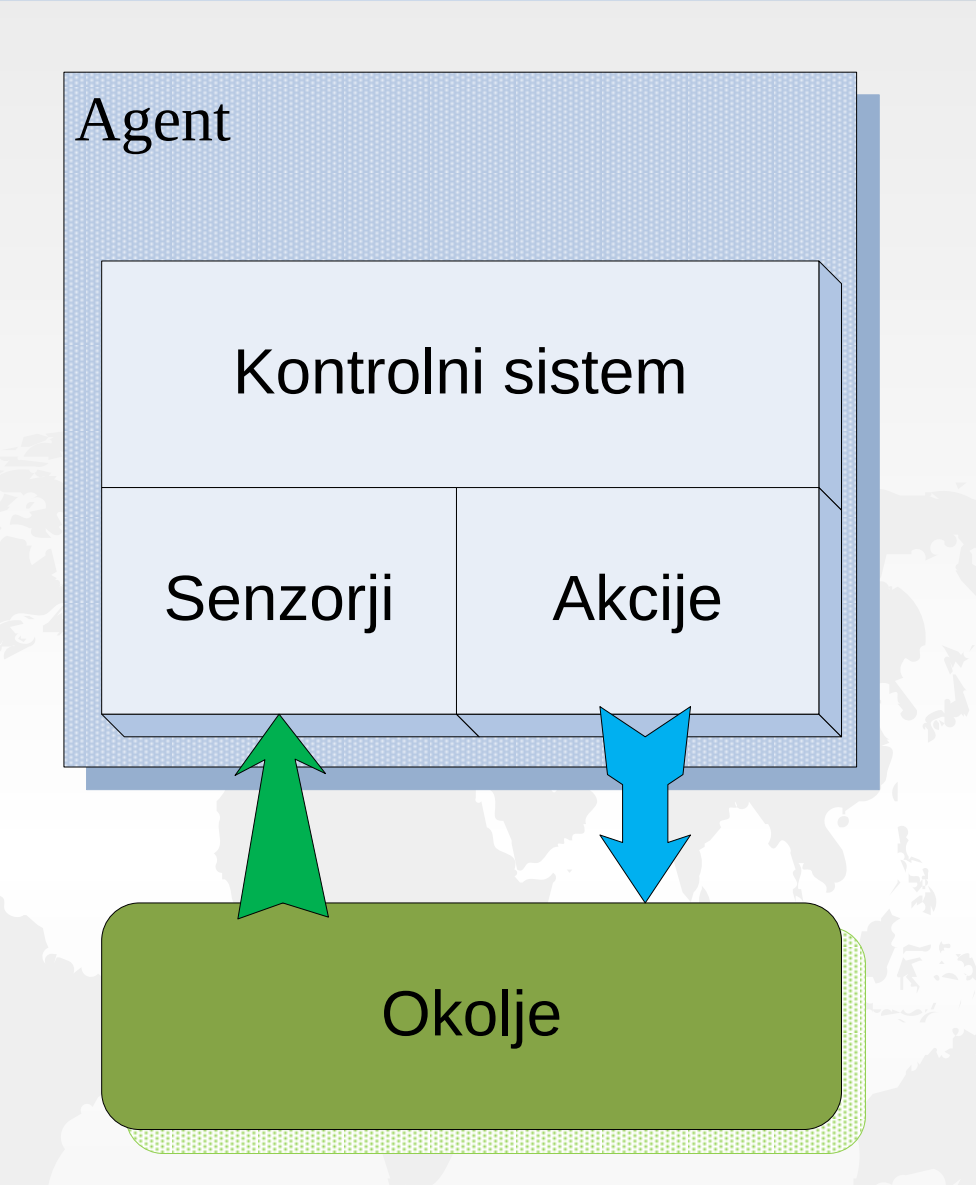
Inteligentni agenti in roboti

prof. dr. Marko Robnik Šikonja
Januar 2016

Definicije

- agent: entiteta, ki lahko izvrši različne akcije za rešitev določenega problema
- mnogo različnih definicij
- biološki (ljudje, živali), umetni (roboti, programi)
- softverski agent je program, ki za uporabnika izvrši določeno nalogo
- inteligentni agenti
- lastnosti, tipi agentov, agentne arhitekture
- eden temeljnih konceptov umetne inteligence

Agent



Multiagentni sistemi

- distribuirano reševanje problemov

program z več agenti, ki rešujejo probleme v interaktivnem okolju in so zmožni

  - avtonomnih,
  - fleksibilnih,
  - skupinsko organiziranih akcij
  - (ki so usmerjene k doseganju nekega cilja).

- Primer: kiloboti

Interaktivno okolje

- agent sprejema informacije,
- akcije spreminjajo okolje
- internet, igranje iger, robotika (npr. robo-soccer)

Avtomnomnost in fleksibilnost

- delovanje brez eksplicitnih ukazov
- kontrola nad svojimi akcijami
- notranje stanje
- (učenje iz izkušenj)

- odzivnost in aktivnost glede na situacijo
- načrtovanje, sledenje cilju

Skupinsko delovanje

- interakcija z drugimi entitetami (agenti in ljudmi)
- sodelovanje za doseganje ciljev: strategije, pogajanja, specializacija
- distribuirano, asinhrono reševanje problemov
- objekti in agenti

Aplikacije

- proizvodnja: hirerarhična organizacija, agenti modelirajo procese, optimirajo vire,..
- avtomatska kontrola procesov: transportni sistemi (nadzor, optimiranje poti, ...), zračni promet, ...
- telekomunikacije: porazdeljeni sistemi sodelujočih komponent, nadzor, upravljanje
- upravljanje z informacijami (zbiranje, filtriranje,)
- e-poslovanje (borze, iskanje produktov)
- interaktivne igre
- storitve (zdravstveni roboti, nakup kart,...)

Vrste agentov

- odzivni (reactive agents)
- sodelujoči (collaborative agents)
- vmesniški (interface agents)
- mobilni (mobile agents)
- informacijski (information-gathering agents)
- hibridni agenti

Odzivni agenti

- odziv na okolje glede na vnaprej definirana pravila
- primer: razvrščevalnik elektronske pošte, spam filter, vpisi v koledar
- učenje pravil, revizija pravil

Ciljno usmerjeni agenti

- ne le odziv na okolje, pač pa zasledovanje določenega cilja
- uporaba iskanja ali načrtovanja
- primer: iskanje specifičnih strani na internetu, najcenejše letalske karte

Uporabnostni agenti

- (utility based)
- definirana funkcija uporabnosti (koristnosti, zadovoljstva)
- poleg doseganja ciljev, agent maksimira še druge kriterije vsebovane v funkciji uporabnosti
- primer: poleg relevantnih strani še čas iskanja, kvaliteta strani
- funkcija uporabnosti pojasnjuje racionalnost agenta (npr. izgubi igro, da doseže druge cilje)
- racionalnost odločanja (A. 100% 3000, B. 80% 4000 C. 20 % 4000 D. 25% 3000), Kahneman, Tversky

Vmesniški agenti

- opravljajo naloge za uporabnika
- osebni asistent
- sodelujejo z uporabnikom
- učenje (lahko tudi od drugih agentov)
- primer: pomoč pri učenju programa, uporabnik označi katere spletne strani (novice) so mu všeč

Mobilni agenti

- fizično (roboti) ali virtualno premikanje (mreža, računalniki)
- primer: virus, nadzorni program

Informacijski agenti

- zbiranje informacij (na internetu)
- uspešnost merimo s priklicom in natančnostjo
- učenje
- problem: šumnost in verodostojnost podatkov

Sodelujoči agenti

- tudi zelo šibki agenti lahko v skupini dosežejo dober rezultat (npr. mravlje, genetski algoritem)
- prednost: redundanca, paralelnost

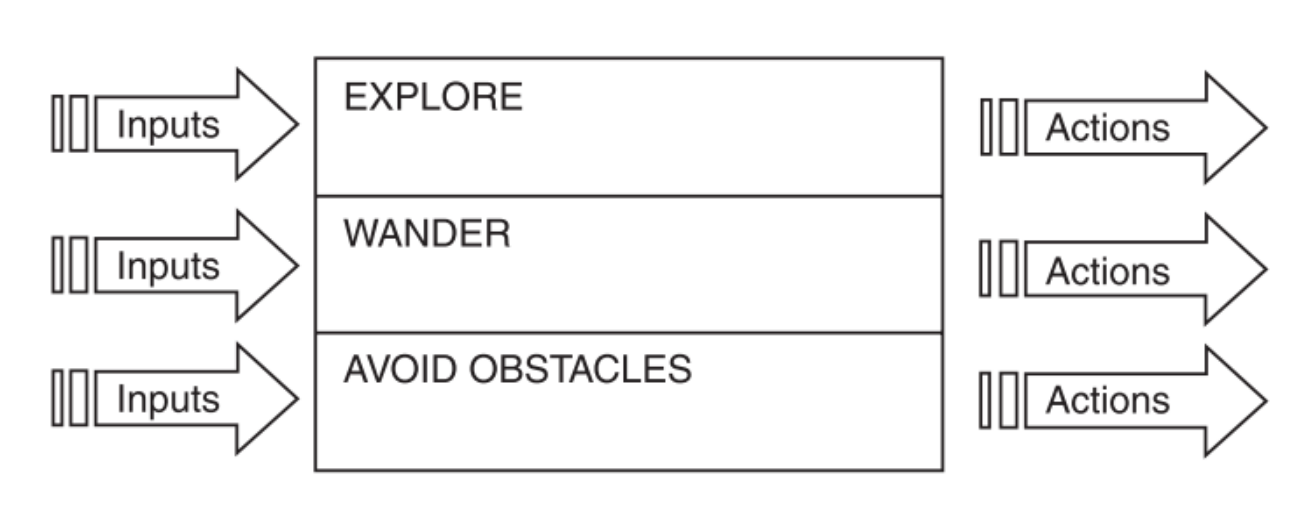
Agentne arhitekture

- način kako je agent zgrajen, kako so posamezni deli med seboj povezani in z okoljem

Arhitektura vsebovanosti

- Brooks, 1985, angl. subsumption architecture (intelligence without representation)
- večnivojska arhitektura namenjena primitivnim robotom, brez centralne inteligence
- vhod, izhod in več plasti, kjer vsaka zasleduje svoje cilje
- višji nivoji lahko blokirajo akcije nižjih

Primer:



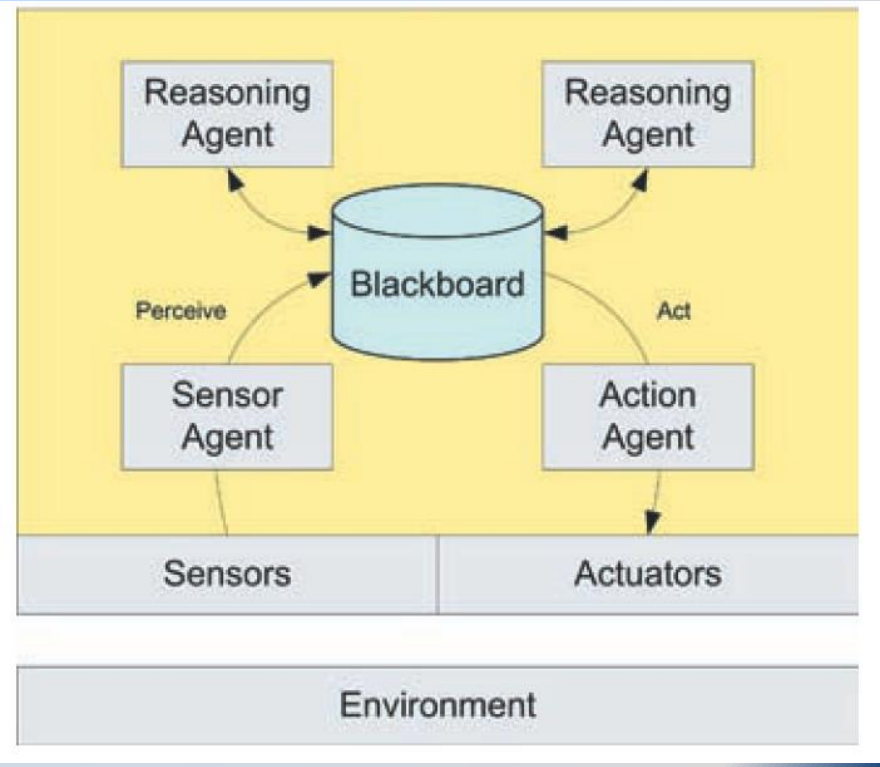
- vsak nivo ponavadi vsebuje pravila (npr. if-then)
- enostavno dodajanje novih nivojev

BDI arhitektura

- BDI (Belief Desire Intention – verjetje, želja, namen)
- verjetje je izjava o okolju, za katero agent verjame, da je resnična (množica takih izjav opisuje agentov pogled na okolje)
- želja je ciljno stanje, ki ga želi agent doseči
- namen je načrt, ki ga ima za dosego cilja
- agent preveri različne možnosti, izbrana možnost postane njegova želja in nameni se, da jo bo dosegel kot svoj cilj
- namenu sledi, dokler ga ne doseže ali ne postane nemogoč
- cilj se lahko spremeni
- drzni in previdni agenti

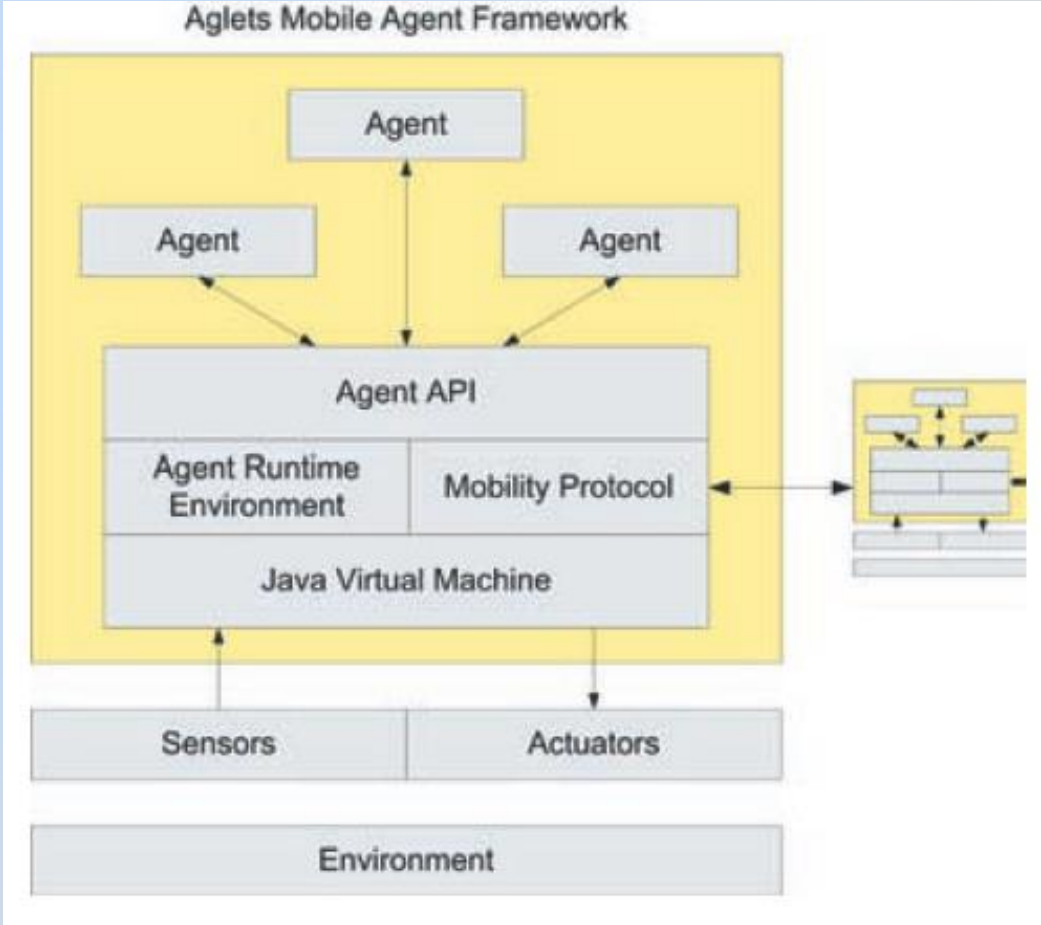
Arhitektura skupne delovne površine

- blackboard architecture
- vsi agenti si delijo skupno delavno področje
- delovna površina vsebuje informacije o okolju in vmesne rezultate
- agenti so lahko tudi specializirani
- koordinacija dostopa
- implemetacija agentov kot niti



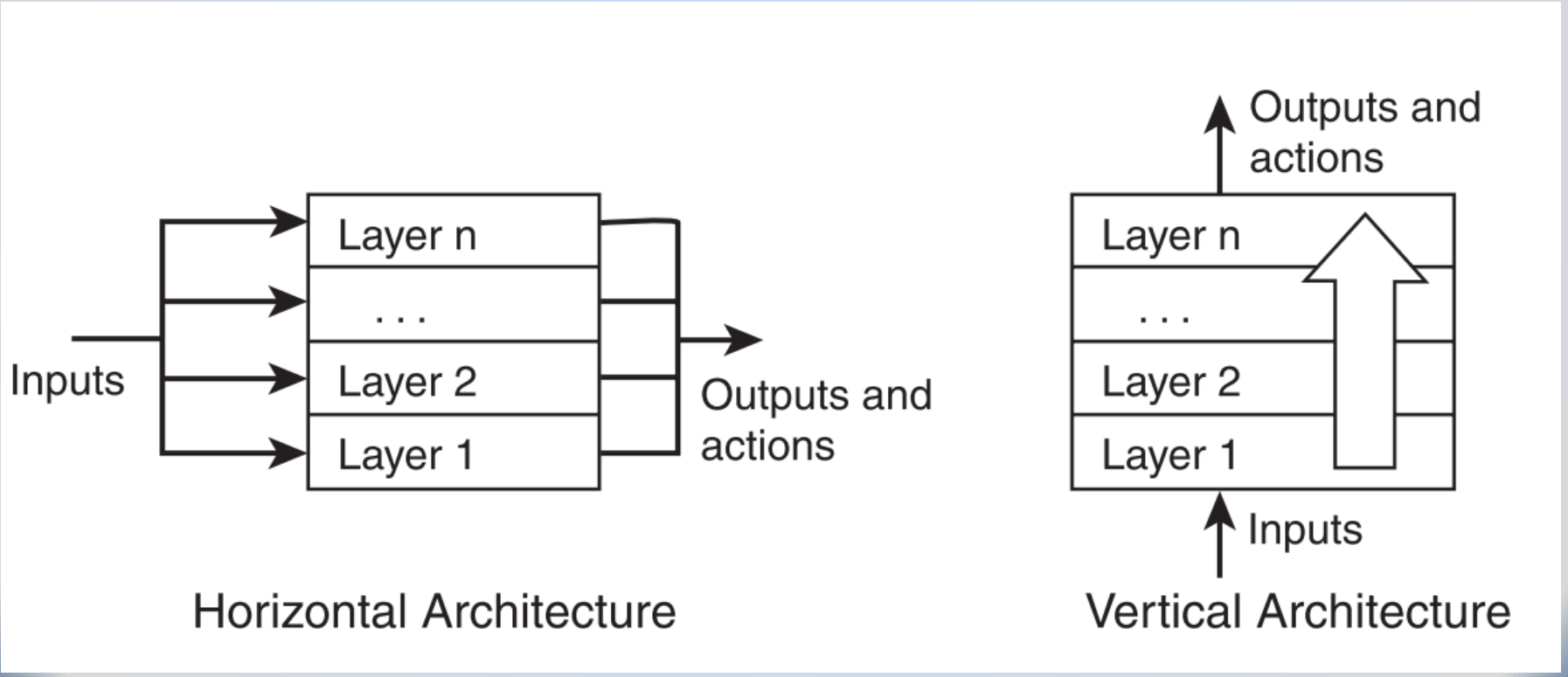
Primer mobilne arhitekture

- Aglets, IBM 1990
- Java, serializacija, peskovnik



Horizontalne in vertikalne arhitekture

- vhod, modeliranje okolja, načrtovanje, sodelovanje, izhod
- vsak nivo hrani svoje znanje



Okolje

- deterministično (poznamo vsa dejstva, agent lahko predvidi učinke svojih akcij – npr. preproste igre, tovarna)
- nedeterministično: agent lahko predvidi verjetnost za spremembo stanja, sklepa stohastično (npr. spremenljivo okolje, ne poznamo nasprotnikove poteze pri igri)

Učeči se agenti

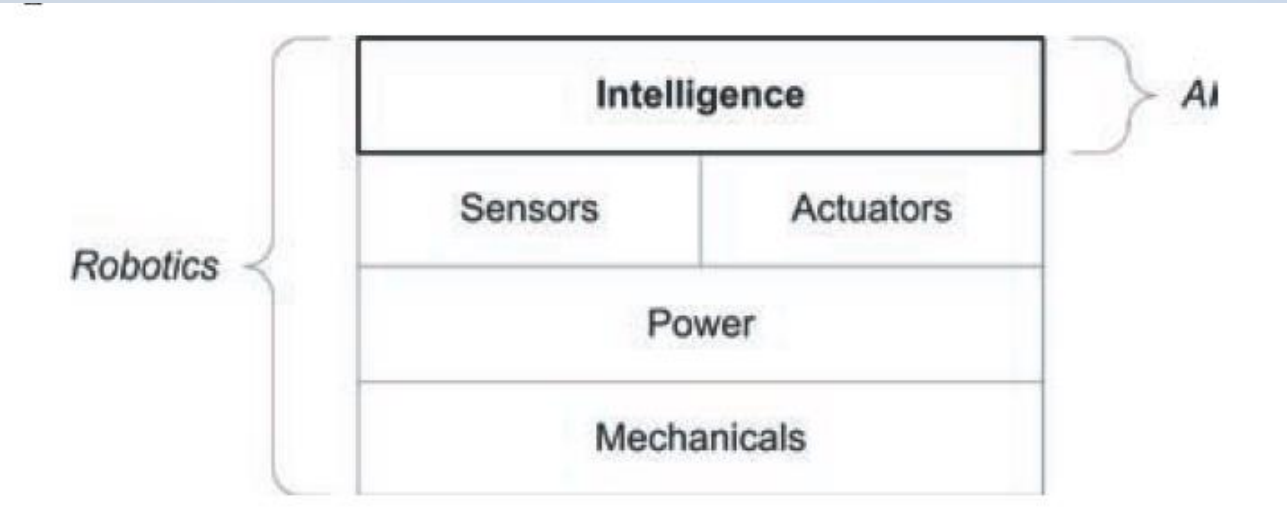
- učenje je prilagoditev okolju
- večagentno učenje:
  - centralizirano (centralni sistem se uči od vseh agentov)
  - decentralizirano: agenti se učijo vsak zase in od drugih, večja prilagodljivost sistema

Robotski agenti

- realno okolje je bolj zahtevno, nepredvidljivo
- zahteva po obvladanju negotovosti
- industrijski roboti
- roboti raziskovalci (Mars, avtonomija, premikanje, zgled insektov)

Robotika z vidika umetne inteligence

- testno okolje za algoritme umetne inteligence



Taksonomija

- fiksni (npr. industrijski, robotska roka z več prostorskimi stopnjami)
- z nogami (1,2,4,6,8)
- na kolesih
- podvodni (ribe, raki, črvi)
- zračni (avtomatska letala, sateliti)
- drugi (polimorfni, jate)
- fizični in programski (softboti)

Senzorji

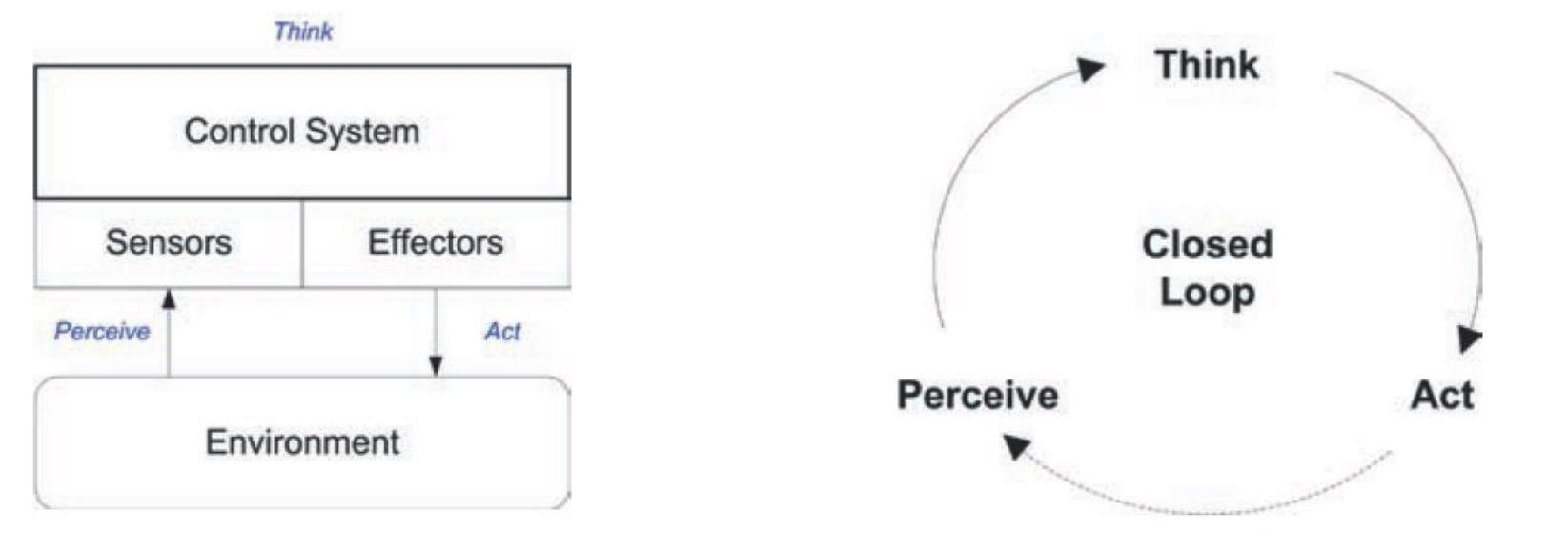
- vid (elektromagnetni valovi)
- sluh (zračni valovi)
- okus in vonj (kemični receptorji)
- tip (pritisk)
- eholokacija (ulrazvok)
- elektrocepcija (električno polje)
- magnetocepcija (spremembe v magnetnem polju)
- ekvilibriocepcija (ravnotežje, pospešek)
- termocepcija (temperatura)
- ...

Aktuatorji

- kolesa, noge, ...
- prijemala, robotske roke

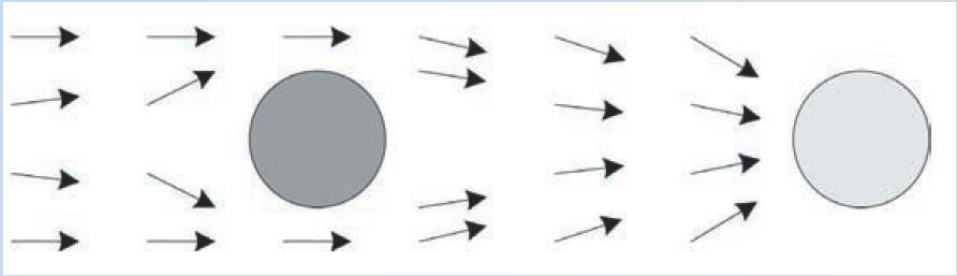
Kontrolni sistem

- reaktivni
- arhitektura vsebovanosti
- nevronske mreže, evolucijski pristopi, ...



Načrtovanje

- bistveno za inteligenco
- načrtovanje gibanja v dinamičnem okolju
- anytime načrtovanje, postopno izboljševanje načrta s časom, a načrt je vedno pripravljen
- dekompozicija na celice
- polje potenciala
- značilne točke
- graf vidljivosti



Pripomočki

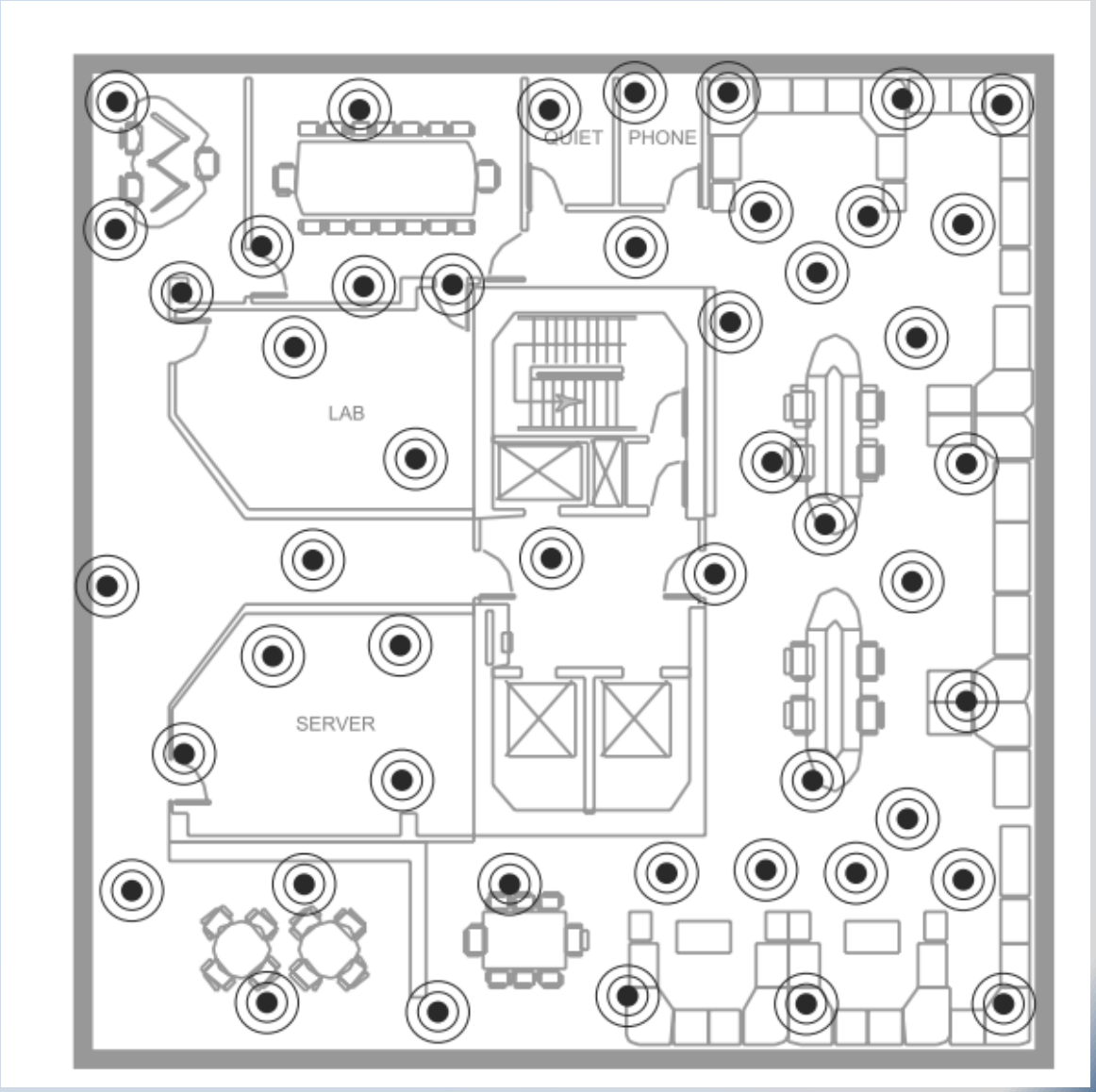
- robotski jeziki
- robotski simulatorji
- okolja, operacijski sistemi, ...

Distribuirano računanje z agenti

- agenti sodelujeju pri skupnem cilju, ki ga definira center
- agenti so samostojni, komunicirajo v soseščini
- poskušajo najti rešitev, ki upošteva globalne omejitve
- naloga: kakšen algoritem naj izvajajo agenti, da bo center dobil globalno informacijo

Primer: mreža senzorjev

- primer: mreža senzorjev (omejene zmožosti procesiranja, komunikacija le v soseščini, želimo globalno sliko)

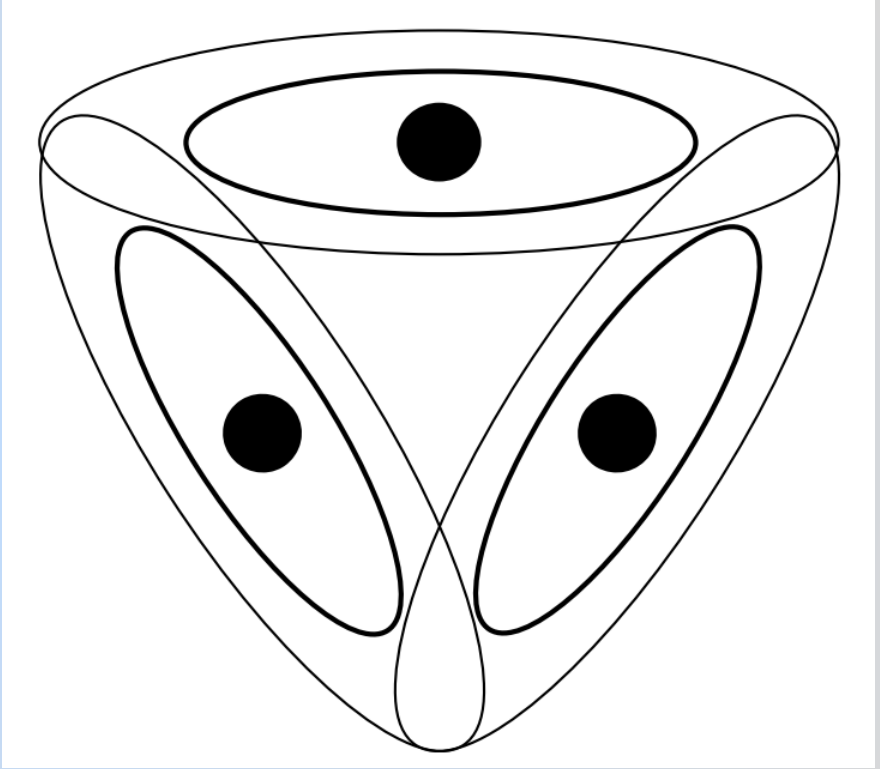


Upoštevanje omejitev

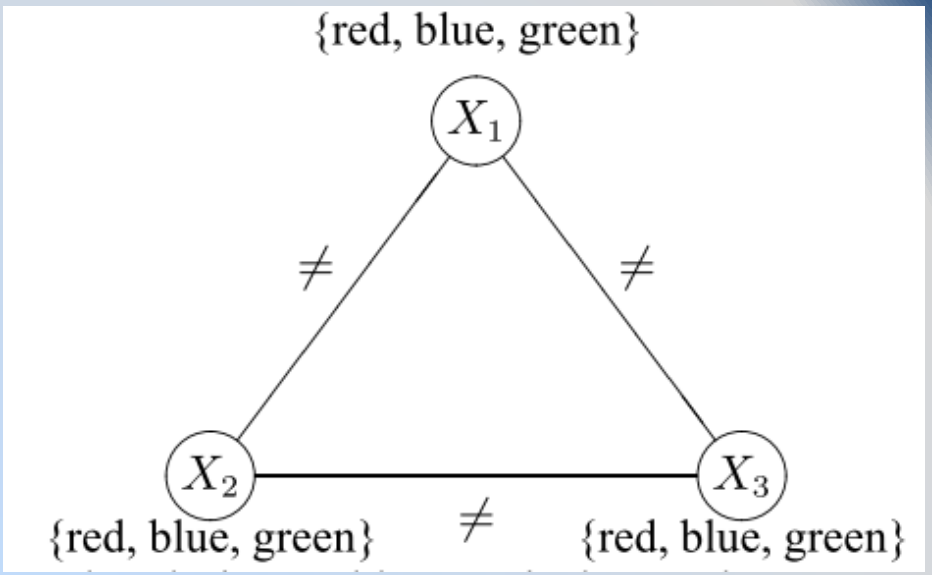
- constraint satisfaction problem
- problem iskanja z omejitvami je definiran z množico spremenljivk, njihovimi domenami in omejitvami glede vrednosti, ki jih lahko spremenljivke zavzamejo
- naloga je spremenljivkam prirediti vrednosti tako, da so upoštevane vse omejitve, ali pa ugotoviti, da to ni mogoče
- primeri: računalniški vid, razumevanje jezika, planiranje, dokazovanje teoremov, razporejanje opravil

Primer: iskanje frekvence za senzorje

- trije senzorji
- doseg signala se prekriva
- naloga: vsakemu prirediti frekvenco iz nabora treh dovoljenih tako, da se ne prekrivajo



Frekvence senzorjev



- omejitve predstavimo kot problem barvanja grafov
- množica spremenljivk $X=\{X_1, X_2, X_3\}$
- domena D_i za vsako spremenljivko je množica {red, blue, green}
- množica omejitev $\{X_1 \neq X_2, X_1 \neq X_3, X_3 \neq X_2\}$

Upoštevanje omejitev: terminologija

- prireditev spremenljivk
  - legalna
  - nelegalna
- rešitev
- porazdeljeno upoštevanje omejitev: vsak agent je svoja spremenljivka, rešitev naj določijo brez centralnega nadzora

Algoritmi z rezanjem domen

- vozlišča komunicirajo s sosedi, da odstranijo prepovedane vrednosti iz svojih domen
- algoritem (arc consistency)
- vsako vozlišče X_i z domeno D_i izvaja naslednji program za vse svoje sosede X_j

```
void revise(x_i, x_j ) {
  foreach ( v_i ∈ D_i )
    if (ne obstaja v_j ∈ D_j konsistentno z v_i )
      D_i = D_i – {v_i}
}
```

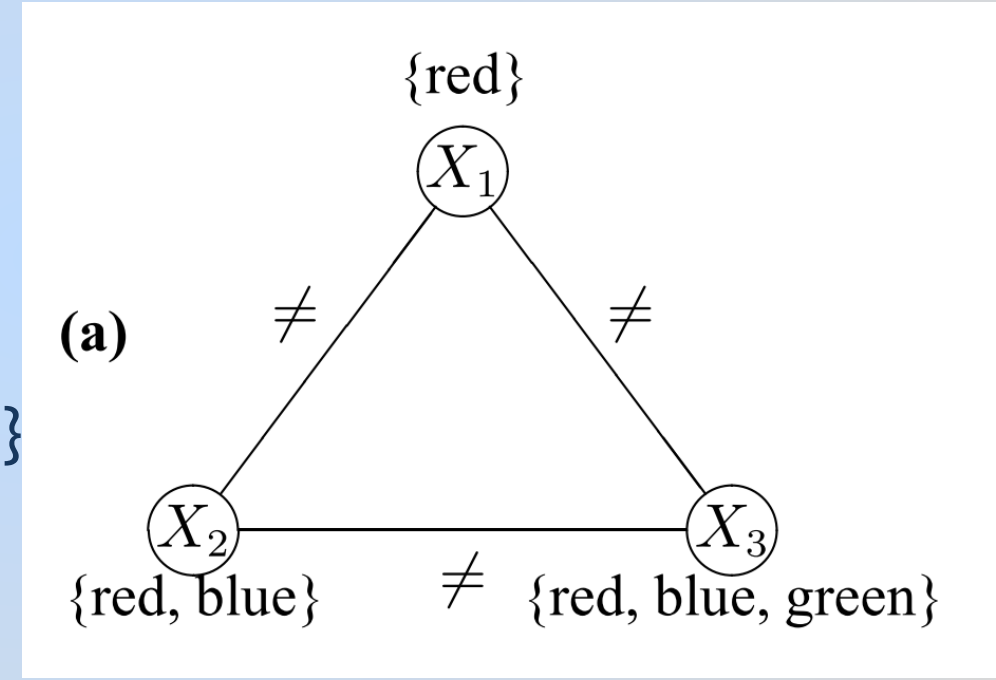
- končamo, ko je ena od domen prazna (ni rešitve), ali ko ni več eliminacij
- če ostane v domeni le ena vrednost imamo rešitev, sicer ne vemo ali obstaja rešitev
- algoritem se konča in je pravilen, ni pa popoln (ni zanesljivo, da najdemo rešitev)

Primer rezanja domen a)

- najprej so učinkovita samo sporočila vozlišču X1, zato in X_2 ter X_3 odstranita *red* iz svoje domene

X2={blue} X3={blue, green}

- nato še X_3 dobi sporočilo od X_2 in zato odstrani blue;
  dobimo pravilen rezultat

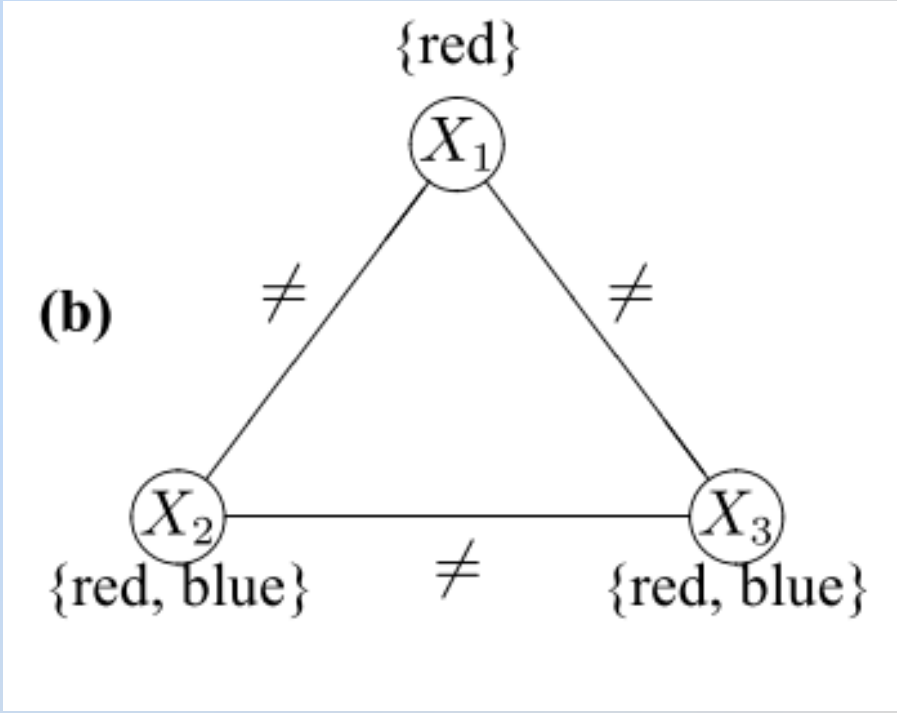


Primer rezanja domen b)

- kot prej: X_2 in X_3 odstranita *red* iz svoje domene

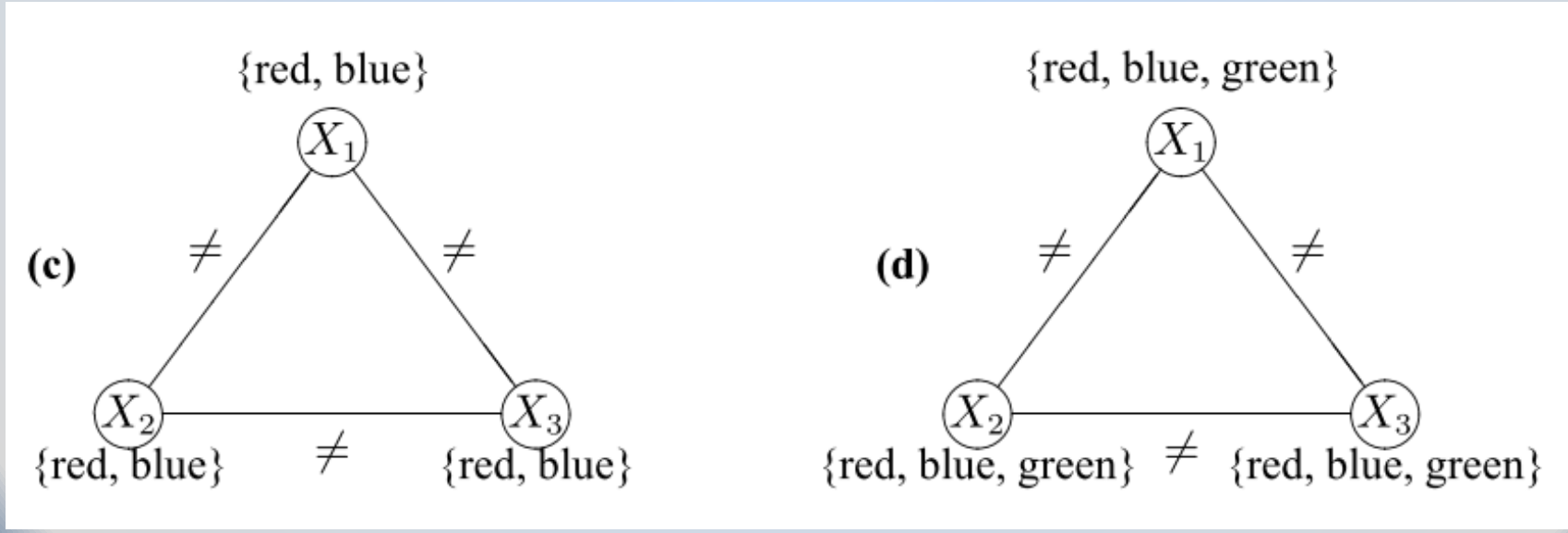
X2={blue} X3={blue}

- nato oba, X2 in X_3, odstranita še *blue;* ostane prazna domena, zato pravilno razglasita, da ni rešitve



Primer rezanja domen c), d)

- nobeno od vozlišč ne more odstraniti nobene vrednosti
- algoritem konča brez zaključka (niti nima rešitve, niti ne ve, da ni rešitve)



Ekvivalentnost logični resoluciji

- rezanje domen je prešibko, služi lahko kot predprocesiranje bolj zahtevnemu algoritmu
- eliminacija je ekvivalentna resoluciji v propozicijski logiki
- pravilo sklepanja

  A_1

  $\neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$

  ———————————

  $\neg(A_2 \wedge \ldots \wedge A_n)$

- zapišemo omejitve v obliki logičnih pravil Nogood, npr. x_1=red $\wedge$ x_2=red

  x_1=red

  $\neg(x_1$=red $\wedge$ x_2=red)

  ———————————

  $\neg(x_2$=red)

Hiperresolucija

- posplošitev resolucije

$$A_1 \vee A_2 \vee \cdots \vee A_m$$
$$\neg(A_1 \wedge A_{1,1} \wedge A_{1,2} \wedge \cdots)$$
$$\neg(A_2 \wedge A_{2,1} \wedge A_{2,2} \wedge \cdots)$$
$$\ldots$$
$$\neg(A_m \wedge A_{m,1} \wedge A_{m,2} \wedge \cdots)$$
$$\text{———————————————}$$
$$\neg(A_{1,1} \wedge \cdots \wedge A_{2,1} \wedge \cdots \wedge A_{m,1} \wedge \cdots)$$

- pravilna in popolna za propozicijsko logiko

Algoritem s hiperresolucijo

- vsak agent generira nove omejitve za svoje sosede, jih o tem obvešča in reže svojo domeno glede na omejitve, ki jih dobi od sosedov
- NG_i je množica omejitev (Nogoods), ki se jih agent *i* zaveda
- NG^*_j množica novih omejitev, ki jih agent i dobi od agenta *j*

void reviseHR(NG_i, NG^*_j) {

do {

$NG_i \leftarrow NG_i \cup NG^*_j$

$NG^*_i \leftarrow$ hiperresolucija(NG_i, D_i)

if ($NG^*_i \neq \{\}$)

$NG_j \leftarrow NG_i \cup NG^*_i$

pošlji Nogoods NG^*_i vsem sosedom vozlišča i

if ($\{\} \in NG^*_i$)

stop

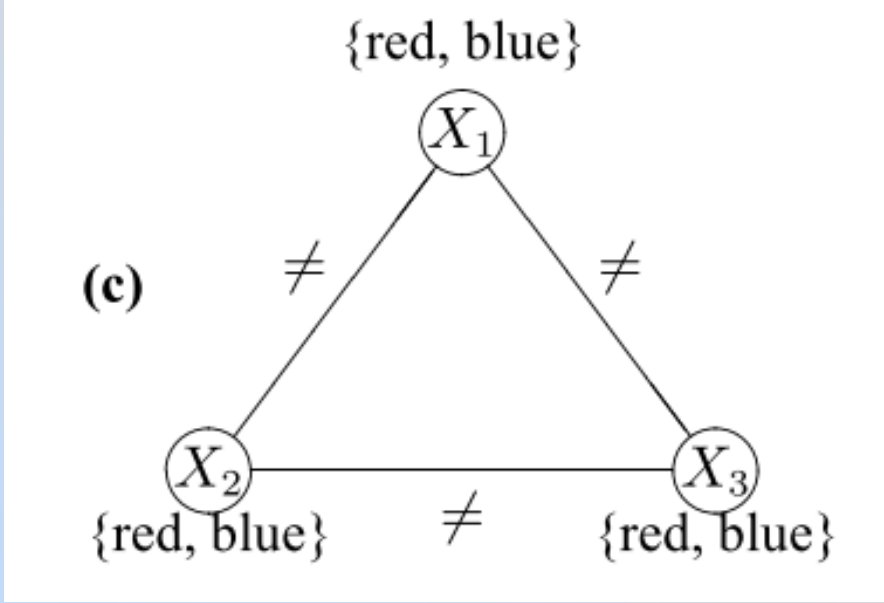
} **while** (sprememba v NG_i)

}

algoritem po končnem številu poslanih sporočil konča izvajanje

če rešitev obstaja, jo najde

Hiperresolucija za c)

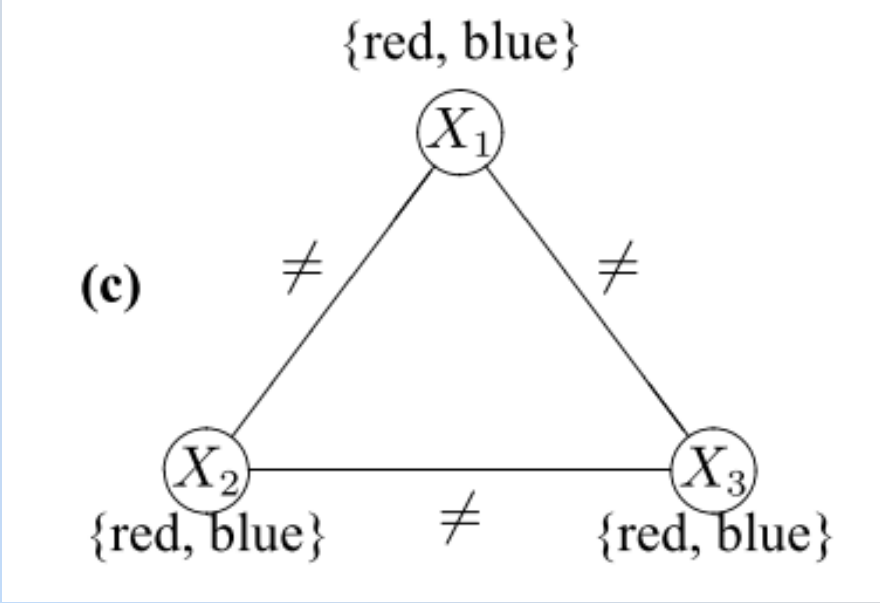


- X_1 ima naslednje omejitve v svoji množici Nogoods:
  {x_1 = red, x_2 = red},
  {x_1 = red, x_3 =red},
  {x_1 = blue, x_2 = blue},
  {x_1 = blue, x_3 = blue}
- velja tudi, da ima X_1 vrednost: x_1 = red ∨ x_1 = blue.
- s hiperresolucijo lahko X_1 sklepa

  x_1 = red ∨ x_1 = blue
  ¬(x_1 = red ∧ x_2 = red)
  ¬(x_1 = blue ∧ x_3 = blue)
  ———————————
  ¬(x_2 = red ∧ x_3 = blue)

in doda {x2 = red, x3 = blue} k svojim Nogoods

Hiperresolucija za c)



- podobno k Nogood doda tudi {x_2 = blue, x_3 = red}.
- x_1 pošlje oba Nogoods svojima sosedoma x_2 in x_3
- x_2 lahko na podlagi svoje domene, svojih Nogood in poslanega sklepa
  x_2 = red ∨ x_2 = blue
  ¬(x_2 = red ∧ x_3 = blue)
  ¬(x_2 = blue ∧ x_3 = blue)
  ———————————
  ¬(x_3 = blue)
- na podlagi drugega prejetega x_2 izpelje tudi ¬(x_3 = red)
- ko pošlje ta dva Nogood sosedu x_3, x3 generira {} in algoritem se konča z ugotovitvijo, da rešitve ni

Slabosti hiperresolucije

- močnejša kot resolucija, vendar lahko število generiranih Nogoods zelo naraste
- pri vzporednem procesiranju bi dobili še precej več sporočil in omejitev
- procedure delujejo previdno: sklepajo samo na to, kar je dokazljivo
- alternativa: nedokazljive predpostavke in vračanje

Hevristično iskanje rešitev z omejitvami

- enostavna rešitev je centralno vodeno dogovarjanje in sortiranje spremenljivk
- spremenljivke sortiramo, npr. $x_1, x_2, \ldots, x_n$, in pokličemo funkcijo chooseValue(x_1, {}), pri tem so vrednosti $\{v_1, v_2, \ldots, v_{i-1}\}$ že določene spremenljivkam $\{x_1, x_2, \ldots, x_{i-1}\}$

void ChooseValue(xi, $\{v_1, v_2, \ldots, v_{i-1}\}$) {
  $v_i \leftarrow$ vrednost $v \in D_i$ konsistentna z $\{v_1, v_2, \ldots, v_{i-1}\}$)
  if (v_i ne obstaja)
    vračanje
  else if (i == n)
    stop
  else
    chooseValue(x_{i+1}, $\{v_1, v_2, \ldots, v_i\}$)

}

- kronološko vračanje: vrnemo se en nivo nazaj in izberemo še ne uporabljeno konsistentno vrednost
- izčrpno preiskovanje, agenti se kličejo sekvenčno

Naivna paralelna asinhrona rešitev

- vsi agenti hkrati izvajajo naslednjo kodo

```
izberi vrednost iz svoje domene
do {
   if (lastna vrednost je konsistentna z vrednostmi sosedov OR
         nobena vrednost ni konsistenta )
      ne stori ničesar
   else {
     izberi vrednost iz svoje domene, ki je konsistentna s sosedi
     obvesti sosede o izbrani vrednosti
   }
} while (so še spremembe vrednosti)
```

pravilna, a nepopolna rešitev: lahko nikoli ne konča, lahko ne najde prave rešitve

Asinhrono vračanje

- potrebujemo močnejši algoritem, ki vsebuje elemente prejšnjih: globalno urejenost in izmenjavo sporočil
- ABT (asynchronous backtracking)
- agenti imajo prioriteto (urejenost), sporočila potujejo od agentov z višjo k agentom z nižjo prioriteto
- agenti delujejo paralelno, vrednost postavijo hkrati, obvestijo o vrednostih agente, s katerimi so povezani
- vsi agenti čakajo na sporočila in nanje odgovarjajo
- ko agent spremeni svoje stanje, pošlje svojo novo vrednost vsem povezanim
- ko agent prejme sporočilo o nastavitvi od agenta z višjo prioriteto, poskuša svojo vrednost prilagoditi tako, da ne krši prejetih omejitev

Asinhrono vračanje - komunikacija

- agenti sporočajo vrednosti s sporočili ok?
- ko agent A_i prejme ok? sporočilo agenta A_j , shrani prejete vrednosti v podatkovno strukturo agent_view
- preveri, če njegova trenutna vrednost še ustreza trenutnemu agent_view.
- če ustreza, ne stori ničesar, sicer v svoji domeni poišče novo konsistentno vrednost
- če jo najde, jo priredi svoji spremenljivki in pošlje ok? sporočilo vsem povezanim agentom z nižjo prioriteto, sicer začne vračanje

Asinhrono vračanje - vračanje

- vračanje se začne s sporočilom Nogood
- Nogood vsebuje nekonsistentne delne prireditve
- Nogood za A_i je torej kar njegova vrednost agent_view
- A_i pošlje Nogood agentu z najnižjo prioriteto med tistimi,ki imajo že prirejene vrednosti
- A_i, ki pošlje Nogood agentu A_j, predvideva, da bo A_j spremenil svojo vrednost in iz svojega agent_view zbriše prireditev za A_j ter ponovno poskuša najti konsistentno prireditev

Asinhrono vračanje: lastnosti

- požrešna verzija hiperresolucije
- namesto pošiljanja vseh omejitev vsem agentom, agent poiskuša zadovoljiti svoje omejitve in v Nogood vključi le vrednosti, ki so jih priredili agenti nad njim
- nove vrednosti sporoči le nekaj agentom, omejitve v Nogood pa le enemu

Psevdokoda za agenta A_i

// upoštevanje omejitev s postopkom asinhronega vračanja
// kodo izvaja vsak agent A_i vzporedno in asinhrono

```
void ABT() {
  when sprejeto(Ok?, (Aj , dj )) {
    add (Aj , dj ) to agent_view
    check_agent_view()
  }

  when sprejeto(Nogood, nogood) {
       dodaj nogood  v seznam Nogood
       foreach (Ak, dk) ∈ nogood {
          if Ak ni sosed vozlišča Ai {
             dodaj (Ak, dk) v agent_view
             zahtevaj od Ak da doda Ai kot soseda
             // soseda moramo dodati, da bomo obveščeni o njegovih spremembah
          }

       }
      check_agent_view()

  }

}
```

Preverjanje konsistentnosti

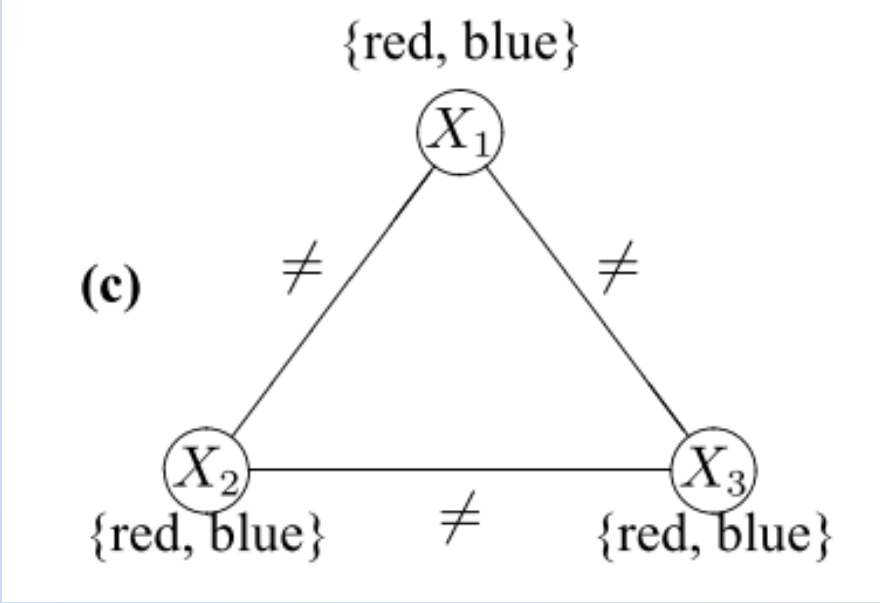
```
void check_agent_view() {
    if  ( not konsistentna(agent_view , current_value) ) {
      if  (ne obstaja z agent_view  konsistentna vrednost v D_i)
          backtrack
      else {
        izberi  d ∈ D_i konsistentno z agent_view
        current_value ← d
        pošlji(ok?, (A_i, d)) sosedom z nižjo prioriteto
     }
    }

}
```

Vračanje

void backtrack() {
nogood ← nekonsistentna množica, ki smo jodobili npr. s hiperresolucijo
if (nogood vsebuje {})
sporoči vsem agentom, da ni rešitve, končaj
else {
izberi (A_j , d_j) ∈ nogood AND
A_j ima najnižjo prioriteto v nogood
pošlji (Nogood, nogood) agentu A_j
odstrani (A_j , d_j) iz agent_view
check_agent_view()
}

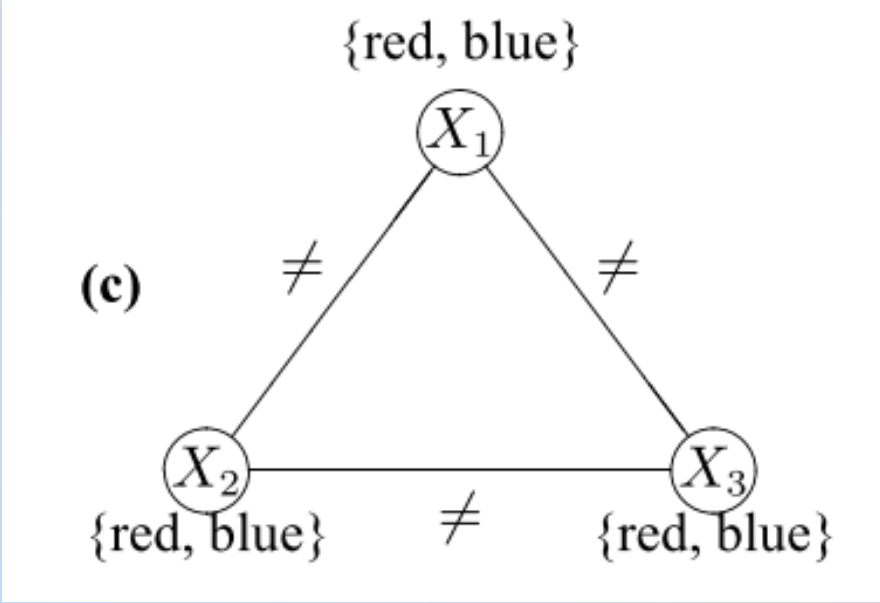
}

Asinhrono vračanje za c)



- denimo, da so agenti urejeni po prioriteti x_1, x_2, x_3
- na začetku izberejo naključno vrednost, npr. vsi "blue"
- x_1 obvesti x_2 in x_3 o svoji izbiri, x_2 obvesti x_3
  x_2 doda v svoj agent_view {x_1=blue}, x_3 doda {x_1=blue, x_2=blue}.
- x_2 in x_3 morata preveriti konsistentnost z lastno vrednostjo
  - X_2 ugotovi konflikt, spremeni svojo vrednost v "red" in obvesti x_3
  - v tem času tudi x_3 ugotovi konflikt, spremeni vrednost na "red", a nikogar ne obvesti
  - x_3 sprejme drugo sporočilo od x_2 in popravi svoj agent_view na {x_1 = blue, x_2 = red}.

Asinhrono vračanje za c)



- x_3 ne more najti konsistentne vrednosti zato uporabi hiperresolucijo in generira Nogood {x_1 = blue, x_2 = red}
- ta Nogood pošlje x_2, ker ima ta najnižjo prioriteto v Nogood
- zdaj x_2 ne more najti konsistentne vrednosti in generira Nogood {x_1 = blue} ter ga pošlje v x_1.
- x_1 ugotovi nekonsistentnost in spremeni svojo vrednost na "red" ter pošlje sporočilo o tem x_2 in x_3
- tako kot prej, x_2 spremeni svojo vrednost na blue, x_3 ne najde konsistentne vrednosti in generira Nogood {x_1 = red, x_2 = blue}, nakar x_2 generiraNogood {x_1 = red} in ga pošlje x_1
- v tem trenutku ima x_1 Nogood {x_1 = blue} in {x_1 = red}, uporabi hiperresolucijo in zgenerira Nogood {}. Algoritem se ustavi z ugotovitvijo, da ni rešitve.

Porazdeljena optimizacija

- Naloga: agenti naj **skupaj** najdejo optimalno vrednost globalne funkcije.
- Primer: iskanje najcenejših poti v usmerjenem grafu z n vozlišči in m povezavami,
- povezavi (a,b) je pripisana cena $c(a,b)$; naloga je najti pot z najmanjšo vsoto cen povezav od začetnega vozlišča s do enega od končnih vozlišč $t \in T$.
- Uporaba: za telekomunikacijske in transportne probleme, planiranje
- Razlika s standardnimi pristopi k iskanju najkrajših poti (Dijkstra, Bellman-Ford) je porazdeljenost (vsak agent izračuna del najboljše rešitve in komunicira le z agenti v svoji okolici)

Asinhrono dinamično programiranje

- Ideja: princip dinamičnega programiranja
- opazimo: če leži vozlišče *x* na optimalni poti od *s* do *t*, potem je optimalen tudi del poti od *s* do *x*, kakor tudi del poti od *x* do *t*
- optimalno rešitev gradimo postopno od spodaj navzgor
- najkrajšo razdaljo od vozlišča *i* do končnega vozlišča *t* označimo s $h^*(i)$.
- najkrajšo pot od *i* do *t*, ki gre preko sosednjega vozlišča *j* , lahko zapišemo
  $f^*(i, j) = c(i, j)+h^*(j)$
- najkrajšo pot iz *i* preko poljubnega sosednjega vozlišča pa

  $h^*(i) = \min_j f^*(i, j)$

Opis delovanja

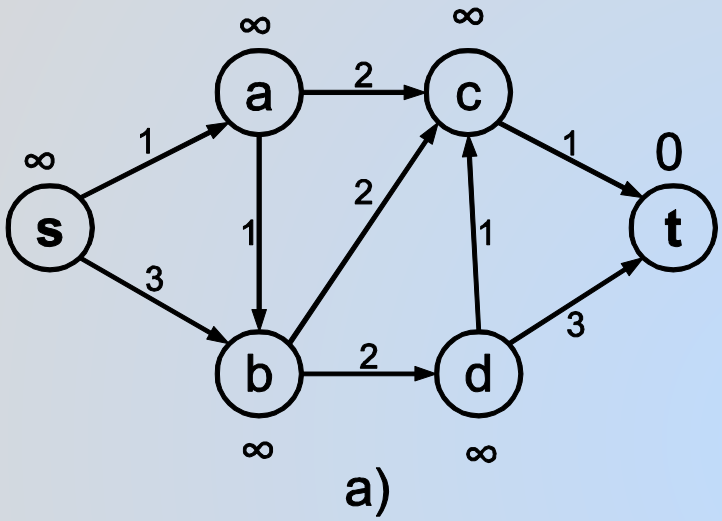
- vsako vozlišče *i* hrani vrednost *h*(*i*), ki je približek *h**(*i*)
- inicializacija h(i)=∞,
- med izvajanjem se vrednosti h(i) zmanjšujejo in konvergirajo k pravi vrednosti *h**(*i*)
- konvergenca zahteva en korak za vsako vozlišče na najkrajši poti, kar pomeni, da bo v najslabšem primeru potrebno *n* korakov
- Slabost: potrebujemo agenta za vsako vozlišče grafa, kar je v primeru ogromnih grafov (npr. šah) nesprejemljivo

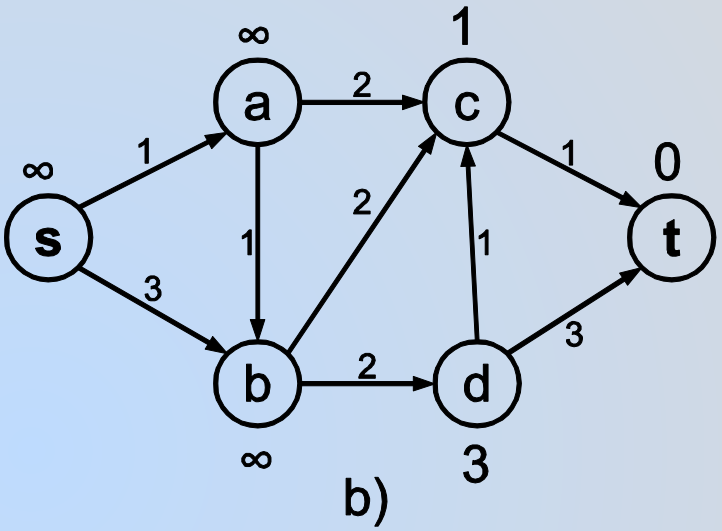
Psevdokoda ADP za iskanje najkrajših poti

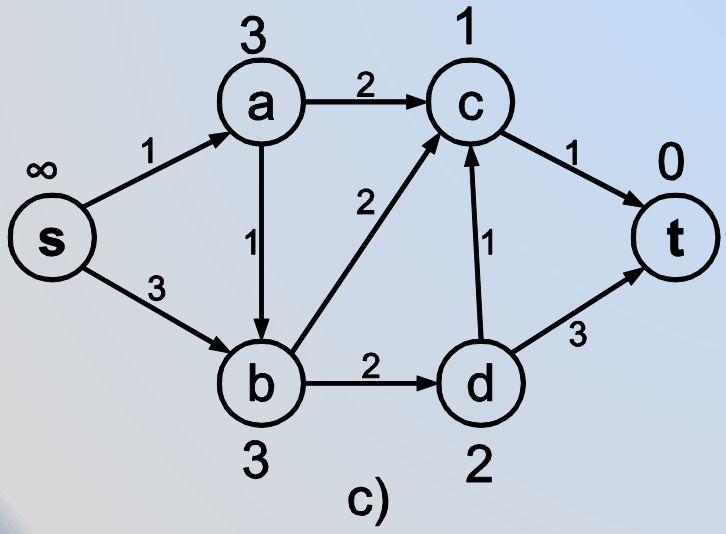
// algoritem se izvaja na vsakem od vozlišč i danega grafa

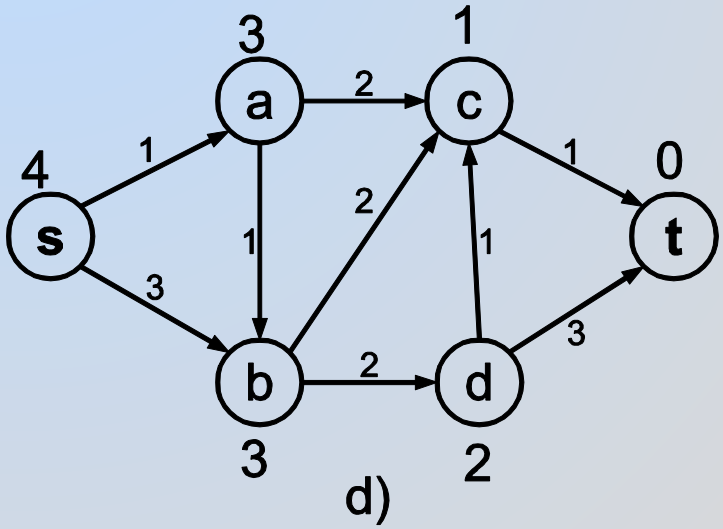
void ADP(Vozlišče i) {
  if (i je ciljno vozlišče)
    $h(i) = 0$; *// točna vrednost*
  else
    $h(i) = \infty$; *// inicializacija*
  do{
    foreach (vozlišče *j* sosedno *i*)
      $f(j) = c(i, j)+h(j)$;
    $h(i) = \min_j f(j)$;
  } **while** (true) ;
}

Ilustracija ADP









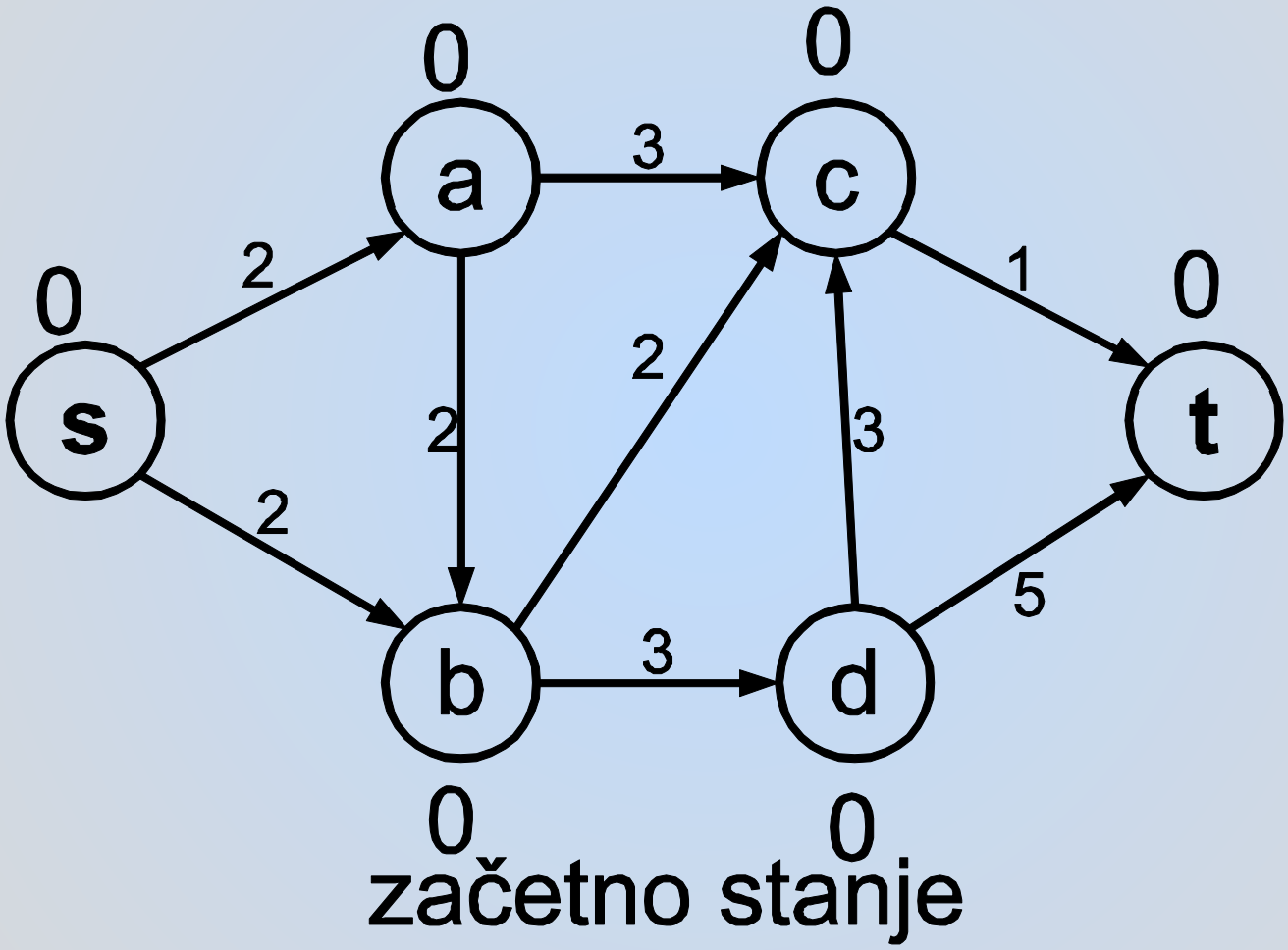
*LRTA**

- algoritem *LRTA** (*learning real-time A**) uporablja enega ali več agentov za iskanje najkrajših poti.
- Inicializacija: h(i)=o
- Agent izvede algoritem večkrat in popravlja vrednosti h(i)
- Primer: izvajanje enega agenta

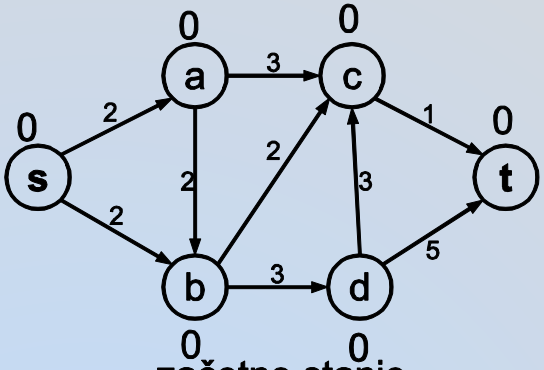
Psevdokoda algoritma LRTA*

```
// algoritem lahko izvaja eden ali več agentov
void LRTA*() {
  i = s ; // začetno vozlišče
  while ( i ni ciljno vozlišče ) {
    foreach ( vozlišče j sosedno i )
      f ( j) = c(i, j)+h( j) ;
    b = argmin_j f ( j) ;     // med enakima vozliščema izberi naključno
    h(i) = max(h(i), f (b)) ;
    i = b ;
  }
}
```

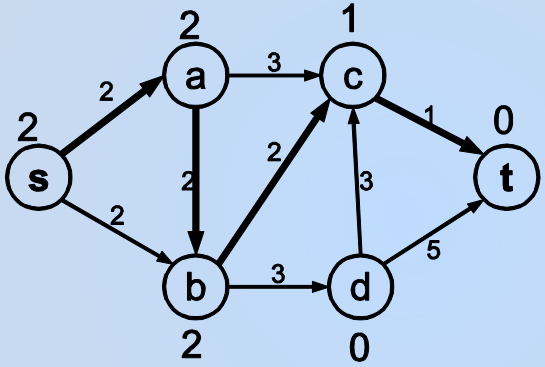
Primer:



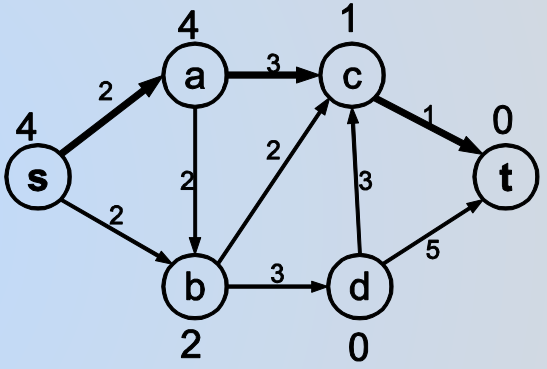
Izvajanje LRTA* enega agenta



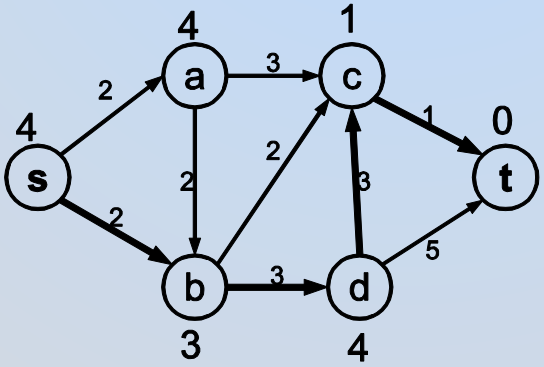
začetno stanje



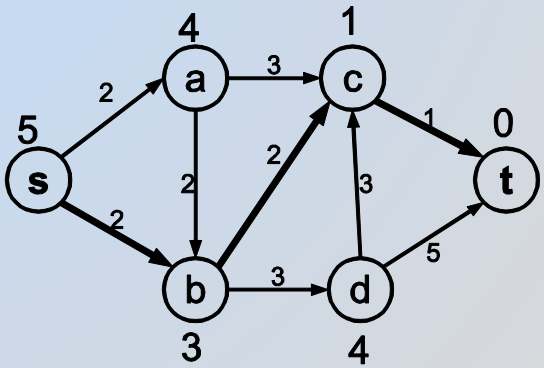
1. poskus



2. poskus

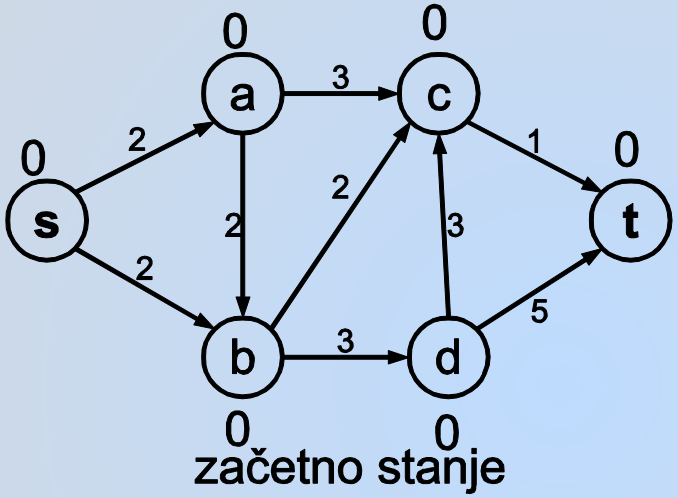


3. poskus

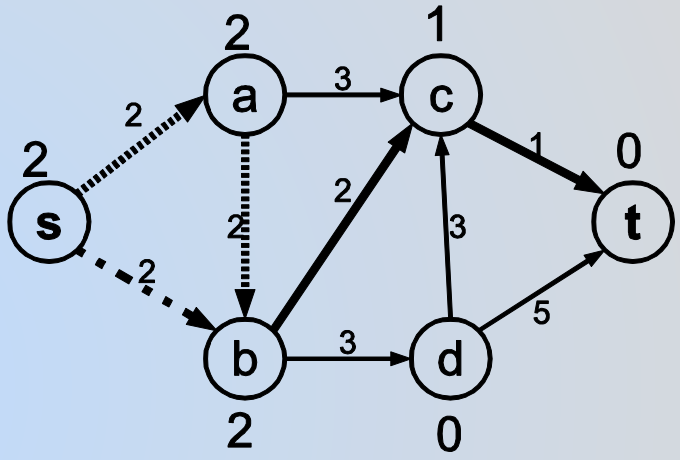


4. poskus

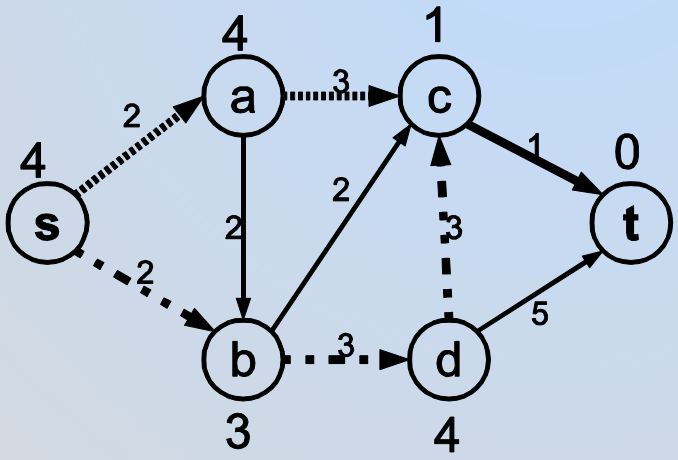
Izvajanje LRTA*(2) (dva agenta)



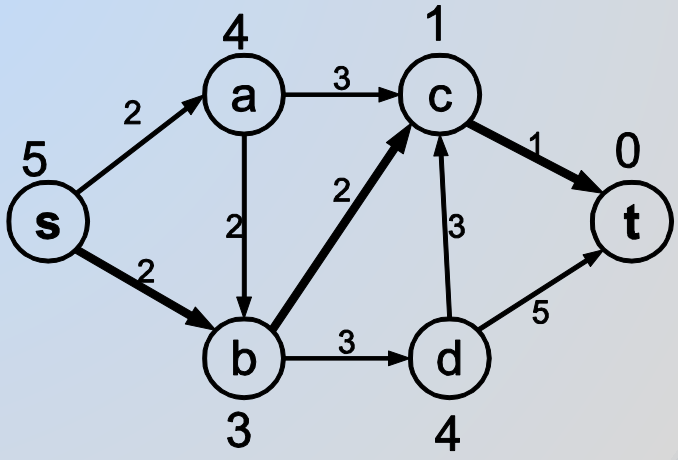
začetno stanje



1. poskus



2. poskus



3. poskus

Agentne tehnologije

- številne razširitve asinhronega vračanja
- optimizacija z upoštevanjem omejitev
- učeči se agenti
- področje teorije iger (kooperativne in nekooperativne igre)
- ...
- Več v Y. Shoham, K. Leyton-Brown; Multiagent Systems, Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press, 2009

Primer: izvajanje ADP in LRTA*