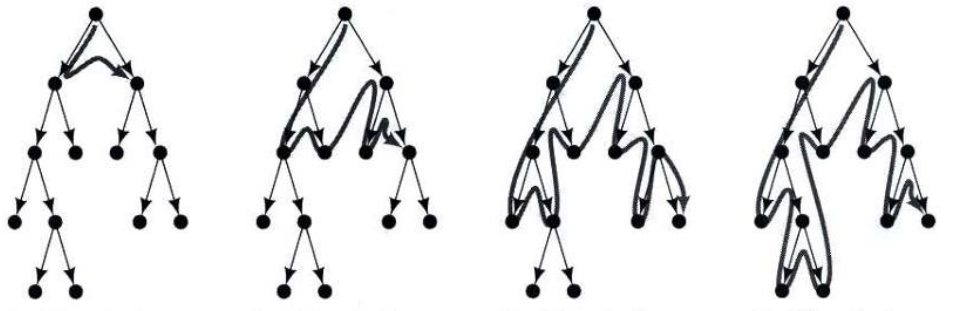
Vreme?O(bm): jako loše ako jemmnogo veće od d

- ali ako plitko, može da bude značajno brži od breadth-first

Prostor?O(bm), tj, linearno!
Optimalnost?Ne

Ne generišu se oni sledbenici koji se većpojavljuju u putu od korena do tekućeg čvora (da bi se izbegle petlje)
Mali zahtevi za memorijom
Ako postoje veoma dugački (ili neograničeni) putevi, ne garantuje rešenje
Nije kompletno, ni optimalno
Depth-limited search(DFS sa ograničenjem dubine traženja)
(čvor je na zadatoj dubini ili nema sledbenike)

- Kompletnost?Da
- Vreme?(d+1)b0+ d b1+ (d-1)b2+ … + bd= O(bd)
- Prostor?O(bd) Optimalnost?Da, ako je cena puta = 1



Stages in Iterative-Deepening Search

6. Heuristika i i informisano traženje

Heuristika: Omogućava da se izabere čvor koji više obećava od ostalih
Prihvatljiva heuristika: predvidjena cena nikada nije veća od stvarne

- h(n) –procenjena cena puta od tekućeg čvora do cilja
- h(G)=0za Ciljni čvor
- h(n)>0 za ostale čvorove

7. **Opšti algoritam informisanog traženja**
   Greedy best-first traženje
   Evaluaciona funkcijaf(n) = h(n) (heuristika) = procenjena cena on ndo cilja
   Greedy best-first traženjevrši ekspanziju čvora koji izgledada je najbliži cilju
   Primer,h(n)= pravolinjsko rastojanje od ndo Bucharest-a
   Kompletan?Ne–može da se zaglavi u petljama, napr, Iasi Neamt ->Iasi ->Neamt ->
   Vreme?O(bm), ali sa dobrom heutistikom može da se drastično poboljša
   Prostor?O(bm) –drži sve čvorove u memoriji
   Optimalan?Ne
8. **Algoritam prvi-najbolji**
   - Kompletan, nije optimalan generalno
   - Treba odrediti h(n)
   - Ideja: koristitievaluacionufunkcijuf(n) za svaki čvor
     - f(n) obezbeđuje procenjenu totalnu cenuputa.
     - Čvor za ekspanziju se bira na osnovu minimalne vrednosti za h(n)
   - Implementacija: Urediti čvorove po vrednosti cene puta –sortirani red.
   - Specijalni slučajevi:
     - Greedy best-first traženje–ekspanzija čvora koji izgledada je najbliži cilju
     - A*traženje

Implementacija: sortirani red

1. Formirati listu čvorova koja inicijalno sadrži samo startni čvor.
2. Dok se lista čvorova ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element liste ciljni čvor
   a) Ako je prvi element liste ciljni čvor, ne raditi ništa.
   b) Ako prvi element liste nije ciljni čvor, ukloniti ga iz liste i dodati njegove sledbenike iz stabla pretrage (ako ih ima i ako nisu već posećeni) u listu. Celokupnu listu sortirati po rastućim vrednostima heurističkih funkcija čvorova.
3. Ako je pronađen ciljni čvor, pretraga je uspešno završena; u suprotnom pretraga je neuspešna.

9. **A* algoritam**
   Izbegava puteve koji su već skupi
   Kombinuje stvarnu cenu putanje g(n)sa predvidjenom cenom h(n)
   Ukupna cena f(n)=g(n) + h(n)
   - g(n) = cena dostizanjan
   - h(n)= procenjena cena od ndo cilja
   - f(n) = procenjena ukupna cena puta od ndo cilja
   - Best First search: f(n)=h(n)

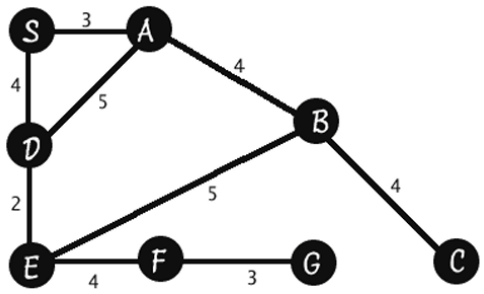
   Implementacija:kao kod Best-first, svi čvorovi u redu se sortiraju, u ovom slučaju po vrednosti f(n)

   Kompletan?Da (osim ako nema beskonačno mnogo čvorovasaf ≤f(G))
   Vreme?Eksponencijalno, b^d
   Prostor?Čuva sve čvorove u memoriji
   Optimalan?Da

10. Algoritam grananja i ograničavanja

Branch and Bound-optimalni algoritmi trazenja

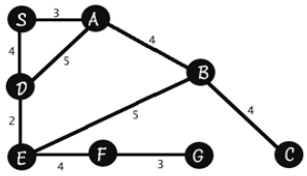
Implementacija: sortiranje pomoćne strukture, stvarna cena putanje g(n)



S(rastojanje= 0) => rastojanje se odnosi na dužinu puta napr do čvora, a ne na procenjeno rastojanj etog čvora do cilja(kao kod Hill Climbing)
Dodati: S->A (rastojanje= 3), S->D (rastojanje= 4) , Sortiranje: S->A(rastojanje= 3), S->D (rastojanje= 4) □ Dodati: S->A->B(3 + 4 (rastojanje od A do B) = 7), S->A->D(8), Sortiranje: S->D(4), S->A->B(7), S->A->D(8)

Rad algoritma B&B:

- S->D->A(9), S->D->E(6), S->A->B(7), S->A->D(8),
  Sortiranje: **S->D->E(6)**, S->A->B(7), S->A->D(8), S->D->A(9)
- S->D->E->B(11), S->D->E->F(10), S->A->B(7), S->A->D(8), S->D->A(9),
  Sortiranje: **S->A->B(7)**, S->A->D(8), S->D->A(9), S->D->E->F(10), S->D->E->B(11)
- S->A->B->C(11), S->A->B->E(12), S->A->D(8), S->D->A(9), S->D->E->F(10), S->D->E->B(11),
  Sortiranje: **S->A->D(8)**, S->D->A(9), S->D->E->F(10), S->D->E->B(11), S->A->B->C(11), S->A->B->E(12),



Nadalje: [Najkraći put] => [lista of puteva])

- S->A->D(8) => S->A->D->E(10)
- S->D->A(9) => S->D->A->B(13)
- S->D->E->F(10) => **S->D->E->F->G(13)** cilj je dostignut sa cenom 13!

Razvoj preostalih puteva, i traženje da li postoji neki sa cenom manjom od 13.

- S->A->D->E(10) ima manju cenu u narednom koraku, razvijanjem dobijamo:
- S->A->D->E->B(15), S->A->D->E->F(14), veći su od 13, ignorišemo ih, **itd**.

11. Local search algoritmi i razlike u odnosu na klasične algoritme traženja

Kod mnogih optimizacionih problema, put do cilja je irelevantan;
Kod takvih problema samo ciljno stanje je rešenje
Prostor stanja= skup„kompletnih" konfiguracija (rešenja)
Pronađi konfiguraciju koja zadovoljava ograničenja
U takvim slučajevima, koristimo local search algoritme
Čuva se jedno, „tekuće" stanje, pokušava se da se ono poboljša
Primer problema za Local Search: n-queens
Postavi nkraljicana tabli nxntako da se ne napadaju (nema dve kraljice u istoj vrsti, koloni ili dijagonali); Pomeri kraljicu da smanjiš broj konflikta

12. Metod Hill- climbing (planinarenja)

"Like climbing Everest in thick fog with amnesia„

- uvek ide prema cilju
- koristi heuristiku za nalaženje pravca koji vodi najbrže (najbliže) cilju
- analogija sa planinarenjem, noću, ako je logor na vrhu brda

Prednosti:

- Smanjuje broj čvorova koje treba obići
- Daje efikasnije ili isto rešenje kao DFS

Nedostaci:

- "Lažni vrhovi " u kojima dolazi do izrazitih povrataka
- Problem ako svi čvorovi izgledaju kao podjednako dobri
- Neophodna heuristika
- Može se desiti da se ne nađe rešenje iako ono postoji

Implementacija Hill-climbing

Napr. DFS + rastojananja svakog čvora do cilja (kao h(n))

- Formirati listu čvorova koja inicijalno sadrži samo startni čvor.
- Dok se lista čvorova ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element liste ciljni čvor
  - Ako je prvi element liste ciljni čvor, ne raditi ništa.
  - Ako prvi element liste nije ciljni čvor, ukloniti ga iz liste i sortirati njegove sledbenikeiz stabla pretrage (ako ih ima i ako nisu već posećeni) po rastućim vrednostima heurističke funkcije čvora (rastojanje od ciljnog čvora). Zatim te sledbenike dodati na početak liste tako da prvi element liste bude sledbenik sa najmanjom vrednošću heurističke funkcije.
- Ako je pronađen ciljni čvor, pretraga je uspešno završena; u suprotnom pretraga je neuspešna.

Hill-climbing varijante

Stochastic hill-climbing

- Slučajna selekcija nekog od „uphill“stanja.
- Verovatnoća selekcije može da varira od strmosti „uphill“stanja.

First-choice hill-climbing

- Stohastic hill climbing sa generisanjem sledbenika slučajno sve dok se ne nađe bolji
- Korisno kada ima veliki broj sledbenika

Random-restart hill-climbing

- Pokušava da izbegne zaglavljivanje u lokalnom maksimumu.
- Više varijanti restartovanja

13. Simulirano kaljenje

Kaljenje je termički proces obrade (metala) u dva koraka:

- Povećaj temperaturu na maksimum (na kojoj se metal topi).
- Smanjuj pažljivo temperaturu sve dok se ne postigne minimum energije (čestice se međusobno organizuju).

Algoritam na osnovu ovog procesa, simulira se Metropolis algoritmom, koji se zasniva na Monte Carlo tehnikama.

Analogija:

- Rešenje problema je ekivalentno sanju fizičkog sistema.
- Cena rešenja je ekvivalentna “energiji” stanja.

Ideja: izbeći lokalni minimum tako što se dozvoli izbor „loših" stanja ali sa postepenim smanjivanjem njihove učestanosti

- Neka su 3 stanja moguća, sa promenama *objective function* d1 = -0.1, d2 = 0.5, d3 = -5. (neka je T = 1).
- Izaberi stanje slučajno:
  - Ako je to d2, izaberi ga.
  - Ako je d1 ili d3, računaj verovatnoću = exp(d/T)
    - stanje 1: prob1 = exp(-0.1) = 0.9, odnosno, 90%
    - stanje 3: prob3 = exp(-5) = 0.05, tj., 5%
- Parametar T = "temperatura"
  - high T => verovatnoća za "locally bad" stanje je veća
  - low T => verovatnoća za "locally bad" stanje je manja
  - T se smanjuje u svakom koraku algoritma – postoji „raspored" promene

Osobine Simulated annealing traženja
Može se dokazati: ako se T smanjuje dovoljno sporo, tada će traženje simuliranim kaljenjem pronaći globalni optimum sa verovatnoćom koja se približava 1

14. Algoritmi za Igre – definisanje preko traženja, funkcije korisnosti

Funkcija korisnosti Koristi se kao heuristička funkcija s tom razlikom što se preko nje vrši evaluacija čvora (koji se odnosi na potez nekog igrača) u smislu procene koliko je on koristan za svakog od igrača
Pozitivne vrednosti indiciraju stanja koja obezbeđuju prednost za Max
Negativne vrednosti indiciraju stanja koja obezbeđuju prednost za Min
Procenjuju vrednost stanja na osnovu njegovih osobina–daje korisnost nekog stanja igre (utility(State):u(s))

15. MinMax algoritam – postavke i varijante MinMax algoritma

- Generiše **kompletno stablo traženja**
  - Na svakom nivou jedan od igraca povlači potez
  - Traženje po stablu mogućih poteza sa ciljem da se nađe potez koji prouzrokuje najbolji rezultat
- Analiza celokupnog stabla daje optimalne poteze (za oba igrača –pretpostavka da i protivnik igra optimalno!)
- Depth First Search(DFS)algoritam
- Generalno nije izvodljivo–stablo može biti preveliko!

Postavke za Minimax
Inicijalno stanje : Pozicija tj stanje na tabli, Ko je na potezu
Operatori: Legalni potezi igrača
Test : Određuje da li je stanje ciljno
Funkcija korisnosti (zavisi od igre)
Implementacija: Dva agenta
MAX

- Pokušava da maksimizujerezultat funkcije korisnosti
- Pobednička strategija, ako, na MIN potez po redu, pobeda je dostižna za MAX za svemoguće poteze MIN

MIN

- Pokušava da minimizujerezultat funkcije korisnosti
- Pobednička strategija, ako, na MAX potez po redu, pobeda je dostižna za MIN, za svemoguće poteze MAX

16. Alfa-Beta odsecanje

Cilj algoritma: smanjiti stablo traženja
Odseca grane koje ne obećavaju
Osnovna ideja:

- Izbegavaju se podstabla koja nemaju efekat na rezultat
- Pamti se vrednost najboljegpoteza do sada
  - α–najbolja vrednost za igraca Max. Koristi se u MIN čvorovima, i dodeljuje u MAX čvorovima
  - β–najbolja vrednost za igraca Min. Koristi se u MAX čvorovima i dodeljuje u MIN čvorovima

Kada Max ispituje moguće akcije, ako je bilo koja od njih veća od β(sto je gore po Min), onda može da se prestane sa traženjem (podrazumeva se da Min nece odigrati potez koji nije dobar).

MAX (ne na nivou 0)

- Ako je nađeno podstablo sa vrednošću kkoja je veća od vrednosti β, nema potrebe za njegovom daljom pretragom
  - MAX može da ima potez koji je dobar najmanje k, tako da MIN nikad neće da izabere taj put!

MIN

- Ako je nađeno podstablo sa vrednošću k koja je manja od vrednosti α, nema potrebe za njegovom daljom pretragom
  - MIN može da ima potez koji je dobar najmanje k, tako da MAX nikad neće da izabere taj put!

Osobine α-β

- Odsecanje ne utiče na konačni rezultat.
- Dobar redosled poteza poboljšava efikasnost odsecanja (pogledati prethodni primer – poslednje odsecanje)
- Sa"perfektnim redosledom" vremenska kompleksnost= O(bm/2) ⍰duplira dubinu traženja

Problem zadovoljenja ograničenja (Constraint Satisfaction Problem – CSP)

1. Definicija CSP

- Konačan skup promenljivih V1, V2, …, Vn
  - (Ne-prazni) domeni mogućih vrednosti za svaku promenljivu: DV1, DV2, … DVn
- Konačan skup ograničenja:C1, C2, …, Cm
- Svako ograničenje Ciograničava vrednosti koje promenljive mogu da uzimaju,
  - Primer, V1 ≠ V2
- Stanje se definiše kao dodelavrednosti nekim promenljivima.
- Konzistentna dodela (Consistent assignment): Dodela vrednosti ne narušava ograničenja
- Dodela je kompletnakada je su uključene sve promenljive.
- Rešenje za CSP je kompletna dodela koja zadovoljava sva ograničenja.
- Neki CSPzahtevaju rešenje koje maksimizuje funkciju cilja.
- Primeri primene CSP:
  - Scheduling(Airline schedules)
  - Kriptografija
  - Computer vision -> image interpretation

2. **Backtracking traženje**
   - Specijalna osobina CSP: dodele su komutativne => redosled dodele vrednosti nije od važnosti.
   - Treba razmatrati dodelu samo jednoj promenljivoj u svakom čvoru
   - Depth-first search za CSP sa single-variable dodelama se zove Backtracking search
     - Backtracking search je osnovni neinformisani algoritam za CSP
     - Može da reši n-queenszan ≈ 25
   - Bolje stablo traženja: Najpre uredi promenljive, nakon toga im dodeljuje vrednosti jedna-po-jedna.
   - U osnovi je DFS
   - Bira vrednosti za jednu promenljivu i vraća se kada za promenljiva nema legalnih vrednosti za dodelu
   - Neinformisani algoritam
   - Generalno nema dobre performanse
3. **Povećanje efikasnosti CSP**

Koriste se metode koje su generalne namene, napr.:
- Koja promenljiva je sledeća za dodelu?
- U kom redosledu treba uzimati vrednosti za dodelu?
- Da li se neizbežni neuspeh može detektovati ranije?
- Možemo li da iskoristimo strukturu problema?

4. **Minimum remaining values (MRV) heuristika**

   Izaberi promenljivu koja ima najmanje legalnih vrednosti(Most constrained variable)
   Primer: Minimum remaining values (MRV) heuristika
   Cilj heuristike: Izaberi promenljivu koja će prouzrokovati grešku što ranije, omogućavajući odsecanje kod stabla traženja.

5. **Degree Heuristic**

   Izaberi promenljivu s najviše ograničenjau odnosu na preostale promenljive (najviše potega u grafu)

6. **Least Constraining Value Heuristic (LCV)**

   Za promenljivu, izaberi najmanje ograničavajuću vrednost: Ona vrednost koja u preostalim promenljivima najmanje smanjuje broj vrednosti
   Ostavlja maksimalnu fleksibilnost za rešenje.

U svim slučajevima cilj nam je da:
- izaberemo granu koja najviše obećava,
- ali takođe želimo da detektujemo neuspeh što ranije.

MRV+DH: promenljiva koja najverovatnije prouzrokuje neuspeh najpre dobija vrednost.
LCV: pokušava da izbegne neuspeh dodelom vrednosti koje ostavljaju najviše fleksibilnosti za preostale promenljive.

7. **Forward Checking (FC)**

   Ideja:
   Pratite preostale legalne vrednosti za nedodeljene promenljive koje su povezane sa tekućom promenljivom.
   Prekinite traženje kad bilo koja promenljiva nema legalne vrednosti za dodelu.

8. **Propagacija ograničenja, konzistentnost potega**

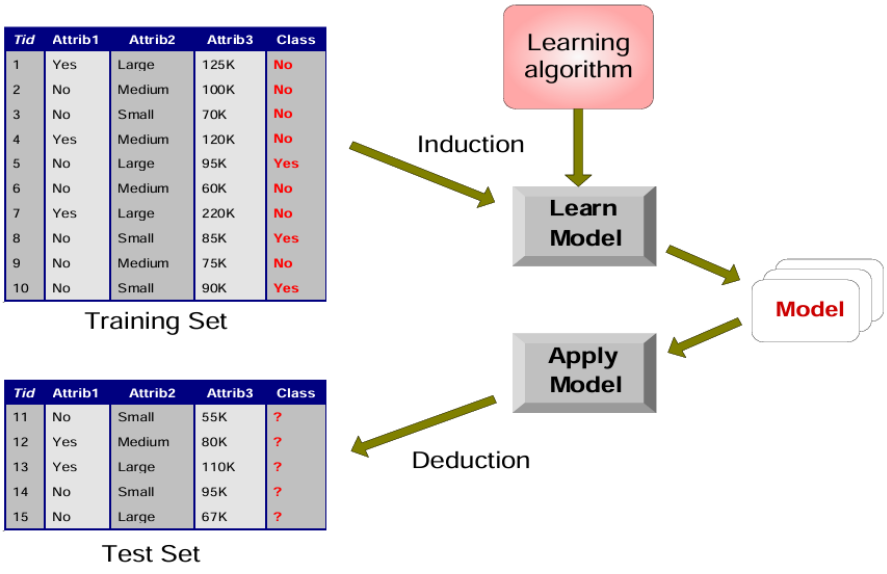
Propagacija ograničenja (CP) propagira konzistentnost potega kroz graf.
Tehnike kao što su CP i FC eliminišu deo prostora traženja
CP ide dalje u odnosu naFC pošto više puta lokalno izvršava ograničenja
Arc-consistency(AC) ili konzistentnost potega, je sistematska procedura (postupak) za propagaciju ograničenja
X ->Yje konzistentanakko zasvakuvrednostx iz X postoji neko dozvoljeno y
Ako X gubi vrednost, susede od X treba ponovo proveriti: dolazni potezi mogu postati nekonzistentni ponovo (odlazni potezi ostaju konzistentni).
Propagacioni algoritam–kao slanje poruka susedima u grafu! Kako možemo da planiramo takve poruke?
Svaki put kada se domen promenljive menja, sve dolazne poruke se moraju ponovo slati. Ponavljati sve dok nema poruka koje menjaju bilo koji domen.
Prazni domen znači da nema mogućeg rešenja povratak iz te grane!
Forward checking je jednostavno slanje poruka promenljivoj koja je upravo dobila svoju vrednost.
Praktično prvi korak iz arc-consistency.

Mašinsko učenje – Sabla odluke

1. **Inteligentni agent koji može da uči i problem klasifikacije**

Inteligentni agent koji uči može da:
Unapređuje svoje performanse na osnovu iskustva E koje stiče obavljajući neki zadatak T u svom okruženju, u odnosu na meru performansi P.

Problem klasifikacije



2. **Algoritam učenja stablom odluke**

(Decisiontree learning algorithm)-uspešno se primenjuje kod ekspertnih sistema za prikupljanje znanja.
Osnovni zadatak kod takvih sistema:

- korišćenjem induktivnih metoda, za zadate vrednosti atributa nepoznatog objekta, odrediti odgovarajuću klasifikaciju na osnovu pravila Stabla odluke.

Stablo odluke predstavlja struktura tipa stabla gde:

- Unutrašnji čvorov iodgovaraju atributima uzoraka i predstavljaju izbor između više alternativa,
- (grane u stablu odgovaraju vrednostima određenog atributa)
- Listovi predstavljaju odluke, odnosno klase kojima pripadaju uzorci sa vrednostima atributa definisanim putem do korena

3. Kreiranje stabla odluke – opšti postupak

Dve faze:

- Faza izgradnje stabla (topdown)
  - Na početku, svi primeri su u korenu stabla Rekurzivno se vrši particionisanje primera odabirom po jednog atributa
- Faza odsecanja stabla (bottomup)
  - Uklanjanje podstabala ili grana u cilju unapređenja tačnosti modela

"Pohlepna" strategija

- Deli primere na osnovu testa na neki atribut, tako da podela bude optimalna po određenom kriterijumu
- Algoritam bira najbolji atribut u datom trenutku i nikada se ne vraća da ponovo razmotri napravljene izbore

Cilj je da stablo bude što manje, tjda se što pre dođe do odluke

4. Kreiranje stabla odluke - kako odrediti atribut koji je na redu

Kriterijumi za podelu zavise od:

Tipa atributa

- Diskretni (kategorički) -kriterijum u formi A Є S', gde je S' podskup svih mogućih vrednosti atributa A
  - Nominalni –redosled nije bitan (plavo, crveno, žuto)
  - Ordinalni –redosled je bitan (nisko, srednje, visoko)
- Kontinualni (kriterijum u formi A <= v, gde je v vrednost koja pripada rangu iz kog atribut A može uzimati vrednosti)

Načina podele

- Binarna podela
- Višestruka podela

Podela na osnovu **diskretnih atributa**

- Višestruka podela:broj grana jednak broju različitih vrednosti atributa
- Binarna podela: podela vrednosti u dva podskupa (potrebno naći optimalnu podelu)

Podela na osnovu **kontinualnih atributa**

Diskretizacija: formiranje diskretnih atributa

- Statička –jedna diskretizacija na početku
- Dinamička –određivanje rangova u toku izgradnje stabla

Binarna odluka: (A < v) ili (A v)

- uzima u obzir sve moguće podele i bira najbolju
- može biti zahtevnije u pogledu izračunavanja

5. Kreiranje stabla odluke – kriterijumi za selekciju atributa

- Informacijska dobit
  - Pretpostavka da su svi atributi kategorički
  - Može se modifikovati za kontinualne atribute
- Mera dobitka
  - Normalizuje informacijsku dobit
  - Rešava problem što informacijska dobit favorizuje atribute sa mnogo vrednosti
- Gini indeks
  - Pretpostavka da su svi atributi kontinualni
  - Pretpostavka da za svaki atribut postoji nekoliko mogućih vrednosti za podelu;može zahtevati dodatne alate (npr. klasterovanje) za određivanje tih vrednosti
  - Može se modifikovati za kategoričke atribute

6. ID3 algoritam

Informacijska dobit

Zaustavljanje:

- Svi primeri pripadaju istoj klasi
- Najbolja informacijska dobit nije veća od 0

Ne primenjuje odsecanje

- ID3 je algoritam koji se koristi za indukovanje stabla odluke na osnovu primera tipa:
  - (v_atribut1, v_atribut2, ..., v_atributN, klasa)
- Dobijeno stablo odluke se kasnije koristi za klasifikaciju novih uzoraka.
  - (v_atribut1, v_atribut2, ..., v_atributN)
  - klasa= ?
- Za nalaženje optimalnog puta za klasifikaciju skupa primera za učenje, potrebno je mimimizovatiuslove/pitanja (minimizacija dubine stabla)
- Neophodna je funkcija koja će biti mera za izbor atributa/pitanja koje vrši najbolju (najbolje balansiranu) podelu.
- Takva funkcija je metrika Informacijska dobit.

Logika predikata

1. Logika predikata kao formalizam za predstavljanje znanja

Formalni jezik za predstavljanje znanja, definisan azbukom i simbolima preko kojih se grade formule (rečenice). Ekstenzija logike iskaza (propositional logic) **Istinitost formula** predikatske logike se utvrđuje pomoću uzastopne primene relacija iz istinitonosnih tablica, počev od unutrašnjosti prema spoljašnjosti formule. **Logike dugog reda**: nemonotone logike, temporalne logike, kvalitativno rezonovanje, belief management, planiranje i prepoznavanje planova, …

Prednosti logike predikata prvog reda:

- Postoji matematički aparat za zaključivanje
- Postoji preslikavanje na i sa prirodnog jezika

Nedostaci:

- Teškoće u predstavljanju aproksimativnog, heuristickog zdravorazumskog znanja, znanja o planovima, verovanja ili znanja o vremenu

2. **Dobro formirane formule u logici predikata prvog reda**
   Induktivno se definišu kao:
   - **Atomske formule (predikati)**

     $\rho(\tau 1, \tau 2, \ldots, \tau n)$
     gde je ρ n-arna relacija (predikat), a $\tau 1, \tau 2, \ldots, \tau n$ su termi (konstante, promenljive, funkcije)

Primer 1:	Primer 2:
Živeti (Pera, Kuća1)	Nosi(A, B)
Kupiti (Otac(Mika), Kuća1)	Kocka(A)
Vece(Abs(-5), 3)	Lopta(B)
Otac(Dejan, Ana)	Sto(C)
Boja(x, Plava)	Lezi_na(B, A)
Jednakost: Otac(Mika)=Petar	

   - **Logičke formule:**
     atomske formule + logicki operatori ($\neg \quad \wedge \quad \vee \quad => \quad <= \quad \Leftrightarrow$)

Primeri logičkih formula:	
Pas (Lesi) => Životinja (Lesi)	← Ako je Lesi pas onda je i životinja.
$Broj(x) \wedge Vece(x, y)$	
$Slon(Dambo) \Rightarrow Leti(Dambo)$	
$\neg Mladji(Nikola, Violeta)$	

   - **Kvantifikovane formule**

Primeri kvantifikovanih rečenica:
$\forall x\ Slon(x) \Rightarrow Boja(x, Siva)$
$\forall x\ Ptica(y) \Rightarrow Leti(y)$
$\exists z\ Ptica(z) \wedge \neg Leti(z)$
$\forall x \forall y\ Nosi(x, y) => Iznad(y, x)$
$\forall x \forall y \forall z\ Iznad(x, y) \wedge Iznad(y, z) => Iznad(x, z)$

3. **Kvantifikacija u logici predikata prvog reda**
   Za izražavanje tvrdnji "za svako ..." i "postoji ..."
   • Univerzalni kvantifikator, $\forall$
   • Egzistencijalni kvanitifikator, $\exists$
   Oblast delovanja kvantifikatora
   Redosled kvantifikatora je važan !! Ugnježdenje kvantifikatora – kombinacija univerzalnog i egzistencijalnog u istoj rečenici (oblast delovanja je formula uklopljena u kvantifikovanu rečenicu).

Veze izmedju kvantifikatora:
$\forall x \neg P \equiv \neg\exists x\ P$
$\forall x\ P \equiv \neg\exists x \neg P$
$\neg\forall x\ P \equiv \exists x \neg P$
$\exists x\ P \equiv \neg\forall x \neg P$

4. **Poklapanje uzoraka i unifikacija**
   Postupak u kome se ispituje da li neka formula može da postane istoventa sa datim podatkom, ako se promenljive zamene odredjenim vrednostima. Podatak je iskaz o stvarnom ili zamišljenom svetu (formula bez promenljivih). Lista smena je skup asocijacija izmedju promenljivih i izraza u kojima je svaka promenljiva vezana za najvise jedan izraz. Lista smena – skup parova promenljiva/vrednost.
   {x/A, y/F(B), z/C}

Primeri:

Otac (Nikola, Petar)	P(A, B, x, D, E)	Q(A, B, x, D)
Otac(x, Petar)	P(A, B, Q(C), D, E)	Q(A, C, B, D)
Lista smena {x/Nikola}	Lista smena: ??	Lista smena: ??

Ograničenja:
Ako uzorak (formula) ne sadrzi promenljive, ona se poklapa sa podatkom jedino ako su identični.
Kada se promenljiva pojavljuje više puta u uzorku, onda se ona uzima kao nepoznata samo prvi put, a kasnija pojavljivanja su vezana za prvo. Lista smena (binding list) se moze primeniti na formulu:
P(x, x, y, z) {x/A, y/F(B), z/C} = P(A, A, F(B), C)

Unifikacija (objedinjavanje) je proces u kome se utvrdjuje da li neka formula moze da bude identicna sa drugom formulom (generalizovano poklapanje uzoraka). Unify(p,q)=θ, tako da subst(θ,p)=subst(θ,q)
Dodatni uslov za listu smena: nijedna promenljiva sa dodeljenim izrazom ne javlja se u asociranom izrazu, ni u izrazima dobijenih iz njega zamenom promenljivih.

5. Zaključivanje u logici predikata prvog reda – pravila izvođenja

Izvodjenje (inference) je process dobijanja zaključaka iz premisa. Opšta pravila izvodjenja se primenjuju se na formule da bi se dobile nove formule. Svako pravilo se sastoji od uslova i zaključaka. Kada su prisutne formule koje odgovaraju uslovima, mogu se izvesti formule koje odgovaraju zaključcima.

Primena pravila izvođenja To je proces dobijanja zaključaka iz premisa. Pravila izvođenja se primenjuju na formule da bi se dobile nove formule. Svako pravilo čine uslovi i zaključci.

1) MP (MODUS PONENS)

$\Phi \Rightarrow \Psi$ ←premisa
Φ ←premisa

Ψ ← zaključak

Nosi(A, B)
Nosi(A, B) => Iznad(B, A)
Iznad(B, A)

2) MT (MODUS TOLENS)

$\Phi \Rightarrow \Psi$
$\neg \Psi$

$\neg \Phi$

3) AND ELIMINATION (AE) Eliminacija knjunkcije

$\Phi \wedge \Psi$

Φ
Ψ

4) AND INTRODUCTION (AI) Uvođenje konjunkcije

Φ
Ψ

$\Phi \wedge \Psi$

5) UNIVERSAL INSTANTIATION (UI) Eliminisanje univerzalnog kvantifikatora

$\forall \nu \Phi$

$\Phi_{\nu/\tau}$ ← τ menja ν
τ je izraz u kome se ν ne javlja kao slobodna promenljiva

$\forall$y Voli(Jelena, y)

Voli(Jelena, Marko)

6) EXISTENTIAL INSTANTIATION (EI) Eliminisanje egzistencijalnog kvantifikatora

$\exists \nu \Phi$

$\Phi_{\nu/\pi}(\nu 1, \nu 2, \ldots, \nu n)$
π je funkcija, a $\nu 1, \nu 2, \ldots, \nu n$ su slobodne promenljive u funkciji Φ

$\exists$z Mrzi(y, z)

Mrzi(y, f(y))

Primer zaključivanja:

P
P => Q
P => R ← Δ
Q => S

Q
R
S

6. Zaključivanje u logici predikata prvog reda – rezolucija

Rezolucija je procedura izvođenja koja se zasniva na pravilu izvođenja koje je poznato kao princip rezolucije. Princip rezolucije je logički zasnovan i kompletan sa jednim ograničenjem: Argumenti principa rezolucije su pojednostavljena verzija logike predikata, tzv. klauzule. Simboli, izrazi i atomi su isti kao u logici predikata, ali se koriste literali i klauzule umesto logičkih i kvantifikatorskih formula. Klauzalni oblik (clausal form): pojednostavljena verzija logike predikata (ekvivalent) Literal je atomska formula ili njena negacija. Klauzula je skup literala koji predstavlja njihovu disjunkciju.

7. Rezolucija – prevođenje u klauzalni oblik

1.Eliminisanje implikacije

$\alpha \Rightarrow \beta$ se zamenjuje sa $\neg \alpha \vee B$

2.Sužavanje oblasti delovanja negacije

$\neg \neg\alpha \equiv \alpha$

$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta \qquad \neg\forall x\, \alpha \equiv \exists x \neg \alpha$

$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta \qquad \neg\exists x\, \alpha \equiv \forall x \neg \alpha$

3.Standardizacija promenljivih

Promenljive se preimenuju tako da svaki kvatifikator ima jedinstvenu promenljivu.

$(\forall x\, P(x,x)) \wedge (\exists x\, Q(x))$ postaje $(\forall x\, P(x,x)) \wedge (\exists y\, Q(y))$

4.Eliminisanje egzistencijalnih kvantifikatora (Skolemizacija)

1) Ako $\exists$ nije u oblasti delovanja $\forall$, onda se uvodi Skolem konstanta: $\exists x\, P(x)$ se zamenjuje sa $P(A)$, gde je A nova konstanta

2) Ako $\exists$ jeste u oblasti delovanja $\forall$, onda se uvodi Skolem funkcija: $\forall y\, \exists x\, P(x)$ se zamenjuje sa $\forall y\, P(f(y))$, gde je f nova funkcija

5.Izbacivanje univerzalnih kvantifikatora

6.Prevodjenje formule u konjuktivnu normalnu formu

$\alpha \vee (\beta \wedge \gamma)$ se zamenjuje sa $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

7.Eliminsanje konjukcije

$\alpha \wedge (\beta \vee \gamma)$ se zamenjuje sa $\{\alpha\}$ i $\{\beta, \gamma\}$ (skup klauzula)

8.Preimenovanje promenljivih

Nijedna promenljiva se ne pojavljuje u više od jedne klauzule.

8. Strategije rezolucije

a) Čista rezolucija (proba svako sa svakim, nema strategije) Jednostavne strategije rezolucije

b) Eliminacija čistih literala - čist literal je onaj koji nema odgovarajući komplementarni literal u polaznoj bazi. Klauzule koje sadrže čiste literale se izbacuju. Ova strategija se primenjuje samo jednom i to na početku.

Primer.

$\{ \neg P, \neg Q, R \}$

$\{ \neg P, S \}$)

$\{ \neg Q, S \}$) ← Izbacuju se ove dve klauzule zato što je S čist literal

$\{ P \}$

$\{ Q \}$

$\{ \neg R \}$

c) Eliminisanje tautologija – tautologija je klauzula koja sadrži par komplementarnih literala. Ova strategija se ne primenjuje ukoliko su literali komplementarni nakon primene unifikacije. Može da se primenjuje više puta u toku rada.

Ostali formalizmi

1. **Arhitektura produkcionih sistema**
   Interfejs
   Baza znanja: produkciona pravila (If-Then pravila)
   Radna memorija: činjenice
   Mehanizam zaključivanja
   - Lančanje unapred (forward chaining) data-driven Sve cinjenice dostupne na pocetku. Pogodno za sintezu
   - Lančanje unazad (backward chaining) goal-driven Sva ciljna stanja dostupna na pocetku. Pogodno za dijagnostiku
   - U oba smera (bi-directional)
2. **Zaključivanje lančanjem unapred**
   Zaključivanje unapred (forward chaining) - početak -> cilj;
   Kreće se od početnog stanja ka ciljnom stanju (stanjima). Ova vrsta zaključivanja je vođena podacima (zavisno od vrednosti parametara stanja u datom čvoru definiše se put pretrage). **Lančanje unapred**: polazi se od činjenica iz radne memorije, upoređuju se sa pravilima, okidaju se (primenjuju) pravila čiji je uslov zadovoljen (IF deo pravila), i zaključak (THEN deo pravila) se dodaje radnoj memoriji. Produkcioni sistemi su oni koji rade na principu zaključivanja unapred (FORWARD CHAINING).
   Winston-ov algoritam:
   - Sve dok se problem ne reši ili nema pravila čiji se uslovi mogu zadovoljiti u trenutnoj situaciji radi:
     - Pronađi pravila čiji su uslovi zadovoljeni (ako ih ima više, primeniti postupak za razrešenje konflikta)
     - Izvršiti akcije koje su pridružene aktiviranim pravilima

   Algoritam zaključivanja:
   1. Match: Pronadji sva primenljiva pravila (If deo je zadovoljen)
   2. Conflict resolution: Izaberi pravilo koje ce se izvršiti
      - Odbaci pravila koja su već ranije primenjena
      - Izaberi novo pravilo (conflict-resolution strategy)
   3. Act: Dodaj novu činjenicu u WM

3. **Zaključivanje lančanjem unazad**
   Zaključivanje unazad (backward chaining) - cilj ->početak; Kretanje se vrši od ciljnog stanja ka početnom stanju. Ovakvo zaključivanje je ciljno orijentisano (ne vodi računa kojim putem će se kretati, ali zna dokle mora da stigne**). Lančanje unazad**: polazi se od ciljnih činjenica, upoređuju se sa THEN delom pravila, primenjuju se pravila čiji je zaključak zadovoljen, uslovi (činjenice iz IF dela) se dodaju radnoj memoriji kao novi ciljevi.

   Kao kombinacija navedena dva načina pretraživanja, u nekim slučajevima se uspešno koristi dvosmerno pretraživanje: unapred od početnog stanja i unazad od ciljnog (ciljnih) stanja. Pretraga se završava kada se ova dva procesa pretrage "sretnu" na nekom delu grafa.

4. Strategije za razrešavanje konflikata

- Primenjuje se samo jedno pravilo, npr. prvo u bazi čiji je uslov ispunjen
  - No duplication: izaberi pravilo koje daje nove zakljucke
  - Do one: izaberi prvo pravilo ciji je uslov ispunjen
  - Do the most specific: primeni najspecificnije pravilo. Specifičniji je uslov koji sadrži više premisa. o Do the most recent: izaberi pravilo ciji uslov koristi najnovije činjenice
- Do all: primeni sva pravila, svi dobijeni zaključci se dodaju bazi činjenica
- Do in sequence: Primena pravila jedno za drugim tako da svaki zaključak može da se iskoristi za ispunjenje uslova novog ili sledećeg pravila

5. AND/OR stablo

Za grafički prikaz procesa zaključivanja koriste se AND/OR stabla

Primer: Nacrtati AND/OR stablo za zaključivanje lančanjem unapred

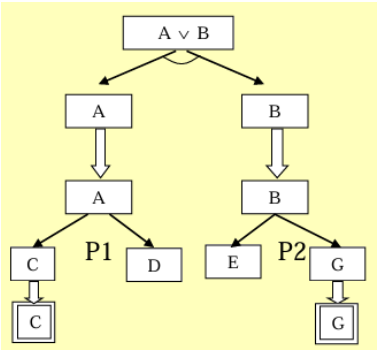
Data su pravila:

P1: A => C ∧ D

P2: B => E ∧ G

Činjenice: A ∨ B

Dokazati: C ∨ G



Primer: AND/OR stablo za lančanje unazad:

Primer produkcionog sistema za dokazivanje nejednakosti

Predikat :

V(x, y) ← x je veće od y

Skup pravila

P1: (V(x,0) ∧ V(y, 0)) => V(puta(x, y), 0)

P2: (V(x, 0) ∧ V(y, z)) => V(plus(x, y), z)

P3: (V(x, w) ∧ V(y,z)) => V(plus(x, y), plus (w, z))

P4: (V(x,0) ∧ V(y, z)) => V(puta(x, y), puta(x, z))

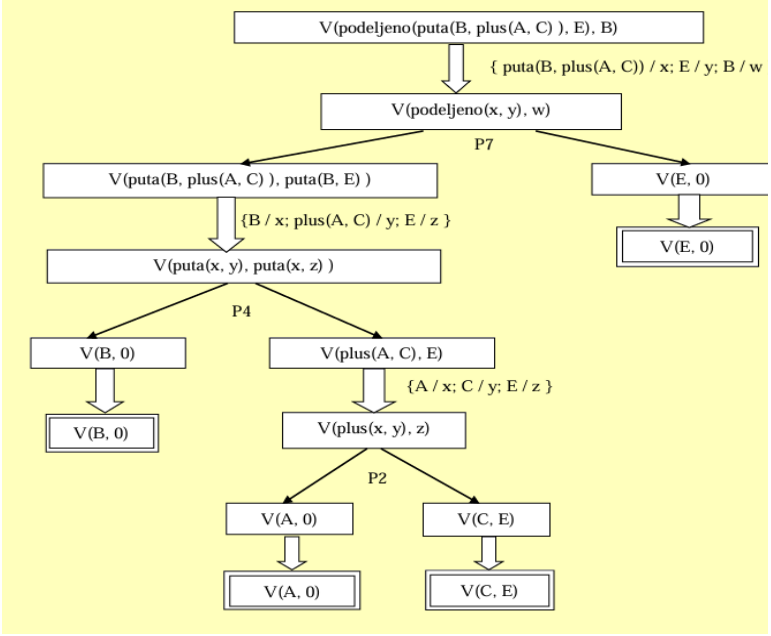
P5: (V(1, w) ∧ V(x, 0)) => V(x, puta(x, w))

P6: V(x, plus(puta(w, z), puta(y,z))) => V(x, puta(plus(w, y), z)

P7: (V(x, puta(w, y)) ∧ V(y, 0)) => V(podeljeno(x, y), w)

Činjenice: E>0, B>0, A>0, C>E, C>0

Dokazati: (B(A + C) / E) > B



6. Semantičke mreže

Formalizam za predstavljanje znanja, nastao u cilju razumevanja načina dugoročnog pamćenja kod ljudi. Semantičke mreže opisuju kategorije objekata (čvorovi u mreži) i relacije medju njima (potezi). Glavni oblik zaključivanja je nasledjivanje. Svaki čvor u mreži ima ime, koje ne mora da bude jedinstveno. Opisuju se prototipovi, a moguće je opisati i izuzetke (nemonotona logika)

Tipovi veza:

- SubsetOf (a-kind-of)
- MemberOf (is-a)
- Relacija izmedju dva objekta (strelica sa imenom relacije)
- Relacija izmedju svakog elementa klase A i objekta B (ime relacije uokvireno)
- Relacija izmedju svakog elementa klase A i nekog elementa klase B (ime relacije dvostruko uokvireno)

Implementacija

- Stvarne vrednosti
- Default vrednosti
- Procedure (if needed, if added)

Procedure

Procedure se mogu dodati u mrežu, da bi se izračunale neke vrednosti (procedural attachment) Procedure za:

- dodavanje i brisanje čvora
- dodavanje i brisanje potega
- pretraživanje
- if-added
- if-needed

Prednosti:

- Pogodne za hijerarhijski uredjene podatke
- Podrazavaju način organizacije memorije kod ljudi Pogodne za unarne i binarne relacije
- Nasleđivanje

Nedostaci:

- Teškoće pri predstavljanju relacija višeg stepena
- Disjunkcija i negacija se tesko predstavaljju
- Teško se izvodi kvantifikacija

7. Okvir

Okvir je struktura za organizovanje znanja, posebno podrazumevanog znanja. Svaki okvir predstavlja jednu klasu elemenata na isti način na koji se koristi čvor klase da predstavi takve elemente u semantičkoj mreži. Okvir opisuje entitet preko njegovih osobina. Svaka osobina ima vrednost koja se smešta u slot okvira. Vrednosti okvira mogu ali ne moraju biti specificirane. Za neke slotove se mogu vezati procedure koje treba aktivirati u određenim situacijama. Okviri su uređeni u hijerarhije preko vrednosti nekog slota. Vrednosti slotova se mogu nasleđivati ili se mogu dodeliti nove vrednosti. Postoji i proceduralno znanje: za svaki slot se vežu određene procedure koje se izvršavaju u određenim situacijama.Okviri se uređuju u hijerarhije na osnovu veze is-a.Moguće je kombinovati deklarativno (vrednosti slotova) i proceduralno znanje (vezane procedure).

Vrste procedura koje se koriste u okvirima:

1. IF-ADDED definiše postupak koji se obavlja kod dodeljivanja vrednosti datom slotu. Procedura mora uspešno da se izvrši da bi se upisala vrednost.

2. IF-NEEDED definiše postupak prilikom pretraživanja vrednosti slota. Mora uspešno da se izvrši da bi se pročitala vrednost.

3. IF-UPDATED definiše postupak prilikom editovanja vrednosti slota.

Nepouzdano zaključivanje

1. **Rad sa nepouzdanim podacima**
   Sposobnosti čoveka: sposobnost efektivnog rada sa nepreciznim, nepotpunim I ponekad nesigurnim informacijama.
   Postoji više vrsta nesigurnosti:
   - Nesigurno znanje
   - Nesigurni podaci
   - Nesigurne informacije

U stvarnom svetu agent skoro nikada nema potpune i tačne podatke o okruženju.

Primer: AkcijaAt–krenina aerodrom t minuta pre poletanja

- Problemi
- Neki delovi okruzenja nisu dostupni
- Sumu podacima dobijenih sa senzora
- Neizvesni rezultati akcija
- Kompleksnost modeliranja I predvidjanja u stvarnom okruženju

2. **Nepouzdano zaključivanje kod produkcionih sistema**
   Nepouzdano zaključivanje se koristi kod produkcionih sistema da bi se omogućio rad sa nepouzdanim i nedovoljno preciznim informacijama.
   Činjenicama i pravilima se podeljuje
   - Faktor pouzdanosti (FP): verovatnoća tačnosti tvrdnje:
     - Činjenice
     - Dobijenog zaključka iz pravila

Genetski algoritmi

1. Opis genetaskog algoritma

Algoritam zahteva polazni skup rešenja predstavljenih odgovarajućim hromozomima->**populacija**. Rešenja iz jedne populacije se koriste da formiraju novu populaciju u nadi da će ona biti bolja Izbor rešenja za novu populaciju bira se na osnovu **dobrote(fitness) rešenja**.
Ovaj proces se ponavlja sve dok se **ne zadovolji određeni uslov**(broj populacija, poboljšanje najboljeg rešenja).

Koraci u genetskom algoritmu:

[**Početak**] Generiše se slučajna populacija od **n** hromozoma.

[**Dobrota**] Računa se dobrota **f(x)** za svaki hromozom iz populacije.

[**Nova populacija**] se kreira ponavljanjem sledećih koraka:

1. [**Izbor roditelja**] Biraju se dva hromozoma iz tekuće populacije prema dobroti (veća dobrota, veća šansa za izbor).
2. [**Rekombinacija**] Novi hromozom se formira kombinovanjem dva izabrana horomozoma.
3. [**Mutacija**] Novi hromozom mutira sa određenom verovatnoćom.
4. [**Dodavanje potomka**] Novi hromozom se dodaje u novu populaciju.

[**Zamena populacija**] Novo generisana populacija postaje tekuća populacija.

[**Uslov za kraj**] Proverava se uslov za završetak algoritma → ukoliko je zadovoljen vraća se najbolje rešenje iz tekuće populacije.

[**Novi ciklus**] Ako uslov nije ispunjen ponavlja se algoritam od 2. koraka.

2. Kodiranje hromozoma

Tehnike koje se koriste za rekombinovanje i mutaciju uglavnom zavise od načina kodiranja hromozoma.
Postoje sledeći načini za kodiranje:

- binarno kodiranje,
- permutacionokodiranje,
- kodiranje vrednostima i
- kodiranje stablom.

Binarno kodiranje

Predstavljanje hromozoma nizom bitova (0 i 1). Najčešće se koristi zbog jednostavnosti. jednostavno mogu da se koriste u operatorima ukrštanja i mutacije. Problem kod binarnog kodiranja je taj što ono nije prirodno za određene primene. Zbog toga su često neophodne korekcije nakon rekombinacije i/ili mutacije.

Permutaciono kodiranje

Koristi se kod problema uređivanja. Svaki hromozom predstavlja sekvencu elemenata čije optimalno uređenje se traži.
Hromozom A: 2 7 4 1 3 5 8 6 Hromozom B: 5 3 7 2 8 1 4 6
Pri rekombinovanju i mutiranju hromozoma mora se voditi računa o očuvanju konzistencije hromozoma.

Kodiranje vrednostima

Direktno kodiranje vrednostima se koristi kod problema gde se koriste složenije vrednosti. Korišćenje binarnog kodiranja u ovim slučajevima može biti nezgodno. Kod kodiranja vrednostima hromozom predstavlja sekvencu vrednosti određenog tipaa

Hromozom A: 1,25 -5,24 -0,23 0,01 -3,32 2,74
Hromozom B: AHGKATWEM
Hromozom C: (nazad), (nazad), (levo), (napred)
Ovaj tip kodiranja zahteva da se razviju specijalizovane operacije **rekombinacije i mutacije koje odgovaraju konkretnom problemu koji se rešava.**

Kodiranje stablom

Kodiranje stablom se uglavnom koristi za razvoj programa ili izraza -> genetsko programiranje.
Svaki hromozom predstavlja stablo koje u čvorovima sadrži određene objekte: funkcije, operatore, konstante, promenljive, komande prog. jezika i dr. Genetsko programiranje je jednostavno u LISP-u jer se tamo program i podaci predstavljaju isto pomoću stabla (ugnježdene liste).

3. Izbor roditelja

Rulet selekcija

Roditelji se biraju na osnovu svoje dobrote.
Što je veća dobrota jedinke veća je šansa da bude izabrana za roditelja.

Rangiranje

Rešava problem kada postoje velike razlike u dobrotama jedinki.
Jedinke se sortiraju po dobroti pa im se dodele rangovi od 1 do N (veličina populacije).
Pri izboru roditelja se ne gleda dobrota već rang.
Na ovaj način sve jedinke imaju šanse da budu izabrane (mana je sporija konvergencija).

Metod stabilnog stanja

Zasniva na ideji da novu generaciju treba da u velikoj meri čine najbolje jedinke iz prethodne generacije.
Metod funkcioniše na sledeći način:

- Iz tekuće populacije izbacuje se određen broj najlošijih jedinki.
- Bira se par najboljih jedinki koje daju onoliko novih potomaka koliko je izbačeno.
- Ostatak populacije se prenosi u novu generaciju.

Elitizam

Elitizamje uveden da bi se sprečilo gubljenje najboljih rešenja pri kreiranju nove populacije. Kod elitizma prvo se kopira par najboljih rešenja u novu populaciju, dok se ostatak nove generacije dobija rekombinacijomi mutacijom.

4. Modifikacija hromozoma – mutacija

Operator mutacije formira potomka samo od jednog roditelja (menja jedinku mutacijom). Proizvodi male nasumične promene.
Promena gena sa verovatnoćom pm
Binarno kodiranje- Na slučajan način izabere jedan bit niza i promeni njegovu vrednost. Definiše se verovatnoća sa kojom se bit u hromozomu invertuje
Permutaciono kodiranje- Vrši se zamena (permutacija) određenih slučajno izabranih stavki iz hromozoma. Verovatnoća mutacije definiše koliko zamena će biti izvršeno.
Kodiranje vrednostima- Mutacija se vrši dodavanjem proizvoljne male vrednosti pojedinim vrednostima koje čine hromozom. Verovatnoća mutacije definiše koliko vrednosti iz hromozoma će biti modifikovano i kolika će veličina modifikacije biti.
Kodiranje stablom - Mutacija se vrši izmenama u proizvoljim čvorovima stabla. Mogu se menjati operatori, funkcije, konstante, promenljive i dr.

5. Modifikacija hromozoma – rekombinacija

Operator rekombinacije formira potomke od dva roditeljska niza bitova, tako što kopira određene bitove iz jednog, odnosno drugog roditelja. Bit na poziciji i svakog potomka je kopiran sa bita sa pozicijei jednog od roditelja. Izbor od kog roditelja će bit na poziciji I biti kopiran određuje se na osnovu dodatnog niza bitova koji se naziva maska rekombinacije ili maska ukrštanja.

Jednostruko ukrštanje –sa jednom tačkom prelaska

Maska ukrštanja se formira tako da počinje nizom od n uzastopnih jedinica i potrebnim brojem nula da bi se kompletirao niz. Primenom ove maske dobija se potomak kod kojeg je prvih n bitova iz prvog roditelja, a preostali bitovi iz drugog roditelja. Svaki put kada se primenjuje jednostruko ukrštanje broj nse bira na slučajan način.

Dvostruko ukrštanje –sa dve tačke prelaska

Maska ukrštanja je niz bitova koji počinje sa n_0 nula, za kojim sledi niz od n_1 jedinica i na kraju je neophodan broj nula da se kompletira niz. Primenom ove maske dobija se potomak kod kojegje središnjideo jednog roditelja upisan na odgovarajuću poziciju u sredinu drugog roditelja. Svaki put kada se maska primenjuje, vrednosti n0 i n1 se biraju na slučajan način.

Uniformno ukrštanje

Maska se formira kao slučajan niz nula i jedinica. Potomci se formiraju tako što se bitovi uniformno kopiraju od oba roditelja.

6. Evaluacija – izračunavanje fitnesa

Dobrota(fitness) novonastalog organizma se meri njegovim uspehom u životu (preživljavanju).
Pri rešavanju nekog problema traži se rešenje koje će biti najbolje od svih. Prostor svih mogućih rešenja, tj. skup rešenja u kome se traži najbolje rešenje se naziva prostor traženja. Svaka tačka u prostoru traženja predstavlja po jedno rešenje problema. Za svako moguće rešenje se može odrediti dobrota (fitness) za zadati problem. Traženje rešenja je ekvivalentno traženju ekstremnih vrednosti (minimum ili maksimum) u prostoru traženja. U određenim slučajevima prostor traženja može biti jasno definisan, ali najčešće su poznate samo određene tačke (rešenja). Korišćenjem genetskih algoritama, proces traženja (evolucija) generiše druge tačke u prostoru traženja (moguća rešenja).

Planiranje

1. **Formulacija planiranja**
   - Šta treba uraditi (**ciljevi**),
   - kako(i kada) to uraditi (**plan**).

   Problem planiranja -kako doći od trenutnog stanja do željenog ciljnog stanja?
   Planiranje uključuje
   - Selekciju akcija, Sekvencu akcijai Upravljanje resursima

   Planovi mogu biti
   - sekvence akcija
   - politika/strategije(stablo akcija)

   Planiranje je proces odlučivanjau nizu akcija koji prethodi njihovom izvršenju. Agent koristi znanje o akcijama i njihovim posledicama da nadje rešenje u prostoru.

2. **Green-ova formulacija planiranja**
   Bazira se na poznavanju posledica mogućih akcija i upotrebi tog znanja radi postizanja nekog cilja.
   Polazeći od opisa σ(polazno stanje), Γ(skup akcija), ρ (cilj), i Ω(baza formula o polaznom stanju, cilju i akcijama) odrediti plan ϒ.
   - Plan mora da zadovolji uslove:
     - sve akcije koje se javljaju u planu moraju biti elementi skupa akcija Γ
     - mora postojati dokaz iz Ω da plan ϒ dovodi do stanja koje zadovoljava cilj ρ počevši od polaznog stanja σ
   - Plan je akcija ili skup akcija koji su ulaz u IZVRŠILAC–primenomakcija na polazno stanje idaje stanje gkoje zadovoljava opis cilja i .

   Posmatra planiranje kao dokazivanje teorema
   Bazirana na rezoluciji (odgovaranje na pitanja)
   Polazi se od formule

   $\exists p\ Cilj(Result(p,s))$

   Akcije se uredjuju u blokove (nizove akcija)
   Result(l,s)
   - $\forall s\ Result([\,],s) = s$
   - $\forall a\ \forall p\ \forall s\ Result([a \mid p],s) = Result(p, Result(a,s))$

3. **Definicija operatora kod Green-ove formulacije**
   Operatori:
   U –unstack, U(x, y) -skida blok xsa bloka y i stavlja ga na sto
   S –stack, S(x, y) -stavlja blok x, koji se nalazi na stolu, na blok y
   M –move, M(x, y, z) -pomera blok x, koji se nalazi na bloku y, na blok z
   Null–not, NOOP

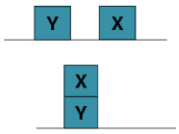
Preduslovi za izvršenje operatora =>Posledice izvršenja operatora

Definicija operatora **U(x, y)**:

$T(On(x, y), s) \wedge T(Clear(x), s) \Rightarrow$
$T(Table(x), Result(U(x, y), s)) \wedge$
$T(Clear(y), Result(U(x, y), s))$

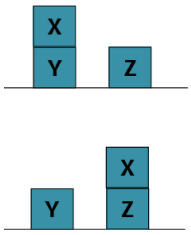
Definicija operatora **S(x, y):**

$T(Table(x), s) \wedge T(Clear(y), s)$
$\Rightarrow$
$T(On(x, y), Result(S(x, y), s))$



Definicija operatora **M(x, y, z):**

$T(Clear(x), s) \wedge T(On(x, y), s) \wedge T(Clear(z), s)$
$\Rightarrow$
$T(On(x, z), Result(M(x, y, z), s)) \wedge$
$T(Clear(y), Result(M(x, y, z), s))$



4. **Aksiome okvira kod Green-ove formulacije**

Koriste se da definišu prelaze iz tekućeg stanja u novo stanje blokova koji nisu pod uticajem operatora.

Unstack-U(x,y)

$T(Clear(u), s) \Rightarrow T(Clear(u), Result(U(x, y), s))$
$T(Table(u), s) \Rightarrow T(Table(u), Result(U(x, y), s))$
$T(On(u, w), s) \wedge u \neq x \Rightarrow$
$T(On(u, w), Result(U(x, y), s))$

Stack- S(x, y)

$T(Clear(u), s) \wedge u \neq y \Rightarrow$
$T(Clear(u), Result(S(x, y), s))$
$T(Table(u), s) \wedge u \neq x \Rightarrow$
$T(Table(u), Result(S(x, y), s))$
$T(On(u, w), s) \Rightarrow$
$T(On(u, w), Result(S(x, y), s))$

Move- M(x, y, z)

$T(Clear(u), s) \wedge u \neq z \Rightarrow$
$T(Clear(u), Result(M(x, y, z), s))$
$T(Table(u), s) \Rightarrow$
$T(Table(u), Result(M(x, y, z), s))$
$T(On(u, w), s) \wedge u \neq x \Rightarrow$
$T(On(u, w), Result(M(x, y, z), s))$

5. **STRIPS algoritam**

Cilj STRIPS-a je da se nađe sekvenca operatora koja sistem prevodi iz početnog stanja u ciljno stanje.
STRIPS algoritam koristi sledeće strukture podataka:

- Tekuće stanjeproblema opisano u predikatskoj logici.
  - Inicijalno je to opis početnog stanja.
- Ciljni stekkoji sadrži stavove koji odgovaraju trenutnom (pod)cilju.
  - Inicijalno je to opis ciljnog stanje.
- Lista akcijakoja na kraju sadrži sekvencu operacija koja predstavlja plan
  - Inicijalno je lista prazna.

6. **Opis stanja kod STRIPS**

Za opis stanja i cilja koriste se ranije definisani predikati:

- Clear(x)-nijedan blok ne stoji na bloku x
- Table(x)-blok x se nalazi na tabli
- On(x, y)-blok x se nalazi na bloku y

7. **Operatori kod STRIPS**

- U –unstack, U(x, y) -skida blok xsa bloka y i stavlja ga na sto
- S –stack, S(x, y) -stavlja blok x, koji se nalazi na stolu, na blok y
- M –move, M(x, y, z) -pomera blok x, koji se nalazi na bloku y, na blok z

Defincija operatora:

- ***U(x, y)***
  - PREDUSLOV: ***Clear(x), On(x, y)***
  - UKLONI: ***On(x, y)***
  - DODAJ: ***Table(x), Clear(y)***
- ***S(x, y)***
  - PREDUSLOV: ***Table (x), Clear(y)***
  - UKLONI: ***Table(x), Clear(y)***
  - DODAJ: ***On(x, y)***
- ***M(x, y, z)***
  - PREDUSLOV: ***Clear(x), On(x, y), Clear(z)***
  - UKLONI: ***On(x, y), Clear(z)***
  - DODAJ: ***On(x, z), Clear(y)***

Definiše se početno stanje.
Definiše se cilj i postavi na stek.
Dok stek nije prazan uzima se stav sa vrha steka:
Ako je stav (pod)cilj proveriti da li je zadovoljen:

- Ako je zadovoljen uklanja se sa steka.
- U suprotnom se bira operator koji ga zadovoljava i stavlja na stek. Stavljaju se i svi njegovi PREDUSLOVI.

Ako je stav operator:

- Skida se sa steka i dodaje se listi akcija.
- Uklanjaju se stavovi prema listi UKLONI iz tekućeg stanja.
- Dodaju se stavovi prema listi DODAJ u tekuće stanje.

Izdati listu akcija.

Neuronske mreže

1. Osobine veštačkih neuronskih mreža

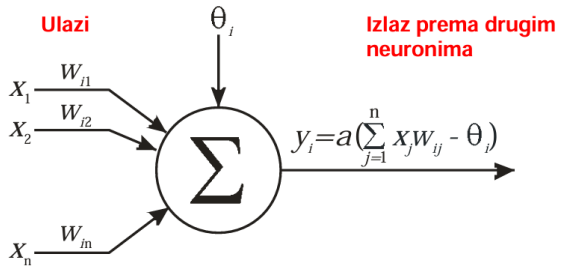
Veštačka neuronska mreža ima osobinu da prepoznaje obrasce u zadatoj kolekciji podataka i da izgrađuje model za te podatke.

Oponašaju prirodne nervne sisteme životinja i ljudi na dva načina:

- Akvizicija znanja se ostvaruje u procesu obučavanja neuronske mreže.
- Težine veza između neurona u mreži (sinapse) se koriste za skladištenje znanja.

2. Model veštačkog neurona

Veštački neuron predstavlja jednostavni procesni element koji obavlja relativno jednostavnu matematičku funkciju. Predstavlja imitaciju biološkog neurona.



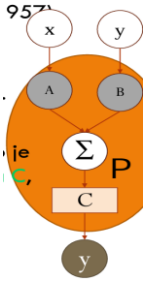
3. Perceptron

Neuron sa funkcijom praga.

Pretpostavka:

- dve ulazne vrednosti, x I y za perceptronP.
- Težine za x i y neka budu A I B respektivno.

Posledica: zbir na izlazu jeAx + By.

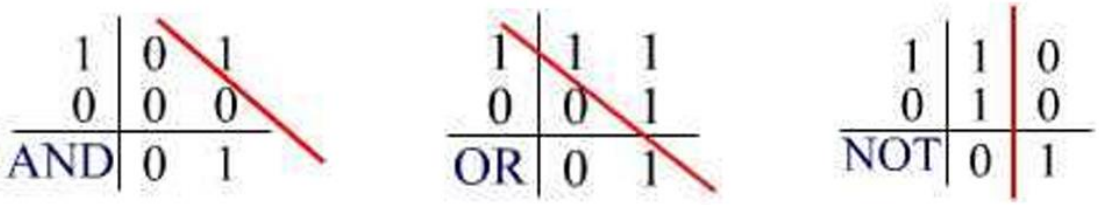


Pošto perceptrondaje non-zero izlaz ako je izračunata suma veća od zadatog praga C, izlaz se može napisati kao:

$$\text{Izlaz za P} = \begin{cases} 1 \text{ ako } Ax + By > C \\ 0 \text{ ako } Ax + By <= C \end{cases}$$

primer rada

- Ax + By > C iAx + By < C su dva regiona u xy ravni podeljeni linijom Ax + By + C = 0.
- Ako razmatramo ulaz(x, y) kao tačku u ravni, tada perceptronu stvari kaže u kom se regionu nalazi ta tačka.
- Takvi regioni, podeljeni jednom linijom su linearno separabilniregioni.
- Ovakav rezultat je koristan!
- Neke logičke funkcije kao što su AND, OR iNOT su linearno separabilne odnosno mogu se izvršiti jednim perceptronom.
- Ilustracija (za 2D prostor):



4. **Kako radi veštačka neuronska mreža**
   Postavi inicijalne vrednosti težina slučajno.
   Ulaz: tabela istinitosti za XOR
   Do
   - Čitaj ulaz (napr. 0, i0)
   - Izračunaj izlaz (napr.0.60543)
   - Uporedi ga sa očekivanim izlazom. (Diff= 0.60543)
   - Modifikuj težine u skladu s rezultatom.

   Loop until ispunjenje uslova
   - Uslov: određeni broj iteracija Uslov: prag greške
5. **Projektovanje ANN**
   - Inicijalne težine (male slučajne vrednosti ∈[-1,1])
   - Funkcija aktivacije(kako se ulazi i težine kombinuju da produkuju izlaz?)
   - Estimacija greške
   - Podešavanje težina
   - Broj neurona
   - Reprezentacija podataka
   - Veličina trening skupa
6. **Aktivacione funkcije**
   Aktivacione funkcije generalno su nelinerane.Linearne funkcije su ograničene pošto je izlaz jednostavno samo proporcionalan ulazu.

Linearna $a(x) = x$

Odskočna $a(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$

Linearna odsečena (korak po korak) $a(x) = \begin{cases} -1, & x \leq -1/k \\ kx, & -1/k < x < 1/k \\ 1, & x \geq 1/k \end{cases}$

Sigmoidalna (logistička) $a(x) = \dfrac{1}{1+\exp(-kx)}$

Tangens hiperbolički $a(x) = \tanh(kx) = \dfrac{\exp(kx) - \exp(-kx)}{\exp(kx) + \exp(-kx)}$

7. **Back propagation**

   Back-propagation je primer nadgledanog učenja. Koristi se na svakom nivou da bi se minimizovala greška između odgovora nivoa i aktuelnih podataka. Greška u svakom skrivenom sloju je srednja vrednost procenjene greške. Skriveni nivoi mreže setreniraju na taj način

   N je neuron.

   N_w je jedna od N-ovih ulaznih težina

   N_{out} je N-ov izlaz.

   $N_w = N_w + \Delta N_w$

   $\Delta N_w = N_{out} * (1 - N_{out}) * N_{ErrorFactor}$

   $N_{ErrorFactor} = N_{ExpectedOutput} - N_{ActualOutput}$

   Ovo radi samo za poslednji nivo, pošto znamo aktuelni izlaz kao i očekivani izlaz.

8. **Slojevite neuronske mreže**

   Neuroni su organizovani u slojeve.
   Na ulaz jednog neuronaiz sloja dovode se izlazi svih neuronaiz prethodnog sloja, a njegov izlaz se vodi na sve neuronenarednog sloja.
   Neuroniu prvom, ulaznom, sloju imaju samo po jedan ulaz -ulazi u mrežu.
   Izlazi neuronaiz poslednjeg, izlaznog, sloja predstavljaju izlaze mreže.
   Ostali slojevi neurona se nazivaju skriveni slojevi.

9. **Klasifikacija načina za obučavanje neuronskih mreža**

   U zavisnosti od toga **da li su poznati očekivani izlazi neuronske mreže** imamo:
   - nadgledano (eng. supervised) i
   - nenadgledano(eng. unsupervised) obučavanje.

   U zavisnosti od toga **da li se pri obučavanju menjaju i neki arhitekturni elementi mreže** (broj neurona) imamo:
   - statičko i
   - dinamičko obučavanje.

10. **Tipične primene neuronskih mreža**

    Klasifikacija
    - dijagnostika, prepoznavanje uzoraka, prepoznavanje govora

    Aproksimacija funkcija
    - modelovanje procesa, upravljanje

    Predikcija vremenskih serija
    - finansijska previđanja, modelovanje dinamičkih sistema

    Predprocesiranje
    - klasterizacija, vizuelizacija podataka, ekstrakcija podataka