Pretrage s nedetrminističkim akcijama

Katica Babić, Anita Jukić, Manuela Pavić *

Osijek, 2012.

1 Istraživanje nedeterminističkih postupaka

Okoliš je potpuno vidljiv i determiniziran (predodređen) i agent (može i agens) zna koji su učinci svakog postupka (akcije). Prema tome, agent može izračunati koje stanje proizlazi iz svakog dijela postupka i uvijek zna u kojem je ono stanju. Predmeti opažanja ne pružaju nikakve nove informacije nakon bilo kojeg postupka iako govore agentu o početnom stanju.

Kada je okoliš djelomično vidljiv ili neodređen (nedeterminiziran) ili oboje, predmeti opažanja postaju korisni. U djelomično vidljivom okolišu svaki predmet opažanja pomaže suziti broj mogućih stanja u kojima bi agent mogao biti čineći tako da agentu bude lakše postići njegove ciljeve. Kada je okoliš nedeterminiziran, predmeti opažanja govore agentu koji se od mogućih ishoda njegovih postupaka zapravo pojavio.

U oba slučaja, budući predmeti opažanja ne mogu biti predodređeni unaprijed i budući postupci agenta ovisit će o tim budućim predmetima opažanja. Dakle, rješenje problema nije redoslijed nego plan mogućnosti (također poznat kao strategija) koji određuje to učiniti ovisno o tome koji su predmeti opažanja primljeni. U ovom dijelu ispitujemo slučaj nedeterminiranosti.

1.1 Nestalan svijet usisivača

Kao primjer upotrijebit ćemo svijet usisivača koji ćemo definirati kao problem pretraživanja. To se može oblikovati kao sljedeći problem:

1. Stanja

Stanje je određeno i lokacijom agenta i lokacijom nečistoće. Agent je na jednoj od dvije lokacije od kojih bi svaka mogla, ali ne mora sadržavati nečistoću. Nadalje, postoje 2 x 2 na 2 = B moguća stanja svijeta. Veći okoliš s n lokacijama ima n x 2 na n stanja.

2. Početno stanje

Svako se stanje može odrediti kao početno stanje.

*Odjel za matematiku, Sveučilište u Osijeku, Gajev trg 6, Osijek, Hrvatska, e-mail: kbabic@mathos.hr, ajukic@mathos.hr, mapavic@mathos.hr

- Akcije - u ovom jednostavnom okolišu svako stanje ima samo tri akcije *lijevo, desno i usiši*. Veći okoliši mogu uključivati i *gore/dolje*.

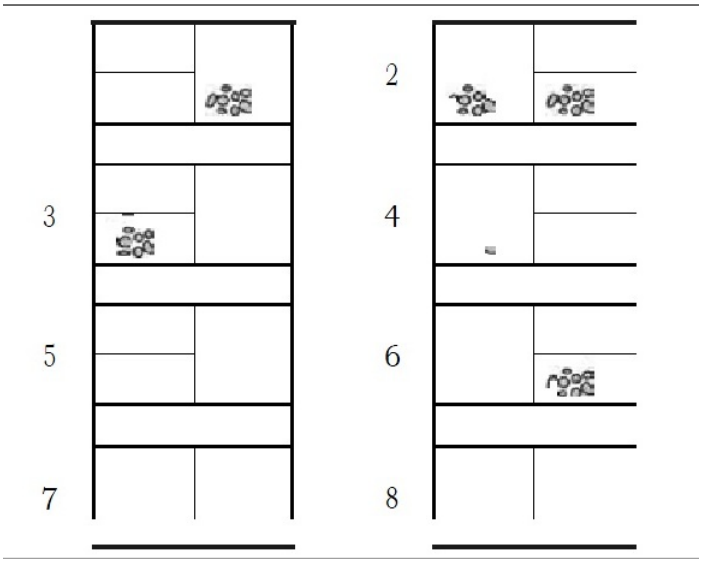
- Prijelazni model: akcije imaju očekivane učinke, osim pomicanja u *lijevo* u najljevijem kvadratiću, pomicanja u *desno* u najdesnijem kvadratiću i usisavanja u čistom kvadratiću - tada te akcije nemaju učinak.

- Test cilja: ovime se provjerava jesu li svi kvadratići čisti

- Vrijednost puta: svaki korak vrijedi 1, tako da je vrijednost puta broj koraka na putu

U usporedbi sa stvarnim svijetom ovaj problem ima posebnu lokaciju, posebnu nečistoću, pouzdano čišćenje i nikada se ne zaprlja više.

Prisjetimo se da *prostor stanja* ima osam stanja kao to je prikazano na slici 1.



Slika 1.

Postoje tri akcije: *lijevo, desno i usiši*, a cilj je počistiti svu prljavštinu (nečistoću). Ako je okoliš vidljiv, determiniziran i potpuno poznat onda je problem lako rješiv pomoću bilo kojeg algoritma pretraživanja i rješenje je postupni niz (postupak redoslijeda). Na primjer, ako je početno stanje 1 onda će postupni niz (usiši, desno, usiši) postići željeni cilj.

Sada pretpostavimo da upoznajemo nedeterminizam u obliku snažnog, ali nestalnog usisavača. U nestalnom svijetu usisivača akcija usiši radi na sljedeći način:

1. kada se primijeni na onečišćeni kvadratić akcija čisti kvadratić i ponekad čisti prljavštinu u susjednom kvadratiću također
2. kada se akcija rabi na čistom kvadratiću ponekad ostavlja nečistoću na tepihu

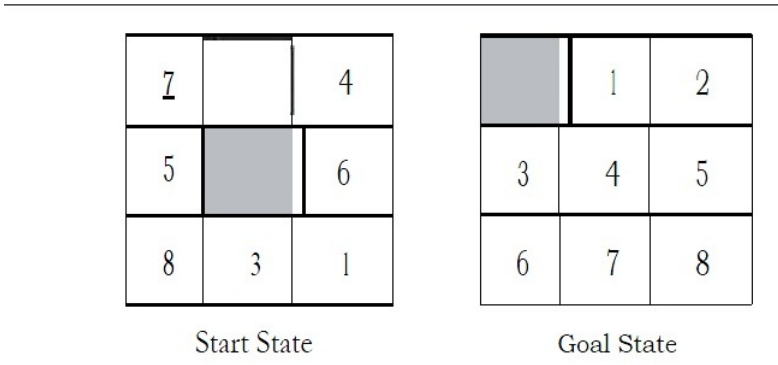
Umjesto da definiramo model prijenosa kao ishodnu funkciju koja vraća jedno stanje, upotrebljavamo funkciju ishoda (znači više njih) koja vraća skup mogućih stanja ishoda. Npr., u nestalnom svijetu usisivača akcija usiši u stanju 1 vodi do stanja u dijelu (5, 7) - nečistoća u desnom kvadratiću može i ne mora biti usisana.

Na primjer, ako počnemo od stanja 1 nema jednog redoslijeda akcija koji će riješiti problem. Umjesto toga trebamo sljedeći plan mogućnosti: (gurnuti ako je stanje = 5 , zatim desno, usiši ili pak)

Nadalje, rješenja za nedeterminizirani problem mogu sadržavati u sebi *if-then-else* izjave. To dopušta izbor akcija koje se temelje na pojavljivanju mogućnosti uslijed izvršavanja. Mnogi problemi u stvarnom, fizičkom svijetu jesu mogućnosti jer je točno predviđanje nemoguće. Iz tog razloga mnogi ljudi drže oči otvorenima kada hodaju naokolo ili voze.

1.2 Osmodijelne puzle

Instanca koja je prikazana na slici 2. sastoji se od 3x3 ploče s osam stupaca i praznih područja.



Slika 2.

Stupac koji je susjedan praznom području moe otklizati u prostor. Predmet je dosegnuti određeni cilj. Standardna formulacija je sljedeća:

1. Stanja

Opis stanja određuje lokaciju za svaki od osam elemenata i praznina u jednom od devet kvadratića.

2. Početno stanje
Svako se stanje može odrediti kao početno stanje. Svaki dani cilj može se doseći iz točno polovice mogućih početnih stanja.

- Akcije: Najjednostavnija definicija određuje akciju kao pomicanje praznina lijevo, desno, gore ili dolje. Drugačiji podsustavi su mogući ovisno o tome gdje se praznina nalazi.
- Prijelazni model: Kada je dano stanje i akcija tada se vraća dobiveno stanje. Npr., ako primijenimo lijevo na početno stanje na slici 2. dobiveno stanje bit će 5 i zamjena praznina.
- Test cilja: Ovime se provjerava podudara li se stanje s konfiguracijom cilja na slici 2.
- Vrijednost puta: svaki korak vrijedi 1, tako da je vrijednost puta broj koraka na putu.

Koja smo izlučivanja uključili ovdje? Akcije su određene svojim početnim i konačnim stanjima ignorirajući središnje lokacije gdje zastoj kliže. Imamo i akcije kao što su prodrmavanje hrpe kada dijelovi zapnu i izvlačenje dijelova nožem i vraćanje natrag. Ispustili smo opis pravila puzle izbjegavajući sve detalje fizičke manipulacije.

2 I-ILI stabla pretraživanja

Postavlja se sljedeće pitanje: Kako pronaći rješenja za nedeterminizirani problem?

Najprije ćemo graditi stablo pretraživanja koje će se razlikovati od stabala u dererminističkom okruženju.
U determinističkom okolišu jedino grananje otkriva se kroz agentove odluke u svakom stanju. Te čvorove zovemo ILI-čvorovi.

U našem primjeru, "svijetu usisavača", na ILI-čvoru agent usisavač izabire jednu od akcija: Lijevo, Desno ili Usiši.

U nedeterminiziranom okolišu grananje ovisi o izboru ishoda okoliša za svaku akciju. Ove čvorove nazivamo I-čvorovi.

Ove dvije vrste čvorova, ILI-čvorovi i I-čvorovi, se izmjenjuju čineći tako I-ILI stabla pretraživanja.

Rješenje za I-ILI problem je podstablo koje:

- ima ciljne čvorove na svakom listu
- određuje jednu akciju na svakom od ILI-čvorova
- uključuje svaku granu ishoda na kraju svakog I-čvora

Na I-čvorovima i granama upotrebljava se *if-then-else* notacija, ali ako ima više od dvije grane na čvoru, bilo bi bolje upotrijebiti *case* notaciju.

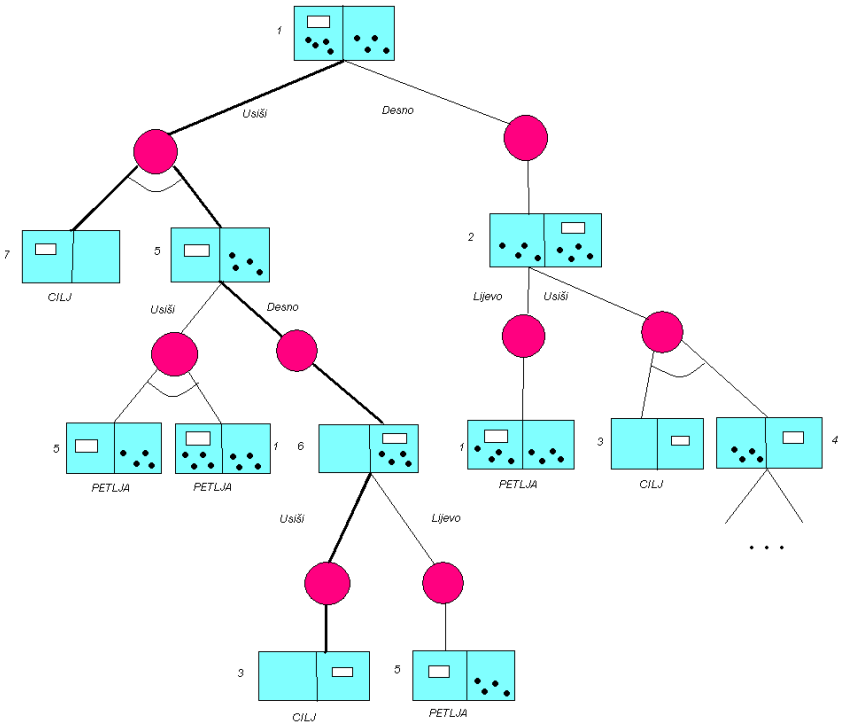
2.1 Algoritam

Pogledat ćemo na dva primjera izgled I-ILI stabla pretraživanja, kao i traženje rješenja.

1. PRIMJER:

Na Slici.3. vidimo prve dvije razine stabla pretraživanja u nedeterminiziranom svijetu usisavača.
Taj svijet sastoji se od dva kvadratića u kojima može i ne mora biti nečistoće, a agent usisavač nalazi se u jednom od kvadratića i njegov zadatak je očistiti oba kvadratića koristeći akcije: Lijevo (pomicanje ulijevo), Desno (pomicanje udesno), Usiši (čišćenje kvadartića).

Čvorovi stanja su ILI-čvorovi gdje neke akcije trebaju biti odabrane. Na I-čvorovima, koji su prikazani krugovima, svaki se ishod može prikazati povezanim lukovima izlaznih grana stabla.
Rješenje je prikazano podebljanim linijama.



Slika 3.

Već smo rekli da u nedeterminističkom okruženju akcija Usiši radi ovako:

- Ako ju primjenimo na onečišćeni kvadratić, ona ga očisti, a ponekad očisti i susjedni kvadratić.
  Vidimo na slici da, kada na stanje 1 primjenimo akciju Usiši, dobijemo I-čvor iz kojeg slijede dva ishoda:
  Stanje 5: usisan trenutni kvadratić
  Stanje 7: usisana oba kvadratića
  Ako su usisana oba kvadratića, riješili smo problem, tj. stanje 7 je i ciljno stanje (oba kvadratića čista). Ako nisu oba, očišćen je samo lijevi kvadratić i tada na dobivenom čvoru možemo provesti sljedeće akcije: Usiši ili Desno.

- Prilikom akcije Usiši na čistom kvadratiću usisavač može ostaviti nečistoću.
  Na slici vidimo, ako na čvor 5 primjenimo akciju Usiši (kvadratić je već čist), može se dogoditi jedan od sljedeća dva slučaja:

  1. Ne dogodi se ništa i tu se pojavljuje petlja jer se ponavlja stanje 5.

  2. Usisavač ostavi nečistoću i pritom se ponavlja stanje 1, dakle, ponovno imamo petlju.

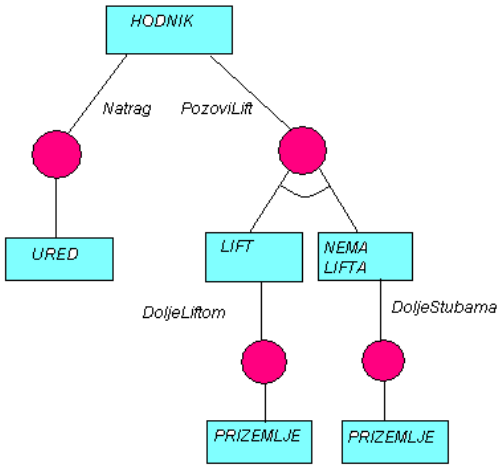
  Očito je da nas taj put ne može dovesti do rješenja. Ako pak na čvor 5 izvršimo akciju Desno, dolazimo u stanje 6 iz kojega opet možemo ići u dva smjera, tj. provesti jednu od akcija: Usiši ili Lijevo.
  Imamo:

  1. Primjenom akcije Usiši, dolazimo u ciljno stanje.

  2. Primjenom akcije Lijevo vraćamo se u stanje 5 i dobivamo petlju.

Osim dva navedena, postoji još rješenja danog problema. Na primjer, mogli smo iz stanja 1 izvršiti akciju Desno što bi dovelo do drugih rješenja.

2. PRIMJER:



Slika 4.

Promotrimo sljedeći problem:
Agent je izašao iz ureda na katu i nalazi se u hodniku u kojem su troja vrata: prva vode u ured, druga na stubište, a treća su vrata lifta. Zadatak agenta je sići u prizemlje u najkraćem vremenu. Akcije koje se provode su: Natrag (vratiti se u prošlo stanje), PozoviLift (pritisnuti gumb za pozivanje lifta), DoljeLiftom (spustiti se liftom u prizemlje) i DoljeStubama (spustiti se stubama do prizemlja).
Pogledajmo moguće rješenje koje je dano na slici.
Na našoj slici Agent se nalazi u hodniku. Može napraviti jednu od sljedečih akcija: Natrag, koja ga vodi u ured (nije put do rješenja) ili akciju PozoviLift. Prilikom pozivanja lifta može se pojaviti jedan od sljedeća dva ishoda (imamo I-čvor):

1. LIFT
   U ovom stanju izvršava se akcija DoljeLiftom koja vodi Agenta do ciljnog stanja PRIZEMLJE.

2. NEMA LIFTA
   U ovom slučaju Agent izvršava akciju DoljeStubama koja ga također vodi do ciljnog stanja PRIZEMLJE.

Ovaj primjer je vrlo jednostavan i navodim ga radi boljeg razlikovanja ILI-čvorova i I-čvorova.

2.2 Algoritam

Pogledajmo sada algoritam za traženje I-ILI grafova (stabala) u nedeterminističkoj okolini. Algoritam vraća uvjetni plan koji dolazi do željenog cilja u svim okolnostima.

ALGORITAM:

function AND-OR-GRAPH-SEARCH (*problem*) **returns** a conditional plan, or failure
 OR-SEARCH(*problem*.INITIAL-STATE,*problem*,[])

---

function OR-SEARCH(*state, problem, path*) **returns** a conditional plan, or failure
 if *problem*.GOAL-TEST(*state*) **then return** the empty plan
 if *state* is on *path* **then return** failure
 for each *action* **in** *problem*.ACTIONS(*state*) **do**
  plan ← AND-SEARCH(RESULTS(*state, action), problem,*[*state* | *path*])
  if *plan* ≠ *failure* **then return** [*action* | *plan*]
 return *failure*

---

function AND-SEARCH(*states, problem, path*) **returns** a conditional plan, or failure
 for each s_i **in** *states* **do**
  $plan_i$ ← OR-SEARCH(s_i,*problem, path*)
  if *plan* = *failure* **then return** *failure*
 return [if s_1 then $plan_1$ else if s_2 then $plan_2$ else . . . if s_{n-1} then $plan_{n-1}$ else $plan_n$]

Opišimo ukratko kako taj algoritam radi.
Najprije imamo funkciju koja traži I-ILI graf. Počinjemo s inicijalnim problemom - početnim stanjem.
Funkcija OR-SEARCH najprije provjerava je li trenutno stanje jednako ciljnom stanju. Ako jest, našli smo rješenje.
Ako je trenutno stanje jednako kao neko koje se već pojavilo na putu od korijena (početnog stanja), onda se vraća s neuspjehom (ovako izbjegavamo petlje).
Za svaku akciju može postojati više ishoda; zato za svaku akciju pravimo plan pomoću AND-SEARCH funkcije. Ona radi tako da na svaki mogući ishod trenutne akcije primjenjuje OR-SEARCH.
Dakle, na svakom od mogućih ishoda provjerava jesmo li došli u ciljno stanje, pojavljuje li se neko od stanja koja su već bila na putu i koje akcije se mogu nadalje primjeniti.
Funkcija AND-SEARCH vraća plan koji može dovesti do rješenja i na temelju nega funkcija OR-SEARCH izvršava određenu akciju.

3 Pronalaženje cilja

Algoritam za traženje AND-OR grafova u nedeterminističkoj okolini (1) daje **rekurzivan** $depth - first$ algoritam za AND-OR graf (2).

- **Pretraživanje u dubinu** (engl. depth-first search, DFS) je slijepa strategija istraživanja koja ne obilazi čvorove po razinama, nego najprije obilazi sve sljedbenike

nekog čvora, a tek nakon što se dođe do dna grafa, odnosno do čvora koji više nema nasljednika, pretraživanje se usmjerava na sljedeći čvor na istoj razini.

(1) Algoritam za traženje AND-OR grafova u nedeterminističkoj okolini:

function OR-SEARCH(*state, problem, path*) **returns** a conditional plan, or failure
 if *problem.*GOAL-TEST(*state*) **then return** the empty plan
 if *state* is on *path* **then return** failure
 for each *action* **in** *problem.*ACTIONS(*state*) **do**
  plan ← AND-SEARCH(RESULTS(*state, action), problem,*[*state* | *path*])
  if *plan* ≠ *failure* **then return** [*action* | *plan*]
 return *failure*

---

(2) Rekurzivan *depthfirst* algoritam za AND-OR graf:

function depthFirstSearch(*s, succ, goal*)
 if *goal(s)* **then return** s
 for $m \in succ(s)$ **do**
  $r \leftarrow$ depthFirstSearch(*m, succ, goal*)
  if $r \neq fail$ **then return** r
 return *fail*

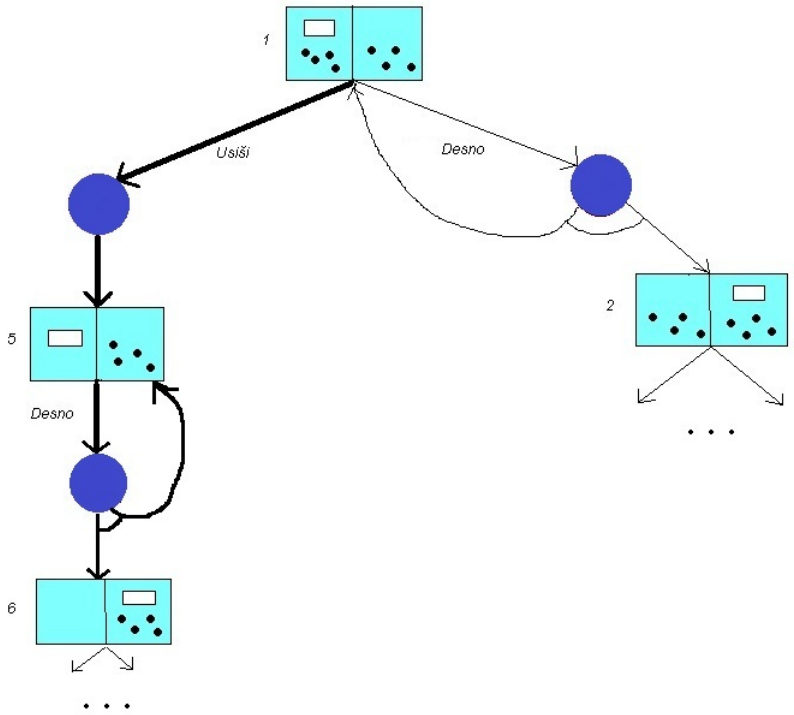
Jedan aspekt algoritma je način na koji se on nosi s ciklusima koji se često javljaju u nedeterminiziranim problemima. Ako je trenutno stanje identično kao na putu od korijena onda se vraća s neuspjehom (npr. ako akcija ponekad nema efekta ili ako neplanirani efekt može biti ispravljen). Ako je početno stanje identično stanju na putu od korijena, onda vraća pogršku. To ne znači da nema rješenja iz trenutnog stanja, jednostavno znači da ako postoji nekičko rješenje mora biti dostupno iz ranije inkarnacije trenutnog stanja tako da nova inkarnacija može biti odbačena.

Ovom provjerom osiguravamo da algoritam završava u svakom konačnom stanju jer svaki put mora doseći cilj, slijepu ulicu ili stanje koje se ponavlja.

Uočimo da algoritam ne provjerava je li trenutno stanje ponavljanje nekog drugog stanja na putu od korijena što je važno za učnkovitost.

AND-OR grafovi također mogu biti istraživani pomoću $breadth - first$ metode.

- **Pretraživanje u širinu** (engl. breadth-first search, BFS) je jednostavna slijepa strategija pretraživanja u kojoj se nakon ispitivanja korijenskog čvora, obilaze sva njegova djeca, a potom i sva njegova djeca itd.
  Ovdje će se prvo ispitati svi čvorovi jedne razine, a tek će se nakon toga, ne pronađe li se ciljni čvor, prijeći na ispitivanje čvorova na sljedećoj razini.



Slika 5.

Slika 5. prikazuje dio grafa pretraživanja za nesiguran svijet vakuuma gdje su eksplicitno prikazani neki ciklusi. Sva rješenja za ovaj problem ciklički su planovi jer ne postoji način za pouzdano kretanje.

Ako na stanje 1 primjenimo akciju Usiši, dobijemo AND-čvor iz kojeg slijedi stanje 5 gdje je usisan trenutni kvadratić. Očišćen je samo lijevi kvadratić i tada (na stanje 5) primjenjujemo akciju Desno i dolazimo u stanje 6 iz kojeg možemo ići u dva smjera. Da smo na stanje 5 primjenili akciju Usiši vratili bi se na stanje 5.
Ako na stanje 1 primjenimo akciju Desno, dobijemo AND-čvor iz kojeg slijedi stanje 2. Nepotrebno je promatrati akciju "pomakni se lijevo" jer će se opet vratiti gdje je i bio.

Razmatrajmo svijet vakuuma koji je identičan uobičajenom svijetu vakuuma osim toga što akcije pokreta ponekad dožive neuspjeh, ostavljajući agenta na istom mjestu. Na primjer, micanje desno u stanju 1 vodi do seta stanja (1, 2). Pogledajmo Sliku 5. koja prikazuje dio grafa pretraživanja. Vidimo da iz stanja 1 više nema necikličkih rješenja i AND-OR graf pretraživanja vraća grešku. Međutim, postoji cikličko rješenje koje ima svrhu pokušavati stalno desno dok god funkcionira. Ono glasi:

$$[\textit{Usiši, Ll: Desno,} \textbf{ if } \textit{Stanje} = 5 \textbf{ then } L_1 \textbf{ else } \textit{Usiši}]$$

Bolja sintaksa za "petljajući" dio ovog plana bila bi:

$$[\textbf{\textit{while}}\ \textit{Stanje=5}\ \textbf{\textit{do}}\ \textit{Desno}].$$

Općenito, ciklički plan mogao bi se smatrati rješenjem ako je svaki list ciljno stanje i ako je list dostupan iz svakog dijela plana.

Ključno shvaćanje je da se petlja iz stanja prostora u stanje L translatira u planiranu petlju sve do točke gdje se izvodi podplan za stanje L.

Agent koji izvodi cikličko rješenje s vremenom će doseći cilj ako se svaki ishod nedeterminističke akcije s vremenom pojavi.

Je li ovaj uvjet razuman?
Ovisi o razlogu za nedeterminizam. Ako se kocka baca onda je razumno pretpostaviti da će s vremenom biti dobivena i šestica. Ako je akcija provući hotelsku karticu kroz vrata sobe i otključati, a ne upali prvi put, ona će možda s vremenom proraditi ili možda osoba ima pogrešan ključ (ili je pogriješila sobu). Nakon 7 ili 8 pokušaja većina će ljudi pretpostaviti da je problem u ključu i vratit će se na recepciju kako bi dobili novi.
Jedan način za razumijevanje ove odluke je da je početna formulacija problema (vidljivo, nedeterminizirano) napuštena u korist drugačije formulacije (djelomično vidljivo, determinizirano) gdje je neuspjeh pripisan na nevidljivo (nepojmljivo) svojstvo ključa.

Zaključak

Postoji slučaj kad je okoliš potpuno vidljiv i determiniziran i agent zna koji su učinci svake akcije, a postoji i slučaj kada je okoliš djelomično vidljiv ili nedeterminiziran ili oboje, predmeti opažanja postaju korisni. U oba slučaja, budući predmeti opažanja ne mogu biti predodređeni unaprijed i budući postupci agenta ovisit će o tim budućim predmetima opažanja.

U nedeterminističkim okolinama agenti mogu primjeniti AND-OR stablo pretraživanja da bi stvorili potencijalne planove kojima dolaze do cilja neovisno o tome koji se ishodi pojavljuju tijekom provedbe. Takvo stablo sadrži dvije vrste čvorova: OR-čvorove, koji su naslijeđeni iz deterministčkog pretraživanja te AND-čvorove koji su karakteristični za nedeterminističko pretraživanje.

U nesigurnom svijetu vakuuma, gdje akcije pokreta ponekad dožive neuspjeh, idealno rješenje je ciklički plan.

Literatura

[1] S. Russell,P. Norvig *Artificial Intelligence A Modern Approach Third Edition*

[2] http://www.cs.nott.ac.uk/ nza/G52PAS/lecture6.pdf