SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1363

UČENJE NEURONSKIH MREŽA GENETSKIM ALGORITMOM

Krešimir Mišura

Zagreb, lipanj 2010.

SADRŽAJ

1 UVOD

Ovaj se rad bavi učenjem neuronskih mreža korištenjem genetskih algoritama. Svaka jedinka predstavlja jednu jednostavnu neuronsku mrežu, čiji je zadatak upravljati robotom kroz okolinu u kojoj su postavljeni keksi i prepreke. Cilj je naravno izbjegavati prepreke i skupljati kekse. Neuronske mreže su prilično jednostavne i samo prenose impulse od senzora do motora. Razlikuju se samo u težinama veza, odnosno u količini impulsa koja se prenosi od kojeg senzora do kojeg motora. Težine veza se mogu predstaviti običnom tablicom u kojoj redovi predstavljaju senzore, a stupci motore. Svaka vrijednost u tablici predstavlja težinu veze između tog senzora i motora. Te vrijednosti se evoluiraju genetskim algoritmom, a dobrota jedinke se procjenjuje na način da se nasumično generira ploča sa keksima i preprekama i pusti robot s danom mrežom da se snalazi po toj okolini. Na kraju se evaluiraju rezultati u odnosu na broj prepreka i keksa na ploči, te analizira snalaženje takvih robota po okolini.

U poglavlju „Genetski algoritmi“ dan je uvod u genetske algoritme i opisani su neki postupci koji se često primjenjuju u takvom pristupu problemima.

Poglavlje „Neuronske mreže“ daje uvod u neuronske mreže, kratak opis nekih značajnijih arhitektura takvih mreža, te se opisuje način učenja neuronskih mreža.

U poglavlju „Primjena neuronske mreže u upravljanju robotom“ opisan je način integriranja neuronske mreže u robota, način njezinog povezivanja sa senzorima i motorima, te kako ona zapravo može upravljati robotom.

Poglavlje „Rezultati“ bavi se analizom rezultata dobivenih nakon cijelog eksperimenta, opisuje način provođenja evolucije, te opisuje neke najznačajnije jedinke koje su se pojavile tokom evolucije.

U zaključku je dano mišljenje autora o primjenjivosti genetskih algoritama za učenje neuronskih mreža.

2 GENETSKI ALGORITMI

Evolucijsko računanje svoje temelje vuče iz teorije evolucije odnosno prirodne selekcije. U prirodi jedinke koje su bolje prilagođene svojoj okolini dulje preživljavaju od onih koje su lošije prilagođene i na taj način imaju veću vjerojatnost za reprodukciju, tj. prijenos svog genetskog materijala na sljedeće generacije. One jedinke koje su bile najuspješnije, uspjele su prenijeti svoj genetski materijal na sljedeću generaciju, dok loš genetski materijal u većini slučajeva nije prenesen. Samim time je sljedeća generacija već bolje prilagođena okruženju od prethodne. Isto razmišljanje se može primjeniti i za rješavanje nekih problema na računalima. Na početku se stvara populacija jedinki od kojih svaka predstavlja neko potencijalno rješenje problema. Nakon toga se na toj populaciji simulira evolucija. Za to je ključna mogućnost procjene koje rješenje je koliko dobro, odnosno koje rješenje je bolje od drugog. To se obavlja s funkcijom dobrote, koja obično vraća neku vrijednost po kojoj se mogu rangirati jedinke, i onda se po tome izabiru jedinke koje su bolje i čiji genetski materijal je potrebno prenijeti u sljedeću generaciju. Zatim izabiremo neke jedinke koje će se križati i na taj način se dobiva jedinka koja će biti u sljedećoj generaciji. Inovativna rješenja se osim križanjem dobivaju i mutacijom. Mutacija oponaša istoimenu pojavu u prirodi. Nasumično se promijeni neki dio genotipa jedinke. Nakon svega toga dobiva se nova generacija čije jedinke predstavljaju bolja rješenja od prethodne. Zatim se sve ponavlja ispočetka i tako dok se ne zadovolji neki uvjet zaustavljanja evolucije. To može biti kad je pronađeno dovoljno dobro rješenje, kad je isteklo neko unaprijed utvrđeno vrijeme, nakon određenog broja generacija...

Oponašanje evolucije na računalu naziva se evolucijsko računanje (engl. *evolutionary computation*). Evolucijsko računanje je podijeljeno na četiri glavne skupine: genetski algoritmi (engl. *genetic algorithms*, GA), genetsko programiranje (engl. *genetic programming*, GP), evolucijske strategije (engl. *evolution strategies*, ES) i evolucijsko programiranje (engl. *evolutionary programming*, EP). [2]

Koraci evolucijskog algoritma su [1]:

1. Inicijalizacija

   U početku se stvara početna populacija od nasumično generiranih jedinki. Veličina populacije ovisi o konkretnom problemu, ali obično se radi sa populacijama od nekoliko stotina do nekoliko tisuća jedinki. Ponekad se ne generira cijela populacija slučajno, nego se u nju ubace i neka dobra rješenja koja su nam već poznata.

2. Selekcija

   Tokom svake generacije dio jedinki se bira te im se omogućuje da svoj genetski materijal prenesu u sljedeću generaciju. Pojedinačna rješenja se biraju na temelju funkcije dobrote, na način da jedinke s najvećom dobrotom imaju najveću vjerojatnost da budu izabrane. Dvije najpoznatije vrste selekcije su selekcija po uzoru na kotač ruleta i turnirska selekcija.

3. Reprodukcija

   U koraku reprodukcije stvaramo sljedeću generaciju jedinki, koristeći genetske operatore poput križanja i mutacije.

   Za svaku novu jedinku biramo dvije „roditeljske“ jedinke za razmnožavanje. Dijete se proizvodi od tih jedinki koristeći križanje ili mutaciju. Za svaku novu jedniku selekcijom se biraju novi roditelji. Nakon reprodukcije stvorena je nova generacija jedinki i ona tipično sadrži jedinke veće dobrote od prethodne generacije. No, bitno je naglasiti da nisu izabrane isključivo najbolje jedinke, već mogu biti prenesene i neke jedinke slabije dobrote. Ta činjenica je korisna jer pomaže da algoritam ne zapinje u lokalnim optimumima. Neko rješenje koje se u ovom trenutku čini lošije možda ima potencijal da na kraju iznjedri bolje rješenje od onoga koje u ovom trenutku ima najveću dobrotu. Reklo bi se da „ostavljamo sve opcije otvorene“.

4. Završetak

   Prethodno navedeni postupci selekcije i reprodukcije se ponavljaju sve dok se ne zadovolji neki uvjet prekida evolucije. Najčešći uvjeti su:

   - Nađeno je optimalno rješenje
   - Prošao je neki unaprijed određeni broj generacija
   - Prošlo je neko unaprijed određeno vrijeme

- Nakon određenog broja uzastopnih generacija nije napravljen nikakav napredak u dobroti
- Programer može ručno prekinuti evoluciju u nekom trenutku

Pseudokod evolucijskog računanja [4]:

1. Generiraj početnu populaciju
2. Procijeni dobrotu svake jedinke u populaciji
3. Ponavljaj za svaku generaciju dok se ne ispuni neki uvjet zaustavljanja
    a. Izaberi najbolje jedinke za reprodukciju
    b. Stvori nove jedinke koristeći genetske operatore krosover i mutaciju
    c. Procijeni dobrotu svake jedinke u populaciji
    d. Zamijeni jedinke lošije dobrote sa novo stvorenim jedinkama

3 NEURONSKE MREŽE

Ideja za neuronske mreže dolazi od ispitivanja centralnog živčanog sustava i neurona, koji čine njegove najznačajnije elemente za procesiranje informacija.

3.1 ŠTO SU NEURONSKE MREŽE

Umjetne neuronske mreže (dalje samo „neuronske mreže“) su matematički ili računalni model kojim se pokušava simulirati struktura i funkcionalni aspekti bioloških neuronskih mreža. Sastoje se od povezane grupe umjetnih neurona i procesiraju informacije koristeći konektivistički pristup računanju. [5]

Neuronske mreže obično dolaze s algoritmima koji su dizajnirani tako da mijenjaju mrežu, odnosno težine veza između neurona da bi postigle željeni tok signala. Na taj način neuronske mreže uče. Iako sposobnost učenja nije nužna, ona je vrlo korisna.

3.2 USPOREDBA NEURONSKIH MREŽA I KONVENCIONALNIH RAČUNALA

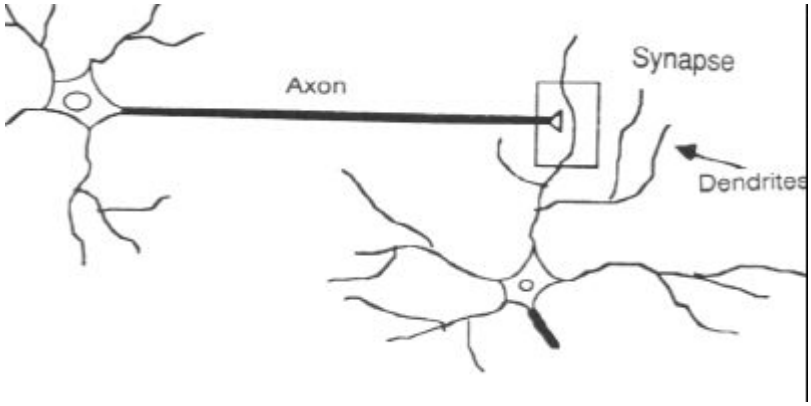
Konvencionalna računala slijede niz instrukcija da bi riješila neki problem. Ukoliko nisu poznati točni koraci koje računalo mora napraviti, ono neće uspjeti riješiti problem.

Neuronske mreže funkcioniraju na način mnogo sličniji ljudskom mozgu. Mreža se sastoji od velikog broja međusobno povezanih neurona koji rade paralelno da bi riješili neki problem. One uče na primjeru, tj. ne programiraju se unaprijed. Nakon nekog broja primjera na kojima se neuronske mreže uče, one mogu riješiti taj problem.

Dakle, kod neuronskih mreža mi sami ne moramo nužno znati riješiti problem, dok kod konvencionalnih računala moramo znati kako riješiti taj problem i također moramo podijeliti rješenje u niz malih jednoznačnih instrukcija. Ponašanje konvencionalnog računala je dakle potpuno predvidivo (ukoliko nismo napravili greške prilikom programiranja), dok se ponašanje neuronskih mreža ne mora nužno moći predvidjeti. [3]

3.3 USPOREDBA PRIRODNOG I UMJETNOG NEURONA

Neuron impulse šalje preko dugačke tanke niti koja se zove akson i koji se grana na tisuće grana. Na kraju svake grane postoji sinapsa koja spaja akson sa dendritom od drugog neurona. Taj drugi neuron skuplja sve impulse iz svojih dendrita. Ukoliko je zbroj tih impulsa dovoljno velik tada neuron prosljeđuje impuls na svoj akson. Učenje se događa tako da se mijenja učinkovitost synapse, tj. učinkovitost prijenosa impulsa sa jednog neurona na drugi. Slika 3-1 prikazuje prirodni neuron.

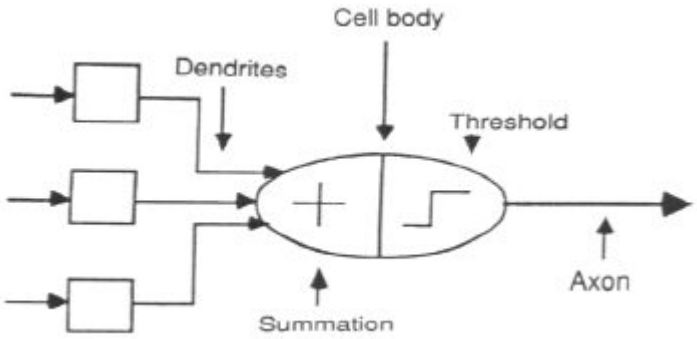


Slika 3-1 Prirodni neuron

Umjetni neuron je pojednostavljena simulacija prirodnog. Sastoji se opet od dendrite, aksona i tijela stanice. Svaka veza između dva neurona je određena težinom te veze. Izlaz iz jednog neurona množi se sa težinom veze i kao takav ulazi u sljedeći neuron. Suma svih tih vrijednosti je ukupni ulaz u neuron. U neuronu imamo prijenosnu funkciju prema kojoj se određuje izlaz neurona u odnosu na njegov ulaz. Najčešće funkcije su linearne ili u obliku praga. Kod funkcije praga izlaz je nula sve dok je ulaz manji od neke vrijednosti. Kada ukupni ulaz prijeđe tu vrijednost tada izlaz postaje 1, tj. tada se neuron aktivira. Kod linearne funkcije obično se ulaz preslikava na izlaz.

Problem kod umjetnih neurona je što ne mogu raditi zaista paralelno, tj. takvo ponašanje je praktično nemoguće postići na današnjim računalima. Taj problem rješavamo na način da se simulacija ponašanja neurona izvodi u diskretnim koracima. Prvo imamo ulaze u neke neurone, i to je sve u trenutku t_0. Tada u odnosu na ulaze svih neurona računamo njihove izlaze, te na taj način dobivamo stanje u trenutku t_1. Sad opet pogledamo kakav je ulaz u koji neuron i izračunamo stanje u t_2 i

taj postupak ponavljamo te tako simuliramo rad mreže. Slika 3-2 prikazuje jednan primjer umjetnog neurona.



Slika 3-2 Umjetni neuron

3.4 NEKE VAŽNE NEURONSKE MREŽE

3.4.1 FEED-FORWARD MREŽE

Feed-forward mreže omogućuju signalima da putuju samo u jednom smjeru, od ulaza prema izlazu. Nema petlji u mreži, tj. izlaz iz jednog sloja ne može utjecati na taj isti sloj. Feed forward mreže su najčešće jednostavne mreže koje povezuju ulaz sa izlazom. Vrlo često se koriste u prepoznavanju uzoraka.

3.4.2 FEEDBACK MREŽE

U feedback mrežama signali mogu putovati u oba smjera jer postoje petlje u mreži. Ovakve mreže su dosta moćne i često dosta složene. Feedback mreže su dinamičke, stanje im se kontinuirano mijenja dok ne dosegnu točku ekvilibrijuma. U toj točki ostaju tako dugo dok se ne promijeni ulaz i tada se traži nova točka ekvilibrijuma.

3.4.3 MREŽNI SLOJEVI

Najčešće postoje tri sloja. Ulazni sloj, skriveni sloj, izlazni sloj.

- Aktivacija ulaznog sloja ovisi o informacijama koje se unose u mrežu.

- Aktivacija skrivenog sloja ovisi o aktivaciji ulaznog sloja i težinama veza između ulaznih i skrivenih neurona.

- Aktivacija izlaznog sloja ovisi o aktivaciji skrivenog sloja i težinama veza između skrivenog i izlaznog sloja.

3.5 UČENJE U NEURONSKIM MREŽAMA

Rad neuronskih mreža je obično podijeljen u dvije faze: fazu učenja i fazu obrade podataka.

Učenje je iterativan postupak predočavanja ulaznih primjera i eventualno očekivana izlaza pri čemu dolazi do postupnog prilagođavanja težina veza neurona. Jedno predočavanje svih uzoraka naziva se epohom.

Učenje se prema načinu predočavanja ulaza dijeli na:

- Pojedinačno učenje (*on-line*)
    - Kod ovakvog učenja nakon svakog primjera koji smo dali mreži podešavamo težine veza među neuronima
- Grupno učenje (*batch*)
    - Cijela epoha, odnosno svi uzorci se prikažu mreži te tek onda podešavamo težine veza između neurona.

Prema tome postoji li učitelj koji uči neuronsku mrežu, učenje dijelimo na

- Učenje s učiteljem (*supervised learning*)
    - Za svaki uzorak se neuronskoj mreže osim ulaza daje i izlaz. Neuronska mreža tada podešava svoje težine da bi prilagodila svoj izlaz traženom izlazu.
- Učenje bez učitelja (*unsupervised learning*)
    - Mreži se daje ulaz, ali je izlaz nepoznat.

Kod učenja neuronskih mreža na nekim uzorcima često zna doći do neželjene pojave koja se naziva pretreniranost. To se događa ukoliko je neuronska mreža previše puta izložena istim uzorcima te ona gubi svojstvo generalizacije i točno je naučila samo dane ulaze. Kažemo da je postala „štreber“.

Da bi izbjegli taj problem, skup primjera za učenje dijelimo na:

- Skup za učenje – prema njemu se podešavaju težine
- Skup za testiranje – provjerava se rad mreže
- Skup za provjeru – konačna provjera

Učenje neuronskih mreža uporabom genetskih algoritama spada u nadgledano učenje. Jedinkama se nakon svake generacije određuje dobrota te se prema tome određuje koliko je koji izlaz neuronske mreže ispravan. Također u našem primjeru se

okolina, tj. ulazni podaci svaki put nasumično generiraju pa se time izbjegava problem pretreniranosti.

Naravno, osim učenja uporabom genetskog algoritma, postoje i standardni algoritmi prema kojima neuronske mreže mogu učiti. Jedan od najpoznatijih algoritama za višeslojne mreže je *backpropagation* algoritam. On se temelji na određivanju pogreške u izlaznom sloju, te prosljeđivanju te pogreške u nazad prema prethodnim slojevima i na temelju toga podešavanju težina veza među neuronima.

3.6 PRIJENOSNA FUNKCIJA

Ponašanje neuronske mreže osim o težinama veza ovisi i o prijenosnoj funkciji između neurona. Ova funkcija najčešće spada u jednu od 3 kategorije:

- Linearna – kod ovakvih funkcija izlaz je direktno proporcionalan ukupnom ulazu.
- „*treshold*“ – Kod ovakve funkcije izlaz može biti jedna od dvije vrijednosti u ovisnosti je li ukupan ulaz veći ili manji od nekog praga.
- „*sigmoid*“ – U ovakvim funkcijama izlaz se kontinuirano mijenja kada se mijenja ulaz, ali ne nužno linearno. Ovakve funkcije mogu postići veću sličnost sa funkcioniranjem pravih neurona od dviju prethodno nabrojanih.

4 PRIMJENA NEURONSKE MREŽE U UPRAVLJANJU ROBOTOM

Neuronske mreže se mogu ugraditi u fizičke robote na prirodan način. Korištenje neuronskih mreža za kontroliranje robota ima veliku perspektivu i pomaže pri objašnjavanju inteligencije živih bića. Neuronske mreže imaju sposobnost učenja iz iskustva „Kada je akson stanice A dovoljno blizu da pobudi stanicu B i učestalo sudjeluje u pobuđivanju te stanice, događa se neki proces rasta ili metaboličke promjene u jednoj ili obje stanice tako da učinkovitost A kao jedne od stanica koja pobuđuje B, je povećana.“ [6]

Neuronske mreže će se učiti pomoću genetskog algoritma. Svaka veza između dva neurona je predstavljena nekim realnim brojem. Skup tih brojeva jest genotip jedinke. Cilj je proizvesti onaj genotip koji ima takve težine da se robot koji je upravljan njime najbolje snalazi u svojoj okolini.

Okolina je predstavljena određenim brojem keksa i prepreka. Cilj robota je zaobilaziti prepreke i skupljati kekse. Dobrota jedinke se određuje prema tome koliko je robot pojeo keksa i koliko puta se zaletio u prepreku. Naime, svaki put kada robot sakupi keks dobit će određeni broj bodova, a kad se zaleti u prepreku izgubit će neki broj bodova. Taj broj bodova će predstavljati dobrotu jedinke.

Naravno, ne postoji genotip koji je najbolji za svaki tip okoline. U okolinama gdje su prepreke jako guste, senzori će stalno biti podraženi te je potrebno naći pravilnu mjeru aktivacije motora koja neće uzalud izbjegavati mnogo udaljenih prepreka i na taj način propustiti pojesti hranu koja je nadohvat ruke.

U ovom radu evoluiraju se neuronske mreže za kontrolu robota u različitim uvjetima. Mijenja se broj prepreka i broj keksa na ploči i promatra se kako se pritom mijenjaju karakteristike neuronskih mreža. Na kraju se dobivena rješenja uspoređuju s nekim standardnim rješenjima za taj tip problema.

4.1 SENZORI

Svaki robot ima 4 senzora, lijevi i desni senzor hrane te lijevi i desni senzor prepreka. Svaki senzor ima određeno vidno polje koje se proteže od smjera kretanja pa za određeni kut u lijevo ili desno. Aktivacija senzora ovisi o udaljenosti od prepreke, i obrnuto je proporcionalna sa tom udaljenosti.

Funkcija na slici 4-1 radi sljedeće:

1. Iterira po svim keksima i za svaki provjerava da li se nalazi u vidnom polju nekog senzora. Ukoliko se nalazi u vidnom polju, tada se taj senzor aktivira sa vrijednošću: 1/udaljenost keksa

2. Iterira po svim preprekama i za svaku provjerava da li se nalazi u vidnom polju nekog senzora. Ukoliko se nalazi u vidnom polju, tada se taj senzor aktivira sa vrijednošću: 1/udaljenost prepreke

Ukoliko je neki senzor aktiviran za više prepreka ili keksa tada suma aktivacija predstavlja ukupnu aktivaciju tog senzora.

```
funkcija postaviSenzore
 resetirajSenzore()
 za svaki keks čini
  ako je keks u vidnom polju lijevog senzora hrane tada
   aktivacija lijevog senzora hrane += 1/udaljenost keksa
  ako je keks u vidnom polju desnog senzora hrane tada
   aktivacija desnog senzora hrane += 1/udaljenost keksa
 kraj
 za svaku prepreku čini
  ako je prepreka u vidnom polju lijevog senzora prepreka tada
   aktivacija lijevog senzora prepreka += 1/udaljenost prepreke
  ako je prepreka u vidnom polju desnog senzora prepreka tada
   aktivacija desnog senzora prepreka += 1/udaljenost prepreke
 kraj
kraj
```

Slika 4-1 Pseudokod funkcije za postavljanje senzora

4.2 MOTORI

Robot sadrži dva motora za skretanje. Motor za kretanje prema naprijed je uvijek aktivan i robot se kreće konstantnom brzinom. Ukoliko je aktivan lijevi motor za skretanje a desni nije tada robot skreće u desno, a ukoliko je aktivan desni motor tada robot skreće u lijevo.

Funkcija na slici 4-2 u ovisnosti o genotipu, odnosno prijenosnoj funkciji neurona između senzora i motora, aktivira motore. Aktivacija motora ne može biti veća od 1, tako da ukoliko je umnožak prijenosne funkcije i aktivacije senzora veći od 1, tada se aktivacija motora postavlja na 1.

```
funkcija postaviMotore
 za svaki motor čini
  za svaki senzor čini
   aktivacija Motora = aktivacija Motora + aktivacija senzora *
težina veze
   ako je aktivacija motora > 1 tada aktivacija motora = 1
  kraj
 kraj
kraj
```

Slika 4-2 pseudokod funkcije za postavljanje motora

4.3 POMICANJE ROBOTA

Kretanje robota po okolini je ostvareno kao veliki broj diskretnih koraka. Svaki taj korak je implementiran funkcijom na slici 4-3.

Ta funkcija služi za izvođenje jednog koraka robota u njegovom okolišu. Prvo u odnosu o aktivaciji lijevog i desnog motora rotira robota za odgovarajući kut. Zatim pomiče robota za neku unaprijed određenu udaljenost u tom smjeru. Na kraju se provjerava da robot nije slučajno izašao izvan polja, jer u tom slučaju se robot "odbija" od ruba, tj. smjer u kojem je okrenut se mijenja za 180 stupnjeva.

```
void pomakniVozilo(){
  smjer=smjer+(aktivacijaMotora[1]-aktivacijaMotora[0])*brzinaSkretanja;
  if(smjer<0) smjer+=2*PI;
  if(smjer>2*PI) smjer-=2*PI;
  korx+=brzinaVoznje*cos(smjer);
  kory+=brzinaVoznje*sin(smjer);
  if(korx<0){
    korx=0;
    smjer=zbrojKuteva(smjer, PI);
  }
  if(korx>SIRINAPOLJA){
    korx=SIRINAPOLJA;
    smjer=zbrojKuteva(smjer, PI);
  }
  if(kory<0){
    kory=0;
    smjer=zbrojKuteva(smjer, PI);
  }
  if(kory>VISINAPOLJA){
    kory=VISINAPOLJA;
    smjer=zbrojKuteva(smjer, PI);
  }
}
```

Slika 4-3 Pseudokod funkcije za pomicanje robota

4.4 RAČUNANJE DOBROTE

Dobrota jedinke se računa prema tome koliko keksa robot skupi, koliko uspješno izbjegava prepreke i koliko brzo skuplja kekse.

Funkcija *provozaj* simulira kretanje robota kroz njegovu okolinu. Okolina se prilikom svake vožnje nanovo nasumično generira, tako izbjegavamo da se robot naučio kretati samo pri jednoj određenoj okolini. Ova funkcija vraća double vrijednost, i ta vrijednost predstavlja dobrotu jedinke.

```
double provozaj(){
  double value=0;
  postaviPreprekeIKekse();
  for(int t = 0; t<BROJKORAKA; t++){
    if(brojPojedenihKeksa<BROJKEKSA){
      value+=napraviKorak();
    }
                                        }
  return value;
}
```

Slika 4-4 Pseudokod funkcije za simulaciju kretanja robota kroz okolinu

Kao što je vidljivo u prethodnoj funkciji, simulacija kretanja po okolini se provodi kao neki broj diskretnih koraka. Prilikom svakog novog koraka mijenja se procjena dobrote, a to se postiže funkcijom na slici 4-5.

```
double promjenaVrijednosti(){
  double vrijednost=-KAZNAPOKORAKU;
  for(int i=0;i<BROJPREPREKA;i++){
    if(udaljenost(korx,kory,prepreke[i][0],prepreke[i][1])<VELICINAPREPREKE){
      vrijednost-=UKUPNAKAZNA/BROJPREPREKA;
    }
  }
  for(int i=0;i<BROJKEKSA;i++){
    if(udaljenost(korx,kory,hrana[i][0],hrana[i][1])<VELICINAHRANE){
      vrijednost+=UKUPNANAGRADA/BROJKEKSA;
      pojediHranu(i);
    }
  }
  return vrijednost;
}
```

Slika 4-5 Pseudokod funkcije za promjenu vrijednosti dobrote

Funkcija na slici 4-5 se poziva nakon svakog koraka. Vraća se double vrijednost koja predstavlja promjenu procjene dobrote jedinke u tom koraku. Svakim novim korakom se dobrota jedinke smanjuje za neku unaprijed utvrđenu vrijednost. Time se favoriziraju roboti koji sakupe svu hranu u što manjem broju koraka. Ukoliko je u trenutačnom koraku robot naletio na keks, tada on pojede taj keks i poveća mu se procjena dobrote. Ukoliko je naletio na prepreku tada se smanjuje procjena dobrote. Dobrota jedinke se računa prema sljedećem izrazu:

$$dobrota = 100 * \frac{broj\ skupljenih\ keksa}{ukupni\ broj\ keksa} - 100 * \frac{broj\ dodira\ prepreke}{ukupni\ broj\ prepreka} - \frac{broj\ koraka}{100}$$

U ovoj konkretnoj implementaciji neuronske mreže ne sadrže skriveni sloj i evoluiramo samo težine veza između senzora i motora. Za zadani problem takva implementacija je bila dovoljna, a pisana je dovoljno općenito da ne bi trebalo predstavljati veći problem dodati još i skriveni sloj u neuronsku mrežu.

4.5 Opis simulacije

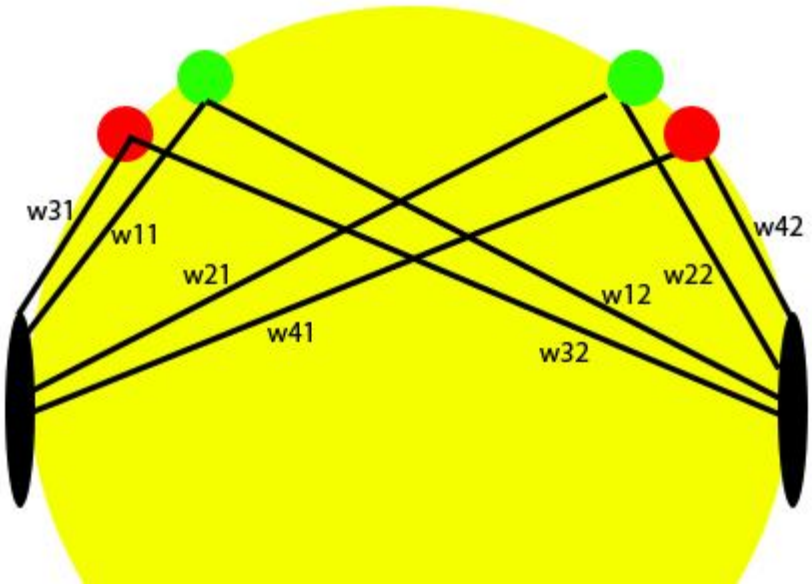
Simulacija se izvršava u obliku velikog broja diskretnih koraka. Robot prvo postavlja senzore na temelju okoline u kojoj se nalazi. Nakon toga se na temelju veza između senzora i motora aktiviraju motori. Zatim se izračunavaju sljedeće koordinate i smjer robota te se robot pomiče na novu poziciju. Kad se nađe u novoj poziciji provjerava se je li se zaletio u prepreku ili hranu. Ukoliko se nalazi na nekoj hrani tada ju jede i ona se miče s ploče. Zatim se dodaju ili oduzimaju određeni bodovi u njegovoj funkciji dobrote. Opisani postupak predstavlja jedan korak. Zatim se cijeli proces ponavlja i tako neki unaprijed određeni broj koraka.

5 REZULTATI

Eksperiment se izvodi za različite vrijednosti broja prepreka i keksa. Za svaku kombinaciju količine keksa i prepreka parametri evolucije su isti. Za svaki keks koji robot sakupi on dobiva nagradu, a za svaku prepreku u koju se zaleti dobiva kaznu. Naravno, keks je moguće pojesti samo jednom dok se u istu prepreku može zaletiti mnogo puta. Ukupna nagrada svaki put iznosi 100 bodova, tj. nagrada za svaki keks je 100/(ukupni broj keksa) na ploči. Isto vrijedi i za kaznu, dakle ukupna kazna svih prepreka je 100, dakle kazna prilikom jednog zalijetanja u prepreku je 100/(ukupni broj prepreka). Roboti prilikom obilaženja ploče mogu napraviti maksimalno 10000 koraka. Prilikom svakog koraka dobivaju kaznu od 0.01 bod. Ukoliko robot skupi sve kekse na ploči, tada više ne dobiva kaznu za svaki korak. Time se postiže da roboti koji brže skupe hranu imaju veću dobrotu. Vidno polje robota je od smjera kretanja pa u lijevo za 90° za lijevi senzor, te od smjera kretanja pa u desno za 90° za desni senzor. Evolucija se provodi kroz 30 generacija. Veličina populacije je 100 jedinki. Koristi se turnirska selekcija i veličina turnira je 3.

Koristit ćemo linearnu prijenosnu funkciju za neurone, odnosno neuroni će samo preslikavati ulaz na izlaz.

Tablica 5-1 prikazuje rezultate pokretanja evolucije. U svakoj ćeliji su dani rezultati za određeni broj keksa i prepreka u okolini. Za svaku konfiguraciju okoline evolucija je pokrenuta tri puta. Navedena je prosječna dobrota najbolje jedinke kroz ta tri pokretanja evolucije. Također je navedena i konfiguracija mreže apsolutno najbolje jedinke za danu konfiguraciju okoline. Na slici 5-1 je vidljivo koja težina se odnosi na koju poveznicu.



Slika 5-1Težine među neuronima

	5 prepreka	10 prepreka	20 prepreka	30 prepreka	40 prepreka
5 keksa	**91,69**	**79,97**	**65,25**	**58,94**	**36,95**
	W11:135,34	W11:-140,531	W11:52.4834	W11:-49,7826	W11:173,848
	W12:105,951	W12:30,1991	W12:276.846	W12:232,021	W12:134,424
	W21:58,3978	W21:-99,6429	W21:173.042	W21:125,672	W21:258,499
	W22:-116,965	W22:-95,9258	W22:-93.0327	W22:-2,81529	W22:-48,8487
	W31:-275,024	W31:35,9304	W31:-285.241	W31:-88,1987	W31:-174,278
	W32:-199,364	W32:-49,929	W32:-263.47	W32:-117,588	W32:-177,336
	W41:-172,264	W41:78,7594	W41:-239.3	W41:-150,309	W41:-253,6
	W42:-42,9892	W42:3,777	W42:-204.949	W42:-174,26	W42:-239,904
10 keksa	**88,07**	**89,09**	**45,94**	**-9,17**	**1,06**
	W11:-37,6242	W11:-125,937	W11:-15.3216	W11:109.631	W11:-262,28
	W12:64,06	W12:-91,8425	W12:25.182	W12:137.079	W12:-66,98
	W21:136,072	W21:293,893	W21:84.033	W21:252.481	W21:148,67
	W22:-14,24	W22:-79,4278	W22:-112.662	W22:-124.344	W22:231,55
	W31:-221,9	W31:-181,163	W31:-20.6317	W31:-286.468	W31:9,21
	W32:-191,728	W32:-116,452	W32:0.6	W32:-134.873	W32:203,328
	W41:-170,652	W41:-187,535	W41:-65.62	W41:-134.873	W41:-133,756
	W42:-96,4385	W42:-152,836	W42:-50.588	W42:-129.599	W42:-238,219
20 keksa	**77,83**	**63,96**	**77,17**	**41,53**	**-18,88**
	W11:-119,565	W11:-68.368	W11:-82.0	W11:-107.7	W11:-171,33
	W12:231,179	W12:134.534	W12:153.83	W12:283.932	W12:204,866
	W21:175,972	W21:287.192	W21:298.746	W21:-44.01	W21:129,004

	W22:32,818 W31:-150,47 W32:-233,7 W41:-53,55 W42:-113,0	W22:83.3 W31:119.26 W32:-51.74 W41:-199.97 W42:-231.023	W22:15.477 W31:-180.89 W32:-238.05 W41:-159.117 W42:-168.44	W22:-112.7 W31:-136.905 W32:-176.704 W41:87.4022 W42:71.3619	W22:98,0224 W31:41,8631 W32:-8,60151 W41:-84,2252 W42:-110,245
30 keksa	**70,60** W11:-23.433 W12:265.786 W21:297.079 W22:-12.08 W31:-89.94 W32:-210.094 W41:-36.8 W42:9.3	**69,15** W11:-67.85 W12:7.11 W21:120.417 W22:-26.7109 W31:-263.049 W32:-292.53 W41:-157.395 W42:-201.653	**37,24** W11:3.0 W12:113.77 W21:220.467 W22:-113.7 W31:-16.02 W32:-31.45 W41:-207.75 W42:-133.32	**-34,92** W11:139,24 W12:56,9512 W21:297,556 W22:-120,334 W31:-120,426 W32:-113,455 W41:-285,095 W42:-239,629	**-56,00** W11:-137.839 W12:0.645 W21:117.597 W22:230.703 W31:36.3882 W32:-1.75326 W41:-141.209 W42:-155.528
40 keksa	**53,38** W11:-84.66 W12:65.14 W21:183.42 W22:-27.35 W31:-75.27 W32:-115.19 W41:-114.84 W42:-165.782	**57,27** W11:-49.78 W12:261.007 W21:292.136 W22:-9.517 W31:-198.997 W32:-237.725 W41:-165.122 W42:-127.256	**20,68** W11:-61.7029 W12:131.93 W21:58.38 W22:-9.02 W31:-82.30 W32:-115.116 W41:-254.717 W42:-227.013	**-41,33** W11:-116,452 W12:36,17 W21:-67,03 W22:-143,68 W31:-44,67 W32:-123,63 W41:-121,506 W42:-101,95	**-44,67** W11:199,355 W12:100,385 W21:-37,5875 W22:-102,426 W31:223,415 W32:3,03 W41:-228,588 W42:-189,329

Table 5-1Prikaz rezultata evolucije

Kako je i očekivano, s porastom broja keksa ili prepreka na ploči ukupna dobrota pada. Kao što je već rečeno, svi keksi zajedno nose nagradu od 100 bodova. Robotu je dakle jednostavnije skupiti sve kekse ukoliko njih ima 5, nego ukoliko ih ima 40. Također, robot dobiva neku malu kaznu za svaki korak koji napravi. Time se postiže da evolucija preferira jedinke koje brže skupljaju kekse. No, to također znači da ukoliko ima više keksa na ploči, i robotu onda naravno treba više vremena da ih sakupi sve, njegova dobrota će biti manja nego što bi bila da ima manje keksa na ploči.

Očito je naravno da i povećanje broja prepreka smanjuje dobrotu. Naime što više prepreka ima to je veća vjerojatnost da će robot naletiti na neku od njih.

Najveća teoretski moguća dobrota bi bila 100, ali to je moguće postići samo ako u jednom koraku robot odmah sakupi sve kekse, i naravno ne naleti na niti jednu prepreku. To praktično naravno nije moguće. No pri manjem broju keksa na ploči dobrota najbolje jedinke se dosta približava toj vrijednosti pa tako dostiže čak 96.74 boda.

Minimalna vrijednost dobrote je jako niska. Robot se može naime u svakom koraku simulacije zaletiti u prepreku. Dobrota bi mu u tom slučaju iznosila –(0.5+0.01)*10000 = -5100.

Čak i pri najtežim uvjetima s velikim brojem prepreka i keksa, najbolje jedinke su imale dobrotu od -52, tj. uspjele su sakupiti samo mali dio keksa, ali su zato dobro izbjegavali prepreke.

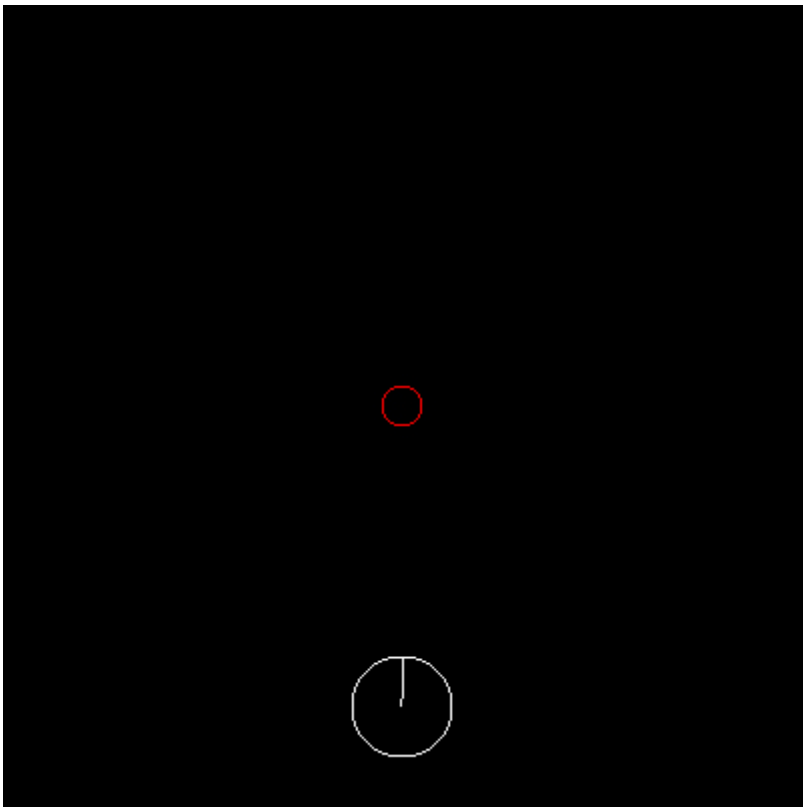
Iz rezultata je vidljivo da su roboti za manji broj keksa uspjeli skupiti sve kekse, nisu se zalijetali u prepreke i nije im trebalo puno vremena da obave sav posao.

U slučajevima kada postoji veliki broj keksa i mali broj prepreka, roboti su se također dobro ponašali i najbolji među njima su uspjevali sakupiti sve kekse.

Najlošijim su se pokazale situacije s puno keksa i puno prepreka. U tim slučajevima čak ni najbolji roboti nisu uspjeli sakupiti sve kekse.

Tokom simulacije kretanja robota kroz okolinu uočeni su neki problemi. Najvažniji su izneseni u nastavku:

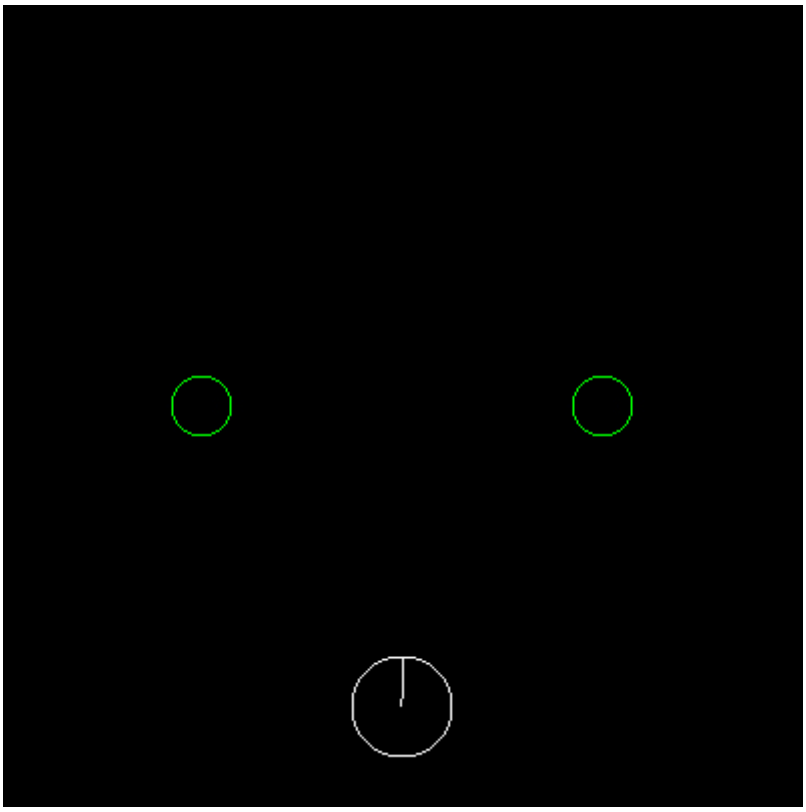
5.1 ROBOT IDE RAVNO PREMA PREPRECI



Slika 5-2 Robot ide ravno prema prepreci

Ukoliko bi neuronska mreža bila simetrična, tj. jednako reagirala na podražaje sa lijeve i desne strane, tada može doći do neželjenih ponašanja kao npr. u ovoj situaciji. Ovdje je robot usmjeren ravno prema prepreci i aktivacija lijevog i desnog senzora je jednaka. Ukoliko je neuronska mreža simetrična tada će aktivacija lijevog i desnog motora biti također jednaka. Dogodit će se to da se robot neće rotirati niti u jednu stranu već će se kretati ravno prema naprijed i tako naletiti na prepreku.

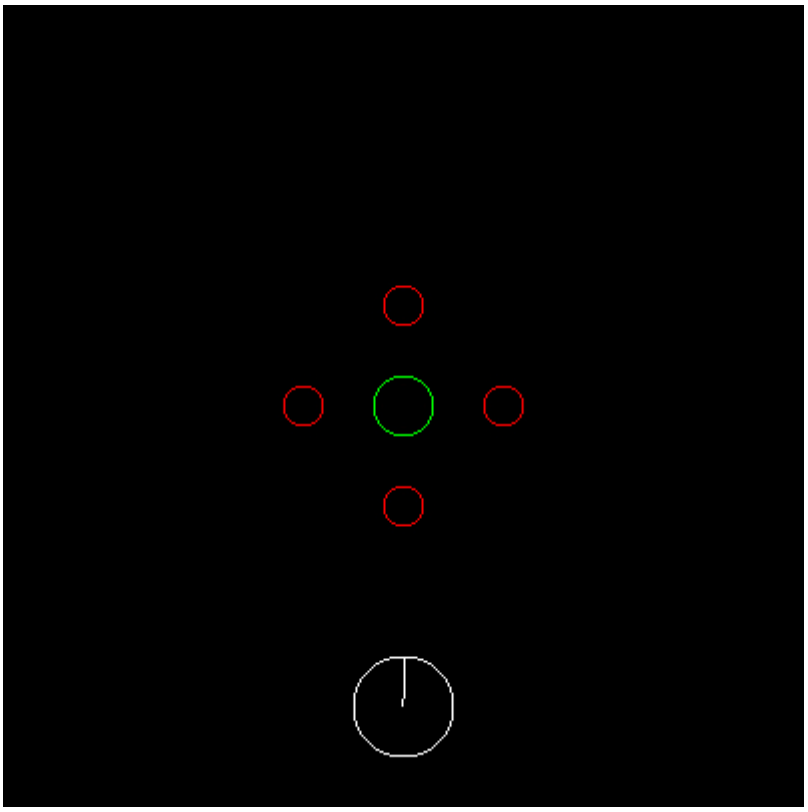
5.2 ROBOT IDE RAVNO IZMEĐU DVA KEKSA



Slika 5-3 Robot ide ravno između dva keksa

Problem kod ovog primjera je vrlo sličan kao i kod prethodnog. Oba senzora su jednako podražena, te ukoliko je mreža simetrična robot će jednako aktivirati oba motora i kretat će se ravno naprijed jer neće moći odlučiti koji keks da skupi. Na kraju će proći točno po sredini između dva keksa, ne skupivši niti jedan. Ovaj i prethodni primjer pokazuju da iako se simetrične mreže čine logičnim izborom, možda i nisu najbolje.

5.3 KEKS JE OKRUŽEN PREPREKAMA



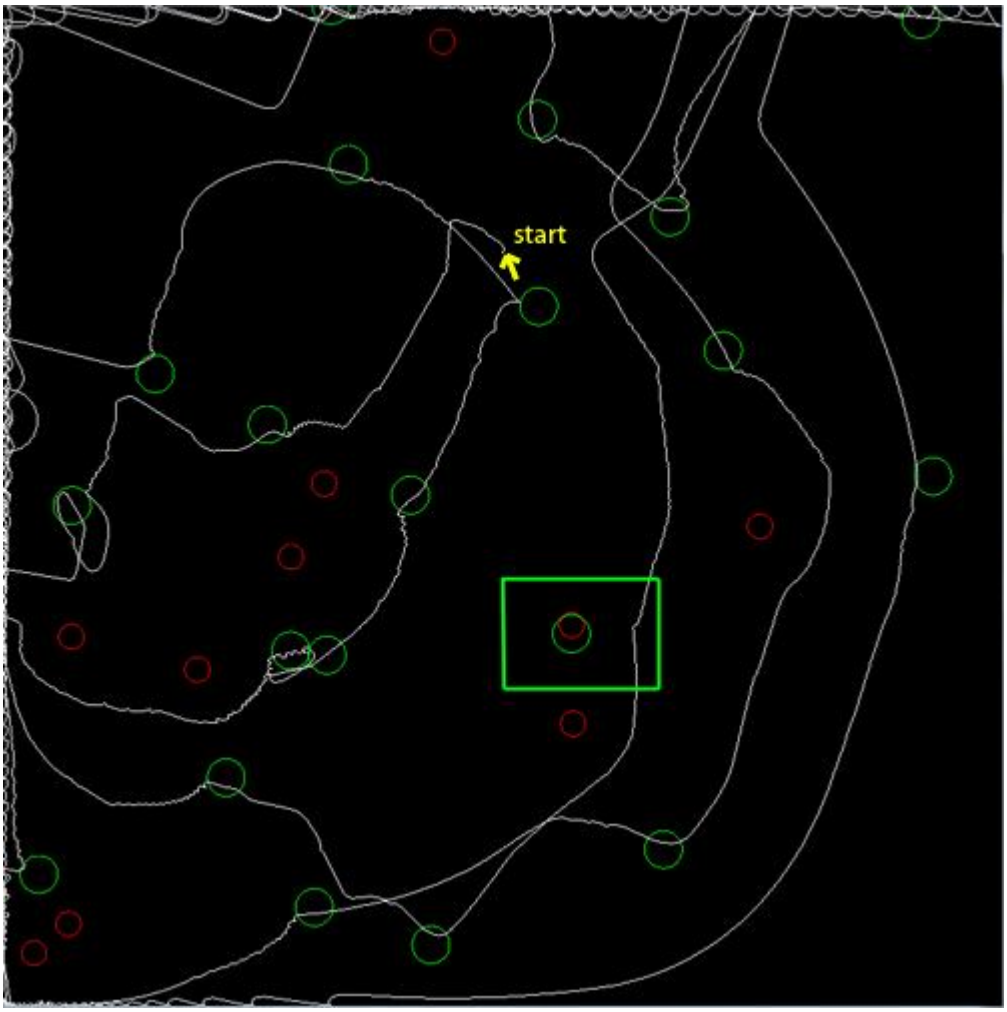
Slika 5-4 Keks je okružen preprekama

Ovdje je jedan keks okružen sa više prepreka. Bilo koji robot u blizini će imati puno veću aktivaciju senzora za prepreke od onog za hranu. Samim time će izbjegavati cijelo to područje i nikad neće skupiti taj keks.

Taj keks se inače može skupiti, jer bi robot mogao proći između dvije prepreke, skupiti keks, i izaći između dvije prepreke.

Iako se prethodna dva primjera mogu riješiti jednostavnim uvođenjem asimetrije u mrežu, ovaj problem je nešto složeniji. S ovakvom arhitekturom neuronske mreže ga nije moguće riješiti na zadovoljavajuć način. Mogli bi ga riješiti tek uvođenjem dodatnih senzora i skrivenog sloja u neuronsku mrežu.

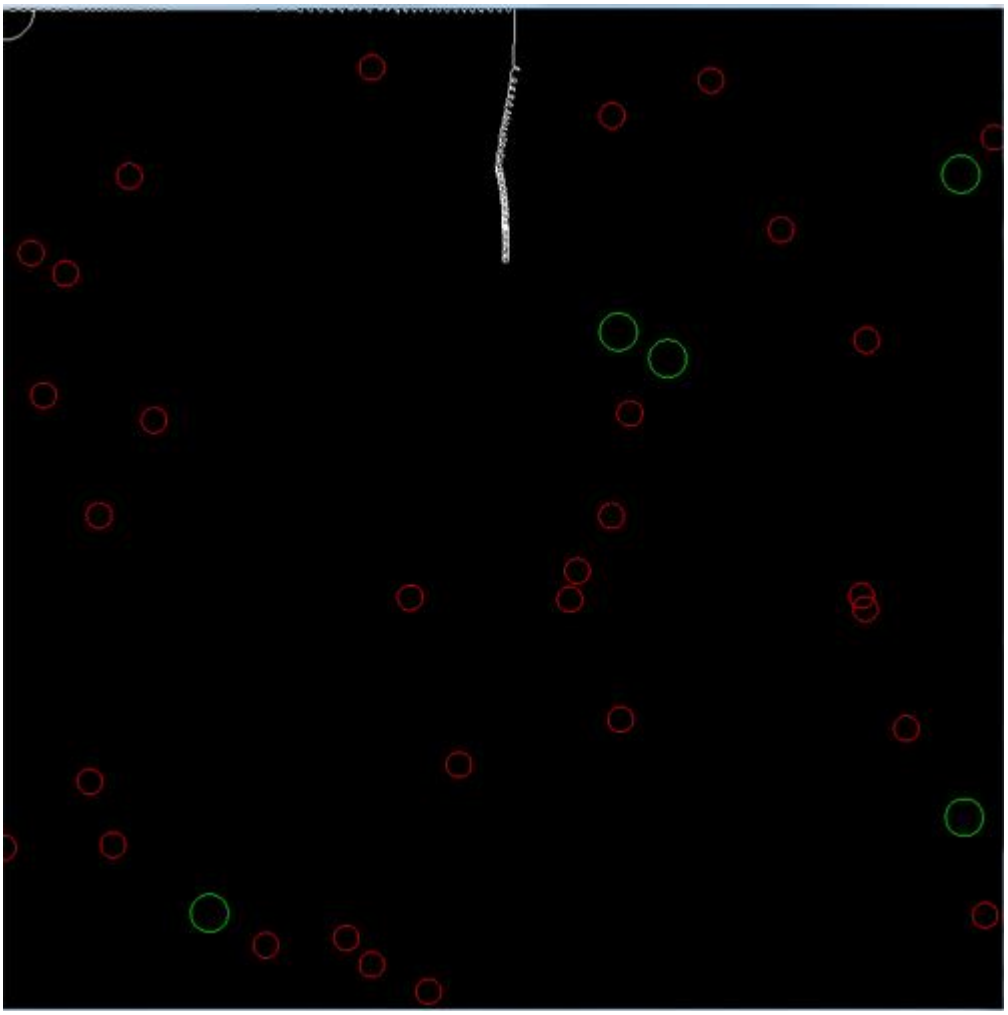
5.4 PRIMJER PUTANJE ROBOTA



Slika 5-5 Primjer kretanja robota za 20 keksa i 10 prepreka

Na slici 5-5 prikazano je ponašanje najbolje jedinke dobivene evolucijom za ploču sa 20 keksa i 10 prepreka. Robot uspijeva skupiti skoro sve kekse, osim jednog. Taj problem označen je na slici zelenim pravokutnikom. Taj keks robot neće skupiti jer se nalazi preblizu prepreci pa mu se ne može dovoljno približiti jer se aktiviraju senzori za izbjegavanje prepreka. Također se primjećuje zanimljivo ponašanje da nakon što je pojedena većina keksa robot jednostavno dođe do ruba ploče i počne kružiti oko poče držeći se ruba. To je zato jer je na ploči ostalo puno više prepreka nego keksa pa su robotu jače aktivirani senzori prepreka od senzora keksa te on izbjegava cijelo područje, tj. pokušava izbjeći cijelu ploču, te kruži samo po rubovima.

5.5 NEPRILAGOĐEN ROBOT



Slika 5-6 Robot neprilagođen okolini

Slika 5-6 pokazuje razlike u evoluciji za pojedine konfiguracije ploče. Ovdje je isti robot sa prethodnog primjera stavljen u drukčiju konfiguraciju ploče. Iako se na originalnoj ploči ponašao vrlo dobro, ovdje je potpuno neprilagođen. Naime robot je evoluirao za okoline gdje ima malo prepreka na ploči i samim time mogao si je priuštiti da prepreke zaobilazi u širokom luku. No takvo ponašanje ovdje nije prihvatljivo. Robot se previše „boji“ prepreka. Prvo odlazi do ruba ploče spiralnom kretnjom pokušavajući se držati što dalje od prepreka koje su mu sa svih strana, te se zatim kreće po rubu dok ne dođe u kut. Tamo se potom samo rotira oko svoje osi, jer se posvuda nalaze prepreke.

6 ZAKLJUČAK

Iako se neuronske mreže tradicionalno uče pomoću posebnih algoritama, kakav je npr. *backpropagation* algoritam, ovdje je predstavljena jedna alternativna metoda. Kombinirane su neuronske mreže i evolucijsko računanje, dvije tehnike rješavanja problema, koje su obje inspirirane biologijom. Konkretno, radi se o učenju neuronskih mreža primjenjujući postupke iz evolucijskog računanja. Ta metoda je uspješno implementirana i pokazalo se da može davati zadovoljavajuće rezultate. Kao i kod stvarne evolucije uočeno je da jedinke vrlo dobro funkcioniraju u okolini u kojoj su evoluirale, dok u drugačijim okolinama se ne ponašaju jednako dobro. Iako je konkretan problem na kojem je primijenjena dosta jednostavan, nije potrebno puno promjena kako bi se ta tehnika mogla primijeniti i u drugim, složenijim problemima. Koja metoda je brža, kvalitetnija i robustnija, možda ovisi o problemu na koji se primjenjuje, a evolucijsko računanje zasigurno ima svoje mjesto kod učenja neuronskih mreža.

LITERATURA

[1] Poli, R.; Langdon, B. W.; McPhee, F. N., A field guide to genetic programming, Lulu, 2008.

[2] Jakobović, D., Raspoređivanje zasnovano na prilagodljivim pravilima, doktorska disertacija, Fakultet elektrotehnike i računarstva, Zagreb, 2005.

[3] Stergiou C., Siganos D., Neural Networks, http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Neural%20networks%20versus%20conventional%20computers, 03.06.2010.

[4] Genetic algorithm, http://en.wikipedia.org/wiki/Genetic_algorithm, 04.06.2010.

[5] Artificial neural network, http://en.wikipedia.org/wiki/Artificial_neural_network, 04.06.2010.

[6] Pfeifer R.; Scheier C.; Understanding intelligence, The MIT Press, London, 2001.

SAŽETAK

U radu je istražena mogućnost učenja neuronskih mreža uporabom genetskih algoritama. Korištene neuronske mreže su vrlo jednostavne i sastoje se od vrlo malog broja neurona. Evolucija se provodila uporabom „*evolutionary computation framework*“ okvira. Zadaća razvijenih neuronskih mreža je bila kontrolirati robota kroz njegovo okruženje, tj. zadatak im je bio izbjegavati prepreke i skupljati kekse. Rezultati su pokazali da je ovakva metoda učenja neuronskih mreža primjenjiva, te da se mogu dobiti neuronske mreže koje se dobro snalaze u svojoj okolini.

Ključne riječi: genetski algoritmi, neuronske mreže, autonomni roboti

SUMMARY

This thesis deals with evaluating the possibility of using genetic algorithms as a learning method for artificial neural networks. Neural networks used in this paper are very simple and composed of only few neurons. Evolution is simulated using „evolutionary computation framework“, by which the evolved neural networks were used to navigate robots in their environment. Their main objective was to avoid obstacles and collect cookies. All of results have shown that this method of learning artificial neural nets is possible, and that it has the potential to produce good neural networks.

Key words: genetic algorithms, neural networks, autonomous robots