SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1832

AUTONOMNO UPRAVLJANJE ROBOTOM TEMELJENO NA GENETSKOM PROGRAMIRANJU

Zlatko Vlašić

Zagreb, siječanj 2010.

Sadržaj

Uvod

Uporaba računala u svrhu automatskog rješavanja problema je centralna stavka umjetne inteligencije, strojnog učenja i cijelog područja koje je Turing nazvao "strojna inteligencija". Pionir strojnog učenja Arthur Samuel je u svom radu "*AI: Where It Has Been and Where It Is Going*" napisao da je primarni cilj strojnog učenja i umjetne inteligencije "*navesti strojeve da pokazuju ponašanje koje bi, kad bi bilo prikazano od strane čovjeka, bilo smatrano inteligentnim*".

Genetsko programiranje je tehnika evolucijskog računarstva koja automatski rješava probleme bez potrebe za znanjem korisnika o obliku i strukturi rješenja. Na apstraktnoj razini, ono je domenski nezavisna tehnika kojom računala rješavaju probleme bez upletanja korisnika, počevši samo od opisa problema na vrlo visokoj razini.

Tehnika genetskog programiranja održava skup računalnih programa - populaciju, sa pridruženom mjerom kvalitete svakom programu - dobrotom, te na njih kroz više iteracija - generacija, primjenjuje operatore koji oponašaju evoluciju u prirodi. U genetskom programiranju ne baratamo cijelim skupom rješenja, već njegovim manjim dijelom, te na temelju različith tehnika uređujemo i oblikujemo programe u sve kvalitetnije i bliže željenom rješenju, te na kraju odabiremo onaj, ne nužno potpuno točan, koji nam na temelju predefiniranih kriterija najviše odgovara.

U prirodnom procesu evolucije, primjerice skupine od 1000 ljudi (populacija), trajanje generacije je prosječno 80 godina, a mutacije i selekcija nastaju zbog različitih prirodnih okolnosti. Ako bismo u takvom prirodnom procesu željeli pratiti razvitak i nasljeđivanje nekog svojstva kroz 200 generacija, to bi trajalo 1600 godina. Okvir genetskog programiranja stvara populaciju od 1000 „umjetnih" jedinki, provodi postupke mutacije i križanja, te odabire metode selekcije najboljih u generaciji, kroz 200 generacija, sve u trajanju od nekoliko minuta ili sati, ovisno o zahtjevnosti problema koji jedinke rješavaju.

Upravo utemeljenost na evolucijskim načelima „rastrošnosti" daje dodatnu prednost pred čovjekovim načinom promišljanja. Čovjekov um je često „zatvoren" u okvire vlastitog znanja ili čak i predrasuda, te zbog toga i ne traži rješenje izvan tih okvira. Metoda genetskog programiranja pristupa rješavanju problema kao prazna ploča, bez ikakvih predznanja, pa time i bez predrasuda.

Zbog općenitosti metodologije, prijeko potreban element za rješavanje problema ostaje ljudski čimbenik koji treba definirati skup informacija, temeljem kojih sustav genetskog programiranja može učiti i stvarati rješenja.

Robot je mobilan i manipulativan fizički sustav koji se autonomno giba kroz nestrukturirani prostor ostvarujući pritom interakciju s ljudskim bićima ili autonomno obavljajući neki posao umjesto njih. Autonomnim mobilnim robotom se smatra mobilni robot koji je sposoban samostalno se gibati kroz prostor bez bilo kakve pripreme prostora, te pri tome obavljati postavljeni mu zadatak.

Pri izgradnju upravljačkih sustava za autonomne mobilne roboti koriste različite pristupi. Konvencionalan pristup izgradnji upravljačkih kontrolnih sustava za autonomne mobilne

robote je dekompozicija problema u skup jedinica od kojih svaka obavlja funkciju poput percepcije svijeta, modeliranja, planiranja, obavljanja zadaća ili kontrole motora. Centralni kontrolni sustav pokreće svaku od ovih jednici i predaje podatke iz prethodne jedinice slijedećoj.

Alternativa gore opisanom pristupu izgradnji upravljačkih sustava je dekompozicija problema u skup asinkronih ponašanja koja rješavaju zadatke (engl. *task achieving behaviours).* U tom pristupu ukupna kontrola nad robotom je ostvarena kolektivnim radom interakcija između tih primitivnih ponašanja, a koja sva komuniciraju sa svijetom i međusobno. Svako od tih ponašanja najčešće obavlja nekvu jednostavnu funkciju, npr. tumaranje, prepoznavanje objekata i slično. Ta ponašanja su često primitivnija od funkcijskih jedinica iz konvencionalnog pristupa izgradnji upravljačkih sustava.

U ovom radu se, koristeći tehniku genetskog programiranja, pokušava evoluirati upravljački program za autonomne mobilne robote temeljen na gore opisanom alternativnom pristupu. Cilj upravljačkog program je praćenje zidova nepravilno oblikovane prostorije. Kako bi se moglo dati ocjenu kvalitete pojedinog upravljačkog programa, prostor uz zidove prostorije dijeli se na kvadratne ćelije. Cilj upravljačkog programa je provesti robota kroz sve rubne ćelije prostorije u minimalnom mogućem vremenu. Upravljački program koji provede robota kroz više ćelija od drugoga će se smatrati kvalitetnijim, a od robota koji obilaze isti broj ćelija bolji upravljački program će imati onaj koji to obavi u kraćem vremenu.

U radu se promatra je li genetsko programiranje prikladna tehnika za zadani problem, te, ako jest, kakav je utjecaj različitih parametara, a posebno primjera nad kojim će naš sustav učiti, na kvalitetu rada sustava i na kvalitetu rezultirajućih jedinki – upravljačkih programa.

Valja napomenuti da ručni razvoj upravljačkih programa nije jednostavan posao i traži izniman trud i sposobnost čovjeka – programera. U ovom radu se pokušava navesti sustav genetskog programiranja da obavi taj posao umjesto programera. Na nikakav način se ne upleće u rad sustava, potrebno je samo kvalitetno prilagođavanje jedinke upravljačkim programima i kvalitetan opis dobrote jedinki, ostalo je na sustavu. Evolucijski proces je navođen isključivo dobrotom programa koji rješavaju problem.

1 Genetsko programiranje

Genetsko programiranje je tehnika koja omogućuje rješavanje nekog problema računalom bez potrebe za stvaranjem programa koji rješava zadani problem. Genetsko programiranje koristi evolucijski proces u stvaranju računalnih programa. Evolucija se u prirodi manifestira kao preživljavanje i reprodukcija sposobnih jedinki koje predaju svoj genetski kôd sljedećim generacijama. Genetski kôd koji čini sposobnije jedinke ima mogućnosti povećati svoj udio u cjelokupnoj populaciji, što uzrokuje promjene na razini vrste.

Genetsko programiranje je dio šire skupine algoritama zajedničkim imenom nazvanih evolucijsko računarstvo. Evolucijsko računarstvo (engl. *evolutionary computation, EC*) je skup postupaka koji oponašaju prirodni evolucijski proces na računalu. Ostvareni optimizacijski program koji se izvodi na računalu i oponaša evolucijski proces nazivamo evolucijskim algoritmom. Identično tome, računalni program koji izvodi postupak genetskog programiranja nazivamo genetskim programom. Postupci evolucijskog računanja dijele se 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*).

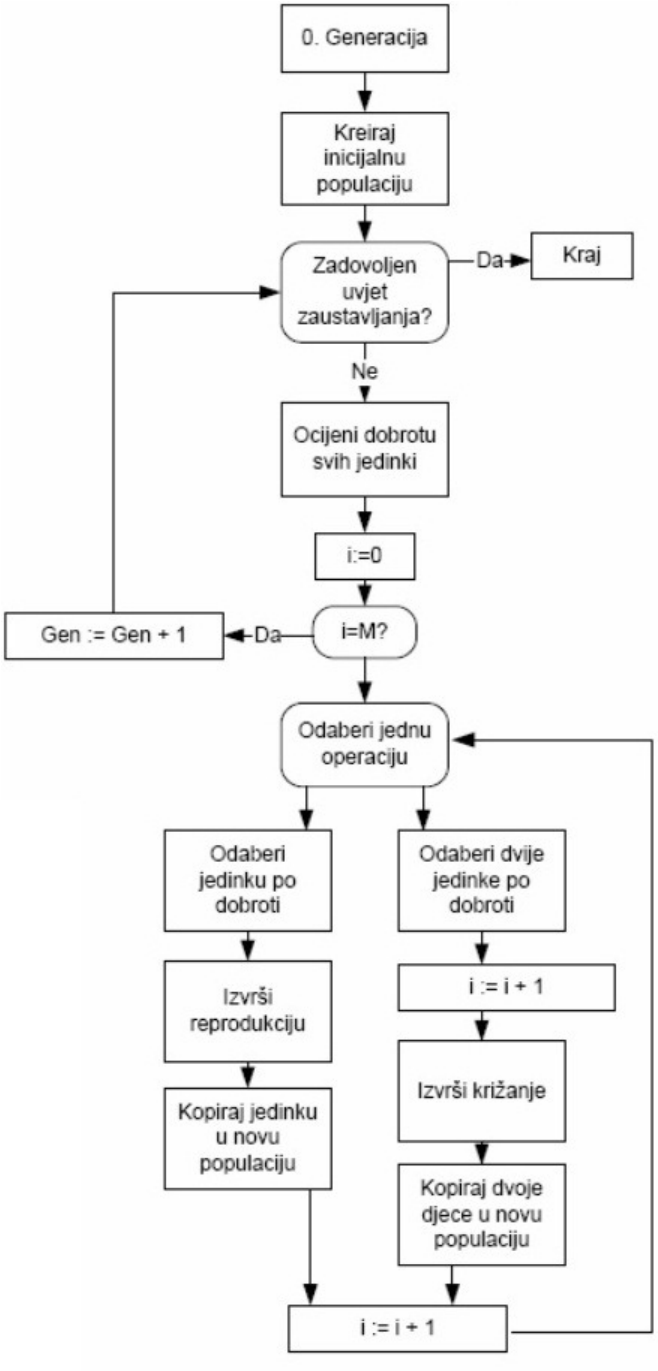
Posebnost genetskog programiranja, u odnosu na ostale spomenute tehnike, je u činjenici da jedinke u populaciji genetskog programa predstavljaju računalne programe, odnosno strukture koje se mogu jednoznačno preslikati u oblik pogodan za izvođenje na računalu. Kako bi se olakšao proces stvaranja novih programa iz jednoga ili više roditeljskih, programi su u većini ostvarenja genetskog programiranja napisani u obliku stabla. Novi se programi dobivaju jednostavnim manipulacijama nad već postojećim stablima - primjerice, uklanjanjem grana iz jednoga stabla te dodavanjem tih grana na određeno mjesto u drugom stablu. Ovaj jednostavni proces kao rezultat također daje stablo i ujedno osigurava sintaktičku ispravnost dobivenih rješenja.

1.1 Elementi genetskog programiranja

Sve paradigme evolucijskog računarstva, pa tako i genetsko programiranje, traži dobro definirane elemente sustava, strukture, koji prolaze adaptaciju (učenje). To su:

- inicijalne strukture
- funkcija dobrote
- operacije za modifikaciju struktura
- stanje sustava u svakom koraku
- metoda zaustavljanja sustava
- metoda određivanja rezultata i parametri sustava

Slika 1.1 prikazuje dijagram toka genetskog programiranja. Svi gore navedeni elementi sudjeluju u tom procesu, kao strukture koje se modificiraju ili kao parametri koji upravljaju procesom evolucije.



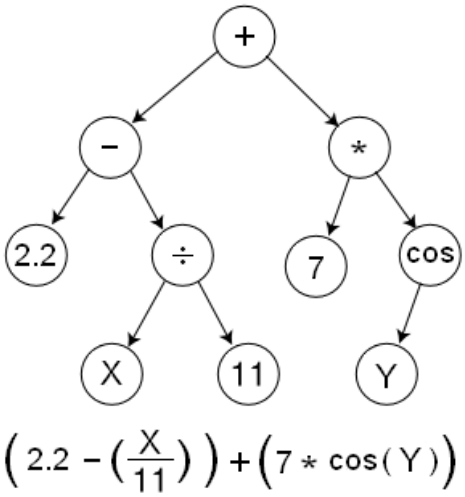
Slika 1.1 Dijagram toka genetskog programiranja

1.2 Inicijalne strukture

Genetsko programiranje barata skupom računalnih programa sa pripadnom mjerom kalitete, dobrotom (engl. *fitness*). Prvi korak rada je generiranje početne populacije, početnog skupa programa.

Iako postoji više različitih načina prikaza (reprezentacije) programa po pitanju memorijske zahtjevnosti, računalne složenosti i složenosti primjene genetskih operatora najbolji izbor

je stablo. Slika 1.2 prikazuje jedan računalni program, jednostavnu matematičku formulu, u obliku stabla.



Slika 1.2 Primjer računalnog programa prikazanog stablom

Odabir ispravnog načina prikazivanja pojedine jedinke je najvažniji preduvjet da bi neki GP mogao pronaći optimalno rješenje zadanog problema. Prikaz jedinke mora s minimalnim brojem gena predočiti sva važna svojstva jedne jedinke. Genetski zapis jedinke mora biti takav da prilikom primjene genetskih operatora ne dolazi do nemogućih rješenja.

Čvor stabla može biti ili operator (unutarnji čvorovi) ili završni znak (vanjski čvorovi, listovi). Primjeri operatora su aritmetičke operacije (npr. zbajanje i oduzimanje), logičke operacije (npr. AND i NOT), uvjetni operatori (npr. IF...THEN) i slično. Završni znakovi su varijable i konstante rješenja te operacije bez argumenata.

Bitni parametri za izgradnju početne populacije su broj jedinki, maksimalna dubina stabla i metoda izgradnje. Jedinke se najčešće grade potpuno nasumično, iako je moguće ubaciti i predefinirana rješenja. Metode izgradnje početne populacije su: *grow, full i ramped half-and-half.*

1.2.1 *Grow* metoda

Jedinke se generiraju prema slijedećem pravilu:

- počevši od korjenskog elementa, slučajno se odabire vrsta čvora koji će se dodati u stablo; pritom je moguće birati operatore ili završne znakove;
- ako je izabran završni znak, sustav odabire jedan iz skupa završnih znakova;
- ako je odabran operator, odabire se jedan iz skupa operatora.

Postupak se dalje rekurzivno ponavlja dok svi novo dodani znakovi nisu završni ili dok se ne dostigne maksimalna dozvoljena dubina stabla *m*. Jedinke kreirane ovom metodom neće imati stabla jednake dubine, a moguće je da će se izraz sastojati i od samo jednoga završnog znaka. Općenito, ovakvi izrazi nemaju neku veću vrijednost jer u općenitom

slučaju neće dati dobro rješenje, pa se parametar minimalne dubine stable postavlja na vrijednost veću od 2. Problematična može biti i specifična brojnost skupova operatora i završnih znakova. Ako skup završnih znakova ima mnogo više elemenata od skupa operatora, velika je vjerojatnost da će stabla izgrađena *grow* metodom biti vrlo plitka. Ako skup operatora ima mnogo više elemenata od supa završnih znakova, većina stabala će biti vrlo duboka.

1.2.2 *Full* metoda

Kod ove metode se pri izgradnji koriste operatori dok se ne dostigne određena predefinirana dubina, od kojeg se trenutka za čvorove djece počinju koristiti završni znakovi.

1.2.3 *Ramped half-and-half* metoda

U ovoj se metodi prilikom generiranja jedinki unaprijed definira samo maksimalna dubina *md*. Prednost ove metode je što generira populacije s dobrom raspodjelom jedinki po veličini i strukturi.

Kreiranje jedinki se obavlja na slijedeći način:

- populacija se podijeli na *md-1* dijelova;
- polovica svakoga dijela generira se metodom *grow*, a druga metodom *full*. Za prvi dio parametar *m* za metodu grow i *d* za metodu *full* je 2, za drugi dio je 3. Niz se nastavlja sve do *md-1* dijela na kojem se koristi *md* kao vrijednost za *m* i *d.*

1.3 Dobrota i mjera dobrote

U prirodi, dobrota jedinke je vjerojatnost da preživi do trenutka reprodukcije i stvori potomke. Analogno, u svijetu genetskog programiranja dobrota je mjera kvalitete rješenja u zadanome prostoru rješenja. Jedinke veće dobrote imaju veću vjerojatnost preživjeti do slijedeće generacije ili stvoriti potomke. Funkcija dobrote je element sustava koji u najvećoj mjeri rukovodi evolucijom i upravlja kretanjem cjelokupne populacije. Od velikog je značaja da funkcija dobrote nagrađuje ne samo bolja rješenja, već i sva poboljšanja pronađena tijekom evolucijskog procesa. U većini ostvarenja genetskog programiranja određivanje dobrote rješenja oduzima najveći dio utrošenog procesorskog vremena, pa treba tražiti što jednostavniji i učinkovitiji opis i način računanja dobrote.

Dobrota se može mjeriti na više eksplicitnih i implicitnih metoda. Najčešći pristup je stvoriti eksplicitnu mjeru dobrote svake jedinke u populaciji. Svakoj jedinci je pripisana skalarna vrijednost dobrote putem neke dobro definirane eksplicitne metode evaluacije. Dobrota se može računati i ko-evolucinarno, npr. kada se dobrota strategije igranja igre stvara tako da se strategija sučeljava sa drugim strategijama iz populacije. Postoji više vrsta prilagodbe dobrote, od kojih su neke: izravna (eng. *raw*), standardizirana (eng. *standardized*), podešena (eng. *adjusted*) i normalizirana (eng. *normalized*).

Kod jednostavnih problema, kao što je pronalaženje maksimuma neke funkcije, dobrotu možemo prikazati kao vrijednost koju ta funkcija ima za zadani kromosom. U tom slučaju veća vrijednost funkcije označava veću dobrotu rješenja. Kod složenijih problema dobrota se može izraziti kao vrijeme trajanja projekta, vrijeme koje je potrebno da se obavi simulacija sustava koji optimiramo i sl.

1.4 Odabir operatora i završnih znakova

Prilikom odabira operatora i završnih znakova bitno je da oni zadovoljavaju dva svojstva, potpunost i zatvorenost.

Potpunost je svojstvo koje označava sposobnost da se skupom operatora i završnih znakova može izraziti rješenje problema kojega pokušavamo riješiti. Zadovoljavanje istoga ovisi o određenom problemu i nije ga u općenitom slučaju moguće definirati. U najvećem broju primjena u skup podatkovnih i funkcijskih elemenata stavljaju se svi elementi za koje se smatra da bi mogli biti od koristi u dotičnom problemu. Sustav u kojem bi nedostajao neki vitalni element (npr. korijen pri traženju rješenja kvadratnih jednadžbi) neće moći pronaći adekvatno rješenje, dok će sustav koji ima uključene suvišne, redundantne, elemente biti sporiji i neefikasniji, no imat će dovoljnu ekspresivnost da opiše ispravno rješenje.

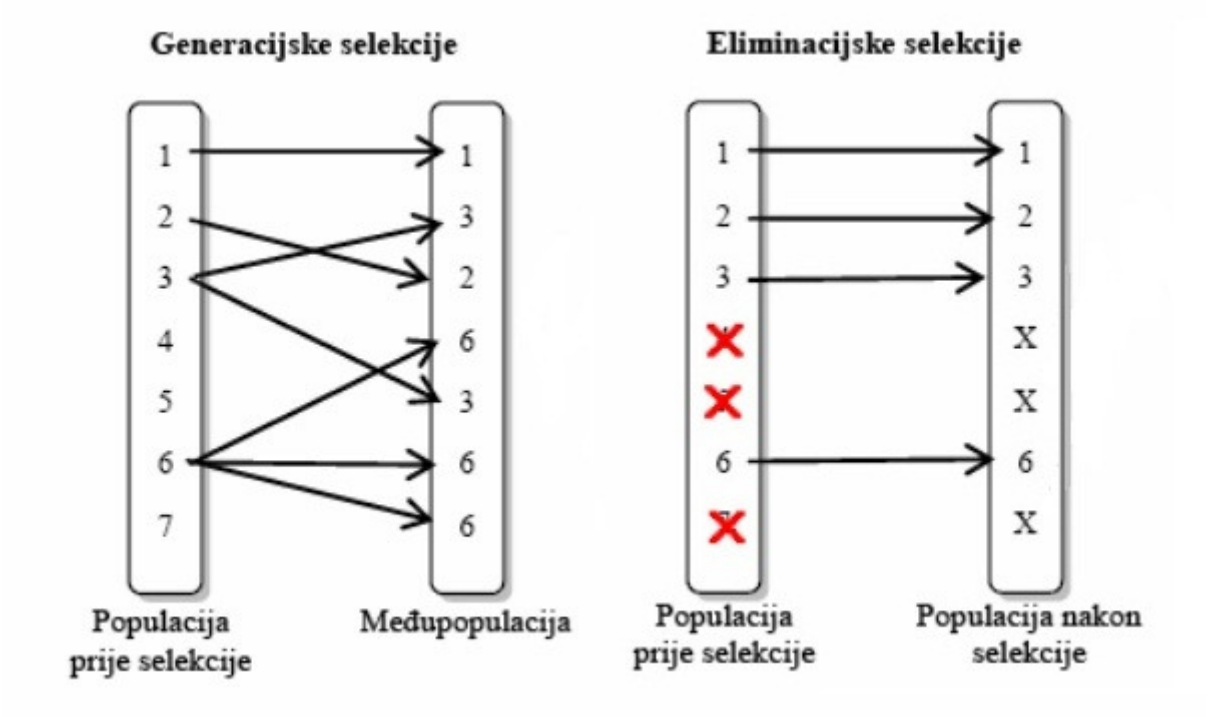
Zatvorenost (engl. *closure)* se definira kao sposobnost operatora da prihvati rezultat od bilo kojeg drugog operatora ili završnog znaka. Zadovoljavanje zatvorenosti ovisi o vrsti podataka svojstvenoj za određeni problem. Ako, npr., rješavamo problem u domeni booleove algebre i koristimo isključivo logičke operatore i logičke vrijednosti 0 i 1, zatvorenost je postignuta, no problem se javlja ako u skup operatora ubacimo npr. korijen. Tada je nužno definirati pretvorbu ili drugačiju interpretaciju podataka za taj operator. Zatvorenost može biti narušena i ponašanjem nekih funkcija: rezultat dijeljenja nije definiran ukoliko je djelitelj jednak nuli. Na dva načina se pristupa tome problemu, ili redefiniranjem funkcije (npr. djeljenje s nulom kao rezultat daje 0 ili 1), ili definiranjem nevaljalog tipa podataka.

1.5 Selekcija

Selekcija se u prirodi obavlja se evaluacijom skupa značajki nastalih na temelju genotipa i kao posljedica interakcije sa okolinom jedinke. Prirodno okruženje odabire one jedinke koje imaju najkvalitetniji genetski materijal. Najbolje jedinke najčešće sudjeluju u reprodukciji, tako da se najveći dio tog dobrog genetskog materijala prenosi na njihove potomke. S vremenom prosječna kvaliteta genetskog materijala postaje sve bolja i bolja. U prirodi taj proces ipak nije egzaktan i moguće je da u nekoj vrsti nakon dugog perioda poboljšavanja nastupi period u kojem prosječna kvaliteta genetskog materijala postaje sve lošija. Prirodna selekcija je mehanizam koji se stalno mijenja. U jednoj „iteraciji“ prirodne selekcije jedinke s određenim svojstvima će biti najbolje, a u nekoj drugoj „iteraciji“ evolucija može cijeniti neka drugačija svojstva. Važan faktor je i određena doza slučajnosti jer nije moguće sa sigurnošću utvrditi na koji način se odabiru pojedine jedinke kao najbolje.

U genetskom programiranju selekcija je odabir jedinki nad kojima će se primijeniti genetski operatori. Selekcijske metode preferiraju jedinke s većom dobrotom i bitne su jer time osiguravaju prenošenje kvalitetnog genetskog materijala u slijedeće generacije. No, poželjno je da i jedinke sa manjom dobrotom mogu sudjelovati u genetskim operacijama, čime se čuva raznolikost genetskog materijala i izbjegava zapinjanje sustava u lokalnim optimumima.

Prema načinu prenošenja genetskog materijala selekcije se dijele na generacijske selekcije kod kojih se odabiru najbolje jedinke iz trenutne generacije i od njih se kreira novu generaciju, te eliminacijske selekcije, kod kojih se eliminiraju najgore jedinke iz generacije. Slika 1.3 prikazuje oba načina selekcije.



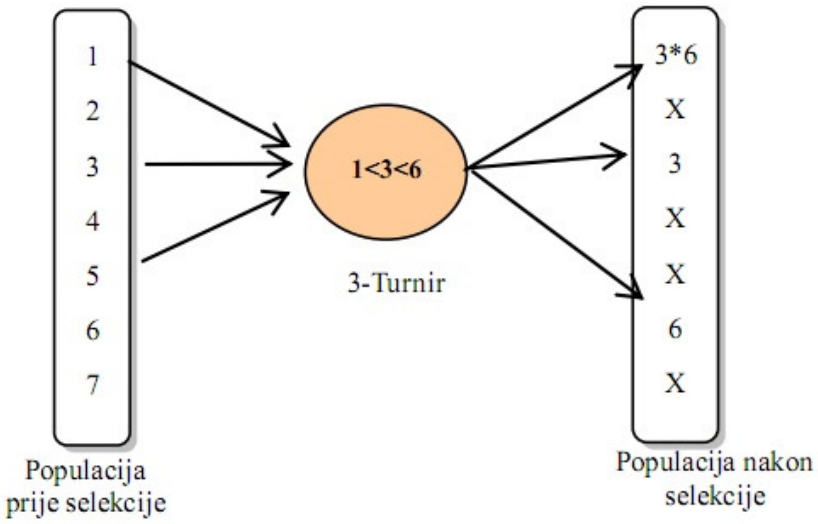
Slika 1.3 Selekcija prema načinu prenošenja genetskog materijala

Generacijske selekcije djeluju tako da iz populacije odabiru određen broj najboljih jedinki i od njih se stvara međupopulacija. Broj jedinki koje prežive selekciju je manji od veličine populacije pa se jedinke koje nedostaju nadomještaju kopiranjem već selektiranih jedinki. Nedostatak ovoga načina selekcije je istovremeno postojanje populacije i međupopulacije u istom spremniku. Drugi nedostatak je proces višestrukog kopiranja istih jedinki, što rezultira nekvalitetnim (vrlo sličnim) genetskim materijalom.

Kod eliminacijskih selekcija nema stroge granice između prethodne i sljedeće generacije. Tijekom procesa eliminacije odstranjuju se slučajno odabrane jedinke (lošije jedinke imaju veću vjerojatnost eliminacije od kvalitetnijih). Eliminirane jedinke nadomještaju se križanjem postojećih.

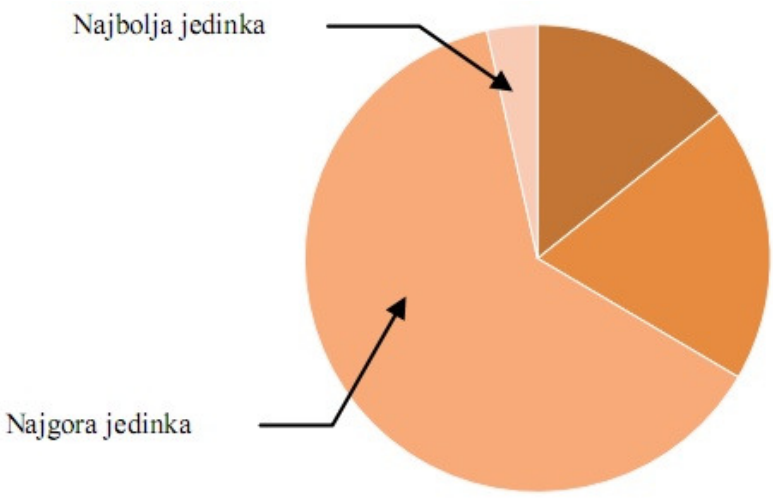
Selekciju možemo definirati i prema načinu odabira pojedinih jedinki:

- k-turnirska selekcija - iz populacije se odabire k jedinki, te se od tih k jedinki odabire ona najveće dobrote. Slika 1.4 prikazuje jednu od mogućih implementacija turnirske selekcije u kojoj su od više odabranih jedninki eliminirane one slabije dobrote, te su nadomještene primjenom križanja nad ostalim, kvalitetnijim, jednikama iz turnira. Postupak se ponavlja dok nije stvorena komplatna slijedeća generacija. Turnirska selekcija je popularna jer je jednostavna za implementaciju, brza i daje povoljne rezultate;



Slika 1.4 Troturnirska selekcija

- rangirajuća selekcija - jedinke se poredaju po dobroti, pa se selekcija obavlja na temelju pozicije na tablici;
- selekcija proporcionalna dobroti - vjerojatnosna selekcija u kojoj jedinke sa većom dobrotom imaju veće šanse biti odabrane od jedinki s manjom dobrotom. Takvu selekciju možemo zamisliti kao kotač ruleta, na kojem najbolja jedinka zauzima najveću površinu, a jedinka najmanje dobrote najmanju površinu, kao što prikazuje slika 1.5.



Slika 1.5 Selekcija proporcionalna dobroti

1.6 Reprodukcija

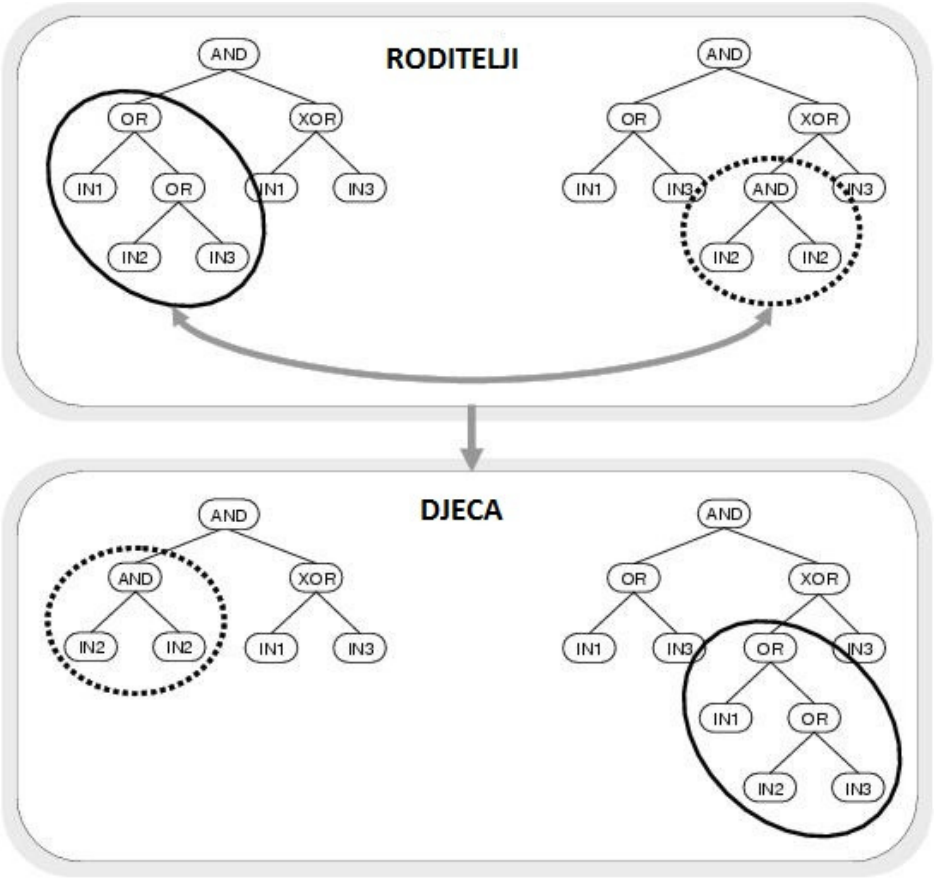
Reprodukcija je aseksualan proces (sudjeluje samo jedan roditelj) u kojem se selektira jedna jedinka iz trenutne generacije i nepromijenjena prenosi u drugu. Najvažniji parametar kod odabira jedinke za reprodukciju je način selekcije. Reprodukcijom jedinke u slijedeću generaciju ne briše se ona u staroj generaciji te je moguće da ista bude opet odabrana.

1.7 Križanje

Križanje jedinki je proces u kojem se rekombinira genetski materijal dvaju roditelja. Rezultat križanja su dvije jedinke djece, koje od svakog roditelja nasljeđuju dio genetskog materijala.

Slika 1.6 prikazuje izvođenje jedne operacije križanja. Tijek je definiran sa:

- slučajno odaberi točku prekida za oba roditelja
- za točku prekida se uzima jedan čvor stabla
- izdvoji podstabla ispod točke prekida
- izdvajaju se podstabla oba roditelja
- zamijeni izdvojena podstabla



Slika 1.6 Primjer genetske operacije križanja

Odabrana podstabla se zamjenjuju. Novonastale jedinke nemaju nužno veću dobrotu od svojih roditelja, no selekcijski pritisak će se pobrinuti da lošije jedinke kroz par generacija nestanu, a bolje opstanu i dalje evoluiraju. Ponekad će, zbog specifične brojnosti skupa terminala i operatora, većina čvorova stabla biti listovi. Tada bi oni imali veliku šansu da budu odabrani kao točke prekida, što bi bilo loše jer bi promjene na stablima bile minimalne. Iz tog razloga je kao parametar poželjno uvesti veću vjerojatnost odabira unutrašnjeg čvora kao točke prekida (Koza preporuča 90% za operator i 10% za završni znak (Koza 1992)).

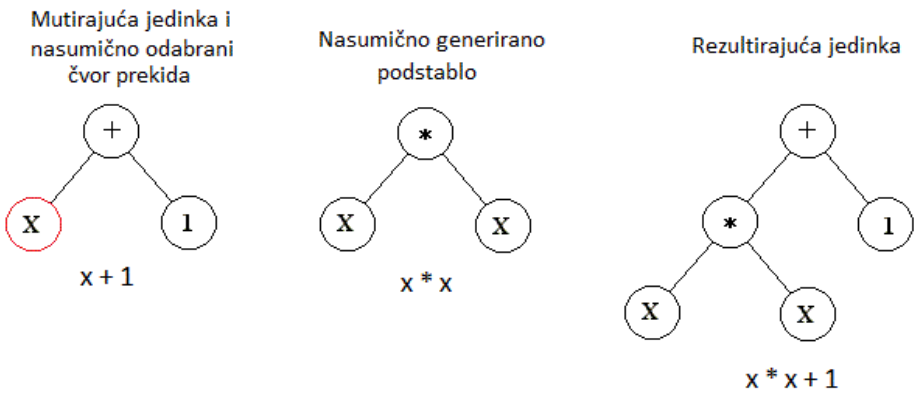
U većini sustava genetskog programiranja unutar jedne generacije je 90% jedniki dobiveno križanjem jedinki iz prošle generacije, pa je bitno da križanje kao operacija u računalu bude izvedena jednostavno i efikasno. To se postiže zapisom jedinki u obliku stablala, jer se križanje obavlja iznimno brzo, referenciranjem čvorova stabla na druga podstabla.

1.8 Mutacija

Mutacija u prirodi je bilo koja promjena genetskog materijala, a najčešće nastaje kao greška u kopiranju prilikom procesa dioba stanica. Velik broj mutacija pokazuje se štetnim za organizam, međutim, da bi proces evolucije bio potpun, potrebne su stalne promjene. Dobre mutacije ugradit će se u organizam i nastaviti prenositi na nove jedinke u slijedećim generacijama, dok će loše mutacije mogu dovesti do odumiranja jedinki.

Mutacija služi kako bi se u nove jedinke uveli neki geni koji će rezultirati novim jedinkama s boljim rješenjem. Općenito gledajući, mutacijama se može dobiti poboljšanje ali i pogoršanje prosječne dobrote jedinki u populaciji. Jedinke sa "smrtonosnim" mutacijama biti će vrlo brzo eliminirane u selekcijskom procesu, tako da iste ne bi smjele dugotrajno narušiti prosječnu dobrotu jedinki u populaciji. Slika 1.7 prikazuje mutiranje jedinke. Mutacija se za jedinke najčešće radi na slijedeći način:

- iz populacije se odabere mutirajuća jedinka;
- u stablu jedinke nasumce se odabere jedan čvor;
- odabrani čvor i sva njegova podstabla se brišu iz stabla;
- na odabranome mjestu se slučajnim odabirom stvara novo podstablo; veličina i način stvaranja novog podstabla definiraju se prilikom pokretanja algoritma.



Slika 1.7 Primjer genetske operacije mutacije

1.9 Stanje sustava

U genetskom programiranju stanje sustava se u bilo kojem trenutku sastoji isključivo od trenutne populacije jedinki. Nikakvo dodatno čuvanje informacija nije potrebno. U računalnoj implementaciji paradigme potrebno je još čuvati kontrolne parametre, skup operatora i završnih znakova (ako se koristi mutacija), te najbolju jedinku do tog trena izvođenja ako se tako određuje najbolji rezultat.

1.10 Metode zaustavljanja sustava

Paradigma genetskog programiranja je, kao i evolucija u prirodi, neprekidni proces. No, zbog praktičnih razloga, izvođenje zaustavljamo kada se ispuni kriterij zaustavljanja.

Najčešće korišteni kriteriji su dostizanje određene generacije evolucijskog procesa. U svom radu (Koza 1992) Koza za sve eksperimente koristi broj od 50 generacija kao uvjet zaustavljanja postupka, ukoliko prije zadnje generacije nije pronađeno željeno rješenje. Argument iza ove vrijednosti je zapažanje iz većeg broja problema. Nakon tog broja generacija genetski program gubi sposobnost pronalaženja bitno različitih rješenja, te su eventualna naknadna poboljšanja vrlo mala. Autor također tvrdi da je u većini slučajeva isplativije poduzeti više eksperimenata nego povećati broj generacija u jednom od njih.

Moguće su i opcije zaustavljanja izvođenja nakon što smo generirali jedinku sa određenom razinom dobrote (to može biti u potpunosti točno rješenje, ali i, npr. u optimizacijskim problemima, samo zadovoljavajuće rješenje) ili nakon što nakon kroz nekoliko generacija nema bitnog napretka u dobroti najbolje jedinke.

1.11 Odabir rezultata

Dvije metode se koriste za odabir najboljeg razultata. Jedna je spremanje ukupno najbolje jedinke, iz svih generacija, dok je druga biranje najbolje jedinke iz zadnje generacije. Ako se koristi elitizam (najbolja jedinka iz svake generacije se sigurno reproducira u slijedeću), onda ne treba spremati najbolju jedinku iz svih generacija, dovoljno je uzeti najbolju iz zadnje.

1.12 Kontrolni parametri

Paradigma genetskog programiranja upravlja se sa 19 kontrolnih parametara, koji uključuju dvije primarne numeričke vrijednosti, jedanaest sekundarnih numeričkih vrijednosti i šest kvalitativnih varijabli koji određuju na koji način će se izvoditi algoritam.

Dva primarna numerička parametra su:

- M - broj jedniki u populaciji (npr. 500);
- G - maksimalan broj generacija (npr. 50).

Jedanaest sekundarnih numeričkih parametara su:

- p_c - vjerojatnost križanja (npr. 0,9);
- p_r - vjerojatnost reprodukcije (npr. 0,1);
- p_{ip} - vjerojatnost da pri križanju bude izabran unutarnji čvor (operator), a ne list (npr. 0,9);
- $D_{created}$ - maksimalna dubina koju stablo može postići nakon križanja (npr. 17);
- $D_{initial}$ - maksimalna dubina koju stablo može postići prilikom generiranja inicijalne populacije (npr. 16);
- p_m - vjerojatnost mutacije (npr. 0,01);
- p_p - vjerojatnost permutacije;
- f_{er} - frekvencija upotrebe operatora pojednostavljivanja;
- p_{en} - frekvencija upotrebe enkapsulacije;
- uvjet pokretanja brisanja populacije;
- p_d - postotak brisanja populacije.

Šest kvalitativnih varijabli za kontrolu izvođenja su:

- metoda generiranja početne populacije;
- metoda selekcije jedinke za reprodukciju i jedinke za prvog roditelja u operaciji križanja;
- metoda selekcije jedinke za drugog roditelja u operaciji križanja;
- mjera dobrote;
- upotreba prenaglašene selekcije boljih jedinki;
- upotreba elitizma.

1.13 Prednosti i nedostaci tehnike genetskog programiranja

Genetskim programiranjem može se riješiti proizvoljni optimizacijski problem. Također, jednom kada se genetskim programiranjem pronađe rješenje ono ne predstavlja rješenje odabranog problema za konkretne vrijednosti upotrijebljene za učenje, već način, odnosno algoritam rješavanja, koji se može primjenjivati nad proizvoljnim podacima. Genetskim programiranjem nastaje čitava populacija rješenja. Time se omogućuje odabir nekoliko najboljih jedinki na temelju kojih se naknadno može odabrati najbolje dobiveno rješenje ovisno o konkretnom primjeru. Prednost genetskog programiranja je i jednostavnost izvedbe. Naime, jednom kada se definiraju osnovni elementi, cijeli postupak izgradnja rješenja je samostalan i ne traži interakciju sa čovjekom, sve se prepušta genetskom programu.

Genetsko programiranje ima i svoje nedostatke. Prostor rješenja koji se pretražuje je najčešće iznimno velik, pa je izvedba, iako relativno jednostavna, računarski i memorijski zahtjevna, te dugotrajna. Također, rješenja koja genetsko programiranje proizvodi nemaju ograničenu duljinu pa i ona mogu biti prekompleksna. Problem je i potreba za prilagodbom izvornog problema ili algoritma u oblik kojim paradigma genetskog programiranja može baratati. Izražen je utjecaj parametara čije ugađanje može biti dugotrajan i složen proces. Nepoznavanje svojstava rješenja predstavlja problem kod definiranja osnovnih elemenata, poput skupa građevnih elemenata ili funkcije dobrote. Uz to, ne može se postići stopostotna učinkovitost, a problem predstavlja i zaglavljivanje u lokalnim optimumima, te prilagodba rješenja primjerima za učenje.

2 Strukture upravljanja autonomnim robotima i problem praćenja zidova

Roboti su, prema ISO 8873, automatski upravljani, reprogramirljivi, višenamjenski manipulatori programirljivi u tri ili više osi, koji mogu biti stacionarni ili mobilni, za primjene u industrijskoj automatizaciji. Može se reći i da je robot mobilan i manipulativan fizički sustav koji se autonomno giba kroz nestrukturirani prostor ostvarujući pritom interakciju s ljudskim bićima ili autonomno obavljajući neki posao umjesto njih.

Autonomnim mobilnim robotom se smatra mobilni robot koji je sposoban samostalno se gibati kroz prostor bez bilo kakve pripreme prostora, te pri tome obavljati postavljeni mu zadatak.

2.1 Strukture sustava upravljanja mobilnim robotom

Velik je broj raznih struktura upravljanja, ali se u pravilu mogu razvrstati u četiri skupine:

- modelske strukture upravljanja (eng. *deliberative control*) – sve isplaniraj (razmišljaj), pa tek zatim djeluj:
- reaktivne strukture upravljanja (eng. *reactive control*) – ne razmišljaj, djeluj (reagiraj):
- ponašajne strukture upravljanja (eng. *beahviour based control*) – razmišljaj o načinu na koji djeluješ:
- hibridne strukture upravljanja (eng. *hybrid control*) – razmišljaj i djeluj istodobno.

Modelska struktura, prikazana na slici 2.1, bila je dominantna strutktura upravljanja autonomnim mobilnim robotima sve do 1985. Kod nje se izgrađuje simbolički model okruženja kroz senzorsku fuziju, uzimajući u obzir sve ciljeve koje robot treba ostvariti. Podzadaci se razlažu u serijski niz funkcionalnih slojeva (horizontalno razlaganje sustava upravljanja), slično klasičnom pristupu umjetnoj inteligenciji.

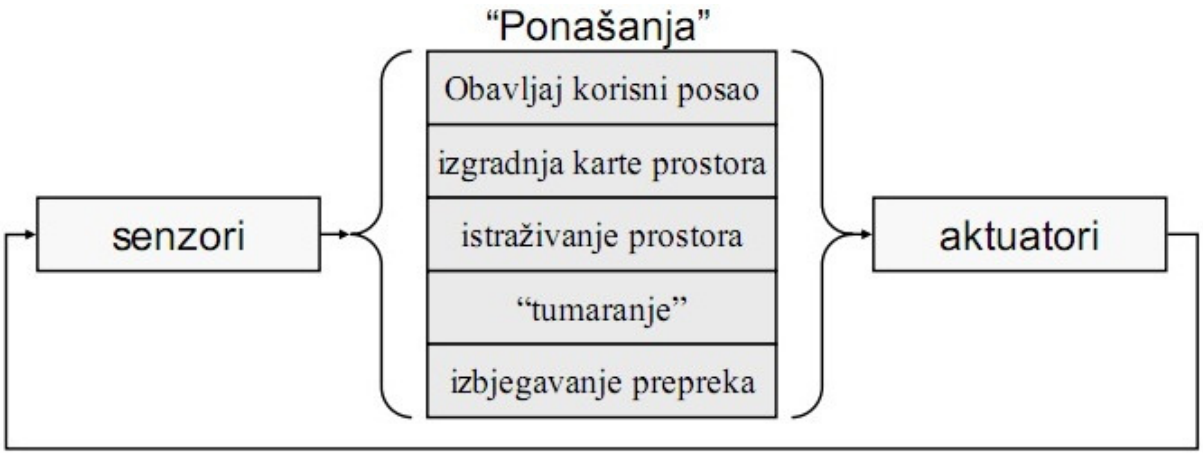
Slika 2.1 Modelska struktura upravljanja

Prednosti modelske arhitekture su sposobnost učenja i predikcije, te mogućnost pronalaženja strateških rješenja. Najveći nedostatak je što se adekvatna, precizna i aktualna karta prostora mora stalno održavati, što je teško ostvariti. Uz to, ova struktura je „krhka“,

kvar u bilo kojem funkcionalnom sloju u potpunosti onemogućuje daljni rad robota. Također, potrebna je i velika procesna snaga, zbog složenog osvježavanja modela prostora, a rezultat toga je spor odaziv.

Reaktivni sustav upravljanja autonomnim mobilnim robotima koristi tehniku koja neposredno povezuje percepciju i akciju (djelovanje). Osigurava brzi odaziv u promjenjivim, nestrukturiranim okruženjima. Prednost ove strukture je vrlo brzo vrijeme reakcije, gotovo trenutačno, a nedostaci su nepostojanje memorije i modela prostora i nemogućnost planiranja i učenja.

Kod ponašajnog modela, prikazanog na slici 2.2, sustav upravljanja razlaže se u vodoravne „ponašajne“ module (engl. *behaviours*), koji se izvode paralelno. Svaki sloj ima izravan pristup senzorima i može neposredno davati naredbe motorima robota.



Slika 2.2 Ponašajna struktura upravljanja

Ponašanja su moduli sustava upravljanja robotom koji predstavljaju pobudno-djelujući par za zadano okruženje. Pravila izvedbe su slijedeća:

- ponašanja moraju postizati ili održavati određeni cilj (npr. dolazak u početni položaj ili praćenje cilja)
- ponašanja koriste ulaze od senzora i drugih ponašanja, a šalju izlaze na aktuatore ili druga ponašanja
- ponašanja su složenija od aktivnosti (npr. aktivnosti stop ili skreni desno, naspram ponašanja slijedi cilj, pronađi predmet itd.)

Ponašanja se izvode istodobno i neovisno, što nam daje sposobnost djelovanja u stvarnom vremenu. Mreža ponašanja može pamtiti stanja (prošlost), izgraditi model (opis) prostora i koristiti ga za generiranje učinkovitijih ponašanja, te planirati unaprijed.

Primjer ponašajne strukture upravljanja je „Motor shema“ Ronalda Arkina u kojoj više konkurentnih shema generira odvojene naredbe mobilnom robotu (brzina i smjer gibanja) koje se kombiniraju kao normirani zbroj, dajući stvarnu naredbu robotu.

Drugi primjer, i onaj kojime ćemo se mi baviti je Brooksova supsumpcijska (engl. *subsumption*) struktura. Kod nje su ponašanja organizirana po prioritetima. To je prva ponašajna struktura upravljanja mobilnim robotima i izazvala je veliki prekret u razvoju autonomnih mobilnih robota.

2.2 Brooksova supsumpcijska arhitektura

Konvencionalan pristup izgradnjih upravljačkih kontrolnih sustava za autonomne mobilne robote je dekompozicija problema u skup jedinica od kojih svaka obavlja funkciju poput percepcije svijeta, modeliranja, planiranja, obavljanja zadaća ili kontrole motora. Centralni kontrolni sustav pokreće svaku od ovih jednica i predaje podatke iz prethodne jedinice slijedećoj. Primjerice, percepcijska jedinica „snima“ svijet i predaje te podatke modelirajućoj jedinici koja od tih podataka oblikuje unutarnji model percipiranog svijeta. Taj unutarnji model se proslijeđuje planirajućoj jedinici koja iz dobivenih podataka, koristeći razne metode poput rezolucije ili unifikacije, generira plan. Rezultirajući plan se proslijeđuje jedinici za kontrolu motora, koja djeluje direktno na svijet. U takvoj strukturi je najčešće vrlo malo jedinica u izravnom kontaktu sa svijetom (u ovom primjeru su to percepcijska i jedinica za upravljanje motorom). Izlaz svake od jedinica je u čvrstoj vezi sa slijedećom jedinicom.

Alternativa gore opisanom konvencionalnom centralno upravljanom načinu izgradnje upravljačkih sustava autonomnih robota je dekompozicija problema u skup asinkronih ponašanja koja rješavaju zadatke (eng. *task achieving behaviours)*. U tom pristupu ukupna kontrola nad robotom je ostvarena kolektivnim radom asinkronih lokalnih interakcija između tih relativno primitivnih ponašanja, a koja sva komuniciraju sa svijetom i međusobno. Svako od tih ponašanja najčešće obavlja nekvu relativno jednostavnu funkciju, npr. tumaranje, prepoznavanje objekata, izbjegavanje prepreka i slično. Ta ponašanja su često primitivnija od funkcijskih jedinica iz konvencionalnog pristupa izgradnji upravljačkih sustava. Prednost takve arhitekture je što podržava višestruke ciljeve i efikasnija je od modelske. Takvi sustavi su jednostavniji za izvedbu, jednostavnije im se otklanjaju eventualne greške u dizajnu i vrlo se lako proširuju. Sustavi su robusni, otkazivanje nekog od slojeva ne onemogućuje robota u radu. Naravno, ovakve strukture imaju i svoje nedostatke, prvenstveno nemogućnost planiranja (čisti oblik ponašajnog sustava upravljanja mobilnim robotom nema memoriju, čak ni memoriju unutarnjeg stanja). Također, teško je predvidjeti rezultat interakcije velikog broja ponašanja.

Svaki stvarni robotičarski prolem mora se razložiti u nekoliko potproblema. Brooks je odabrao vertikalno razlaganje; problem je „podijeljen“ vertikalno na temelju željenih vanjskih manifestacija sustava upravljanja. Definirao je 7 razina, gdje na svakoj razini imamo neformalnu specifikaciju željene klase ponašanja mobilnog robota u bilo kojem okruženju. Više razine kompetencije podrazumijevaju specifičniju definiciju klase ponašanja koje rješava zadatke. Ključna ideja uvođenja razina kompetencije, čiji je primjer prikazan na slici 2.3, je da se razine sustava upravljanja mogu izvesti tako da odgovaraju razinama kompetencije, npr. sustav upravljanja može se izvesti tako da sadrži samo nultu razinu kompetencije (izbjegavanje prepreka). Na nju možemo dodati prvu razinu koja može pregledavati i provjeravati podatke od nulte razine, te može generirati izlazni signal koji sprječava izlaz iz nulte razine (nulta razina ne zna da je njen rad spriječen i nastavlja sa radom).

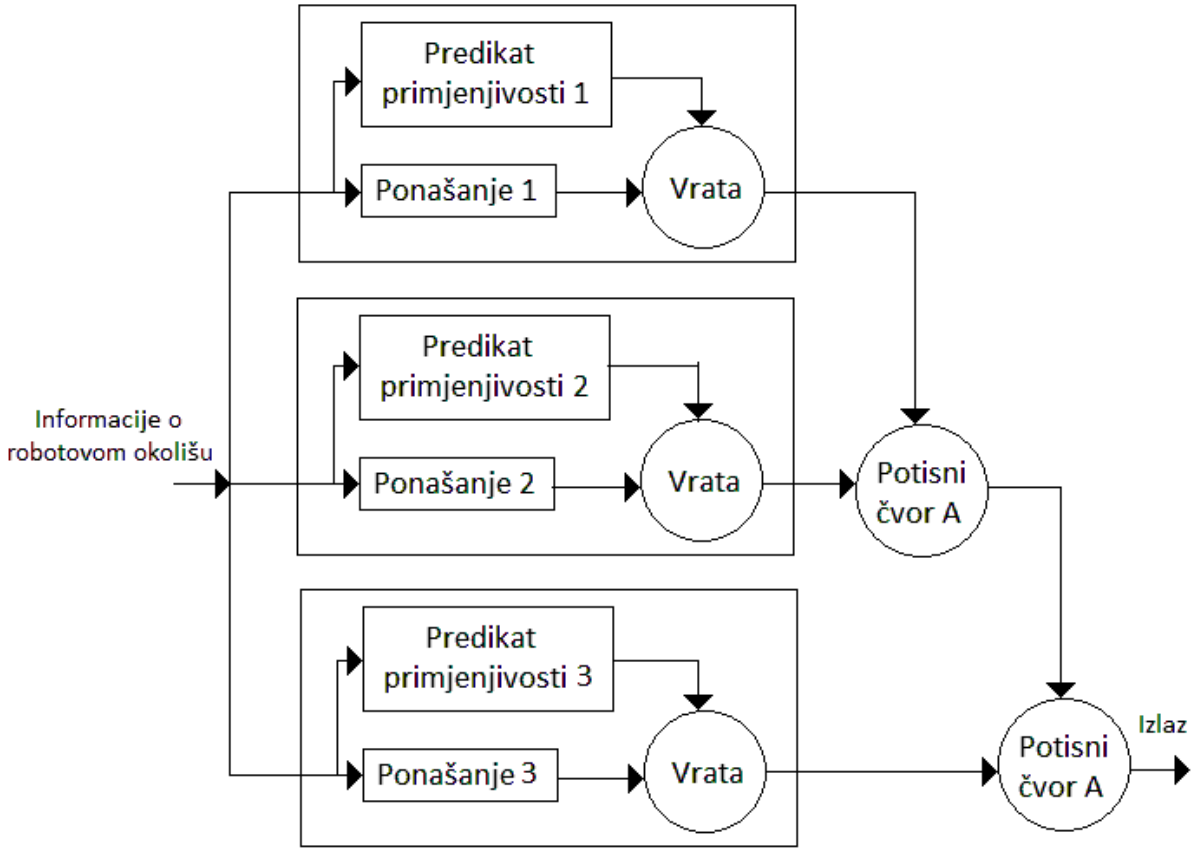
7	zaključuj o vladanju objekata u prostoru i na odgovarajući način modificiraj planove
6	oblikuj i izvodi planove koji uključuju promjene stanja u prostoru, na neki način
5	identificiraj standardne objekte u prostoru i obavljaj radnje koje se odnose na određene tipove objekata
4	uočavaj promjene u statičkom prostoru
3	izgrađuj kartu prostora i planiraj putanju od jednog do drugog mjesta
2	istražuj prostor uočavanjem udaljenih mjesta koja izgledaju dohvatljiva te se gibaj prema njima
1	"tumaraj" besciljno bez udaranja u predmete
0	izbjegavaj kontakt sa stacionarnim/pokretnim predmetima

Slika 2.3 Primjer razina kompetencije

U supsumpcijskoj arhitekturi, prikazanoj slikom 2.4, razni podskupovi ponašanja pokazuju nekakvu djelomičnu kompetenciju u rješavanju jednostavnijeg dijela ukupnog problema. Rješenje problema može biti izgrađeno inkrementalno dodajući novo nezavisno ponašanje već postojećim ponašanjima. Dodatak svakog novog ponašanja dodaje novu funkcionalnost sustavu. Istovremeno, sustav je otporan na greške jer otkazivanje jednog od ponašanja ne uzrokuje prestanak rada, već samo malu degradaciju performansi.

Slika 2.4 Primjer supsumpcijske arhitekture

Slika 2.5 prikazuje većinu glavnih karakteristika supsumpcijske arhitekture. Tri ponašanja koja rješavaju zadatke su prikazana u velikim pravokutnicima. Unutar svakoga tog ponašanja se nalazi predikat primjenjivosti, vrata te ponašanje, odnosno aktivnost koju ponašanje obavlja. Sva tri ponašanja su u izravnoj vezi sa robotovom okolinom. Ako ta okolina zadovoljava neko od ponašanja, vrata će postaviti aktivnost koje ponašanje obavlja na izlaznu liniju tog ponašanja. Potisni čvorovi rješavaju konflikte i stvaraju finalni izlaz.



Slika 2.5 Supsumpcijska arhitektura sa tri ponašanja koja rješavaju zadatke

Budući da ponašanja koja rješavaju zadatke rade nezavisno potrebno je razriješiti moguće konflikte među njima. Na slici 2.5 vidi se hijerarhijska struktura potisnih čvorova korištenih za razrješavanje konflikata. Izlaz svakog od ponašanja može biti predočen kao paket instrukcija za kontrolu robota. Paket koji ulazi u čvor s vrha ima prednost nad paketima koji dolaza s lijeva. Konkretno, ako postoji bilo kakav izlaz iz ponašanja najvišeg prioriteta, ono potiskuje izlaz (ako on postoji) iz ostalih, niže postavljenih, ponašanja. Izlaz iz trećeg ponašanja može upravljati robotom samo ako više postavljena ponašanja ne emitiraju izlaz. Takvo hijerarhijsko oblikovanje potisnih čvorova stvara prioritete među ponašanjima, gdje je najviše postavljeno ponašanje najvišeg prioriteta.

Predikat primjenjivosti svakog ponašanja u supsumpcijskoj arhitekturi sadrži kompoziciju uvjetnih logičkih funkcija i vanjskih senzora okoline. Dio sa aktivnošću tipično sadrži kompoziciju funkcija koje izvršavaju nekakvu akciju.

Hijerarhijski raspored potisnih čvorova koji djeluju nad emitiranim akcijama ponašanja zapravo je nekakav skup logičkih funkcija koje vraćaju ili NIL ili emitiranu akciju.

Predikati primjenjivosti i potisni čvorovi sa slike 2.5 mogu se reformulirati sa par IF – ELSE IF uvjetnih funkcija.

```
IF         P-P-1 PONAŠANJE1
ELSE IF    P-P-2 PONAŠANJE2
ELSE IF    P-P-2 PONAŠANJE3
```

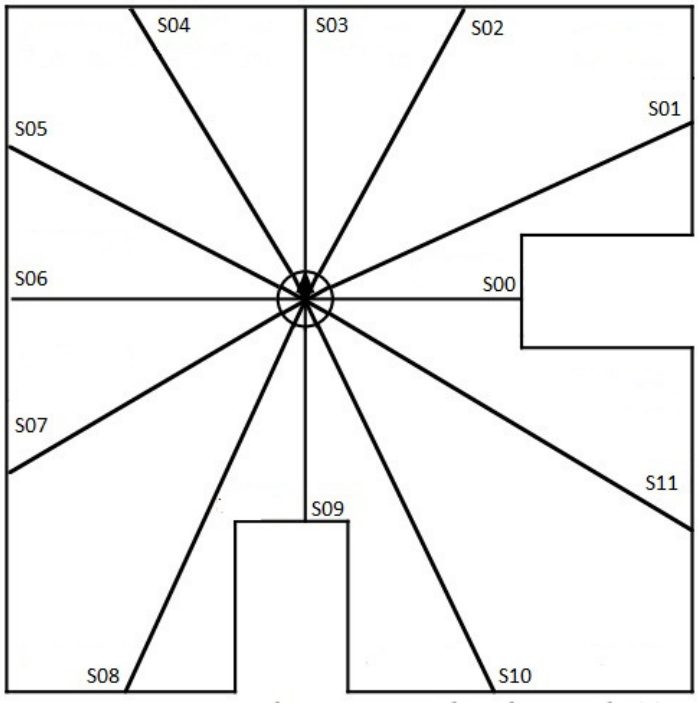
Ako je `P-P-1`, prvi predikat primjenjivosti, zadovoljen izvršava se aktivnost pripadnog ponašanja, `PONAŠANJE1`. Inače, ako je `P-P-2` zadovoljen izvršava se `PONAŠANJE2`. U suprotnom, izvršava se `PONAŠANJE3`.

Ovom reformulacijom postaje očito da je učinak predikata primjenjivosti i potisnih čvorova u supsumpcijskoj arhitekturi ništa više nego kompozicija jednostavnih IF – ELSE IF funkcija.

Potrebno je poprilično programersko znanje i sposobnost da bi se ostvario i napisao prikladni skup ponašanja koja rješvajaju zadatke, a koja su sposobna rješiti problem u stilu supsumpcijske arhitekture. Postavlja se pitanje je li moguće evoluirati takvu arhitekturu. Ta bi evolucija uključivala potragu za odgovarajućim ponašanjima, uključujući primjerene predikate primjenjivosti i ponašajne aktivnosti, te prikladnu hijerarhiju za razrješavanje konflikata.

2.3 Problem praćenja zidova

Mataric (Mataric 1990) je opisala problem upravljanja mobilnim autonomnim robotom na način da on obavlja zadaću obilaska svih rubnih površina neke nepravilne sobe.



Slika 2.6 Primjer sobe, te prikaz robotovih senzora

Opisani robot je sposoban izvršavati 5 primitivnih motornih funkcija: kretanje unaprijed za zadanu duljinu, kretanje unazad za zadanu duljinu, skretanje ulijevo za 30 stupnjeva i skretanje udesno za 30 stupnjeva, te zaustavljanje.

Robot ima 12 sonarnih senzora, a svaki od njih pokazuje udaljenost do najbližeg zida. Raspoređeni su po robotu sa razmakom od 30 stupnjeva, tako da, svi zajedno, pokrivaju svih 360 stupnjeva, kao što se vidi na slici 2.6. Također, robot u svakom trenutku može saznati i minimalnu vrijednost tih dvanaest udaljenosti, što se može smatrati trinaestim senzorom. Robot se nalazi na poziciji (12, 16)[1], a duljine gornjeg i lijevog zida su 27,6 metara.

Ovaj problem se može riješiti na više načina, uključujući konvencionalni pristup, ručno građenu supsumpcijsku arhitekturu ili genetskim programiranjem izgrađenu *subsumption* arhitekturu. Neovisno koja će metoda biti upotrebljena, 13 senzora se može gledati kao ulazne varijable u još nenapisanim programima. Ti će programi obrađivati ulaze i uzrokovati aktivnost nekih od robotskih primitivnih motornih funkcija. Svi će programi biti kompozicija 5 motornih funkcija i 13 dostupnih senzora. Bitno je primijetiti da su tih 5 funkcija i 13 senzora polazne točke u sva tri pristupa rješavanju.

Mataric je problem riješila supsumpcijskom arhitekturom. Za 4 ponašanja napisana su 4 računalna programa koji su omogućili robotu da obilazi rubne površine. Uz vrijednosti senzora i primitivne motorne funkcije, korištene su i konstante EDG, kao preferirana udaljenost robota od zida (ujedno i duljina stranice kvadratne rubne površine), te MSD, minimalna sigurna udaljenost od zida, te DZ, opasna zona (za EDG je korištena vrijednost 2,3, za MSD 2,0, a za DZ 1,0). Četiri napisana programa nazvana su STROLL (LUTAJ), AVOID (IZBJEGAVAJ ZIDOVE), ALIGN (PORAVNAJ SE) i CORRECT (ISPRAVI SE). Svako od tih ponašanja koje rješava zadatke komunicira izravno s okolinom. Također, prestanak rada jednog od ponašanja neće spriječiti robot u radu. Primjerice, robot može obilaziti sobu bez sudara samo sa STROLL i AVOID ponašanjima.. ALIGN dodaje mogućnost obilaska konveksnih granica, a CORRECT dodaje omogućnost obilaska rubnih područja..

Četiri napisana programa sadrže ukupno 25 različitih varijabli i 14 različitih funkcija. 25 varijabli su 13 vrijednosti senzora, tri konstantna parametra i 9 dodatno definiranih varijabli, dinamički izračunatih minimuma vrijednosti raznih podskupova senzora. Četrnaest upotrijebljenih funkcija su pet primitivnih motornih funkcija, te 9 dodatnih LISP funkcija (`IF, AND, NOT, COND, >, >=, =, <=` i `<`).

Realizirani program, implementacija supsumpcijske arhitekture, sastoji se od 151 elementa, što funkcija, što varijabli, te je uz pomoć njega robot obišao 56 rubnih ćelija u 203 vremenske jedinice. Činjenica da je Mataric uspjela osmisliti i napisati njena 4 programa je značajan dokaz za jednu od glavni postavki supsumpcijske arhitekture, a ta je da je moguće izgraditi upravljački sustav za autonomne mobilne robote koristeći slabo povezana asinkrona ponašanja koja rješavaju zadatke, što samo po sebi ipak nije očito. Njena su četiri programa veliki odmak od klasičnog oblikovanja upravljačkih sustava.

[1] Ishodište se nalazi u donjem lijevom kutu prostorije

3 Rješavanje problema praćenja zidova genetskim programiranjem

Jedna od mogućnosti rješenja problema obilaska rubnih površina je i genetsko programiranje. Za rješavanje koristit će se isti terminali i iste primitivne funkcije kao i one koje su korištene za ručnu izgradnju supsumpcijske arhitekture.

Algoritmi za učenje općenito zahtijevaju ponavljanje eksperimenta i isprobavanje kako bi sakupili dovoljno informacija da algoritam riješi problem. U ovom problemu učenje zahtjeva da robot izvršava zadanu mu funkciju više puta kako bi skupio dovoljno informacija. Simulacija robota pruža praktičan izvor potrebnih informacija. Dakle, simulira se rad robota, umjesto rada na stvarnom robotu.

Budući da simulirani robot ne mora nikada stati i ne može se sudariti sa zidovima, možemo izbaciti STOP funkciju i DZ parametar. U eksperimentu robotu uopće nije dozvoljeno kretati se ako će tom akcijom doći u kontakt sa zidom.

Koristit će se svih 12 robotovih senzora, a dodaje se i SS, dinamički izračunatu minimalnu vrijednost svih 12 senzorskih vrijednosti. Od konstanti korisiti se samo EDG i MSD.

Koristit se 4 motorne funkcije :

- MF – *move forward* – pomak unaprijed za 1 metar
- MB – *move backwad* – pomak unazad za 1,3 metra
- TL – *turn left* – okret ulijevo za 30 stupnjeva
- TR – *turn right* – okret udesno za 30 stupnjeva

Da bi osigurali da robot ne dođe u kontakt sa zidom, prije nego što se izvrše MF ili MB naredbe, robot provjerava je li na 110% udaljenosti koju robot namjerava prevaliti prepreka, te ako jest, odustaje od aktivnosti.

Svaka od gore opisanih primitivni aktivnosti traje jednu vremensku jednicu. Stanje robota je određeno njegovim horizontalnim i vertikalnim položajem, te usmjerenošću. Te vrijednosti su kontinuirane (robot nije na diskretnoj mreži). Izvršavanjem bilo koje od tih aktivnosti dolazi do promjene stanja robota. Dinamičke vrijednosti sonara ne računaju se unaprijed (iako bi se time donekle ubrzalo izvođenje), već se izračunavaju na zahtjev.

Time je skup završnih znakova, terminala, kompletiran, i izgleda ovako:

```
T = {S00, S01, S02, S03, S04, S05, S06, S07, S08, S09, S10,
S11, SS, MSD, EDG, MF, MB, TR, TL}
```

Četiri primitivne motorne funkcije MF, MB, TR, TL su mogle biti stavljene i u skup funkcija, no budući da ne primaju nikakve argumente, stavljene su u skup završnih znakova.

Programerima je jednostavno i praktično imati na raspolaganju čitav niz funkcija (=, >=, <=, IF, ELSE...) za pisanje programa. U genetskom programiranju često je poželjno, iako ne i nužno, pojednostaviti skup dostupnih funkcija. Kao što je već spomenuto, supsumpcijska arhitektura koja se sastoji od više ponašanja koja rješavaju zadatke, i svako od njih sa svojim predikatom primjenjivosti i mrežom potisnih čvorova, može se zamjeniti nizom običnih IF-ELSE uvjetnih funkcija. Sve numeričke komparacije (<, >, =, >=, <=) mogu se realizirati koristeći samo jednu komparaciju, manje ili jednako (<=) . Iz tog razloga će se u skup uvjetnih funkcija staviti samo jedna uvjetnu funkciju, IFLTE (eng. *if less than or equal*, ako je manje ili jednako). Ta funkcija prima četiri argumenta, ako je vrijednost prvoga manja ili jednaka od drugoga, onda evaluira i vraća vrijednost trećeg argumenta, a inače evaluira i vraća vrijednost četvrtog argumenta. Ovom funkcijom se omogućava realizacija i predikata primjenjivsti i mreže potisnih čvorova pripadnih ponašanja.

Uz IFLTE, koristi se i spojnu funkciju PROGN2, koja prima dva argumenta, slijedno evaluira prvi pa drugi, te vraća vrijednost drugoga.

Dakle, skup funkcija izgleda ovako:

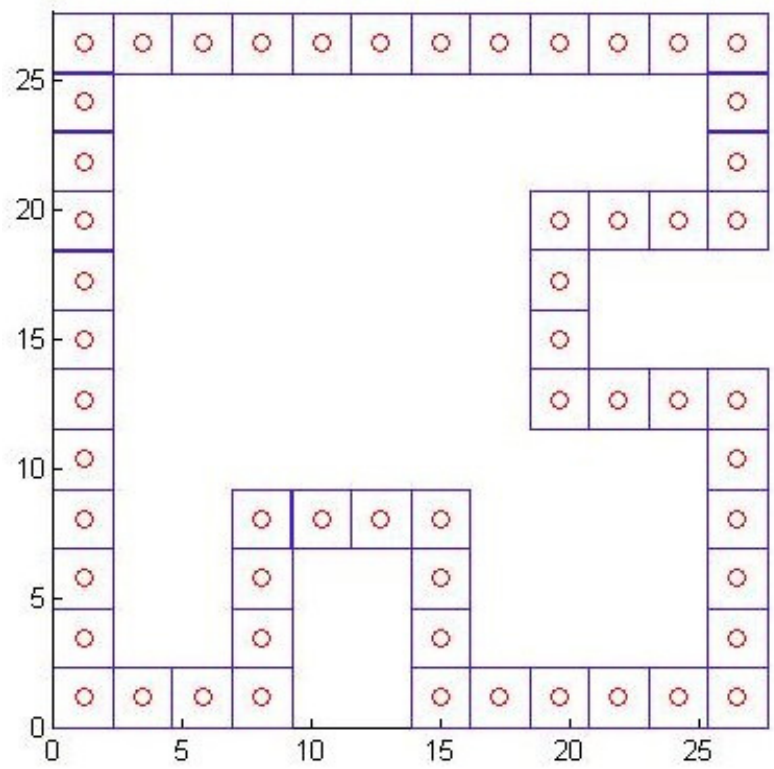
```
F = {IFLTE, PROGN2}
```

Odabranim skupom terminala i funkcija potpunost je zadovoljena jer robot dobiva dovoljno informacija o svijetu oko sebe, a sa četiri dostupne mu motorne funkcije ima potpunu slobodu kretanja, što osigurava da evoluirani upravljački programi mogu imati dovoljnu kvalitetu da riješe problem praćenja zidova. Zatvorenost nije sama po sebi ispunjena, upravo zbog motornih funkcija. One moraju vraćati nekakve numeričke vrijednosti. Korištene funkcije IFLTE i PROGN2 primaju realne vrijednosti, te će se definirati da sve četiri motorne aktivnosti vraćaju minimalnu vrijednost tri senzora robota koji gledaju prema naprijed (senzor koji gleda u smjeru kretanja robota, te njemu susjedni senzori).

Slijedeći korak je određivanje funkcije dobrote. Kada čovjek piše programe, ljudska inteligencija je nit vodilja koja kreira programe. Kod genetskog programiranja, ta nit je pritisak vršen od mjere dobrote i prirodne selekcije.

Cilj simuliranog robota je obići sve rubne površine nepravilno oblikovane prostorije. Rubne površine su one tik do zidova i unaprijed zadane širine. Te će povšine biti podijeljene u kvadratne ćelije duljine stranice 2,3 metra. Na slici 3.1 je primjer prostorije sa ukupno 56 rubnih ćelija (kvadrati sa krugom u sredini), te bi u takvoj prostoriji maksimalna dobrota iznosila 56. Početni položaj robota će iznositi (12, 16), a orijentacija će mu biti 270 stupnjeva. Cilj robota je obići što više tih rubnih ćelija u unaprijed zadanoj količini vremena.

Ovakav opis dobrote je dovoljan za ispravan rad sustava, no njime se neće razlikovati dvije jedinke koje obiđu jednak broj rubnih ćelija u različitom vremenu. Zato se uz broj obiđenih ćelija zapisuje i uspoređuje vrijeme potrebno za taj obilazak. Boljom jedinkom će se, logično, smatrati ona sa manjim potrebnim vremenom za obilazak.



Slika 3.1 Primjer prostorije sa označenim rubnim površinama

U prvim generacijama izvođenja unutar populacije će biti mnogo nekvalitetnih jedinki (dobrota 0), npr. one u koje bi robot kružio oko neke lokacije ili samo stajao na mjestu. Očito je da bi izvršavanje takve simulacije trajalo vječno, pa je, kao i kod svih simulacija, potrebno odrediti i ograničiti vrijeme izvođenja.

Prostorija sadrži 56 rubnih ćelija dugih 2,3 metra. Kada bi se robot kretao unaprijed (1 metar po vremenskoj jedinici), trebalo bi mu 129 vremenskih jedinica za obilazak svih ćelija. Kada bi se robot kretao unatrag (1,3 metra po vremenskoj jedinici) obišao bi sve ćelije za 99 vremenskih jedinica. U prostoriji se nalazi i dvanaest pravih kuteva. Kako robot skreće brzinom od 30 stupnjeva po vremenskoj jedinici, lako je izračunati da mu je za sva skretanja u prostoriji dovoljno 36 vremenskih jedinica. Ako tome dodamo vremenske jedinice potrebne da robot od početne pozicije do zida, dobija se optimalno vrijeme obilaska prostorije od oko 150 do 180 vremenskih jedinica. Iz istog razloga simulacija će završiti izvršavanje nakon 400 vremenskih jedinica. Preporučljivo je ne izabirati mnogo manju brojku, jer će većina inicijalnih jedinki, koje su po prirodi posve nasumične, imati iznimno nisku i vjerojatnu jednaku vrijednost dobrote, što otežava i ograničava izvođenje i napredak.

Kao kriterij zaustavljanja moguće je odabrati postizanje maksimalne dobrote od 56 obiđenih rubnih ćelija, no izvjesno je da će tada te jedinke biti relativno neučinkovite, te će njihovo obilaženje sobe biti više produkt nasumičnog gibanja nego ciljanog obilaženja zidova. Iz tog razloga, izvođenje će se nastaviti i nakon što što jedna ili više jedinki postigne maksimalnu dobrotu, a te jedinke će se međusobno razlikovati po vremenu potrebnom za obilazak svih rubnih ćelija. Time se nastoji usavršiti ponašanje jedinki, što će za posljedicu imati obilazak cijele prostorije u kraćem vremena. Kriterij zaustavljanja će biti dostignuta 50. generacija.

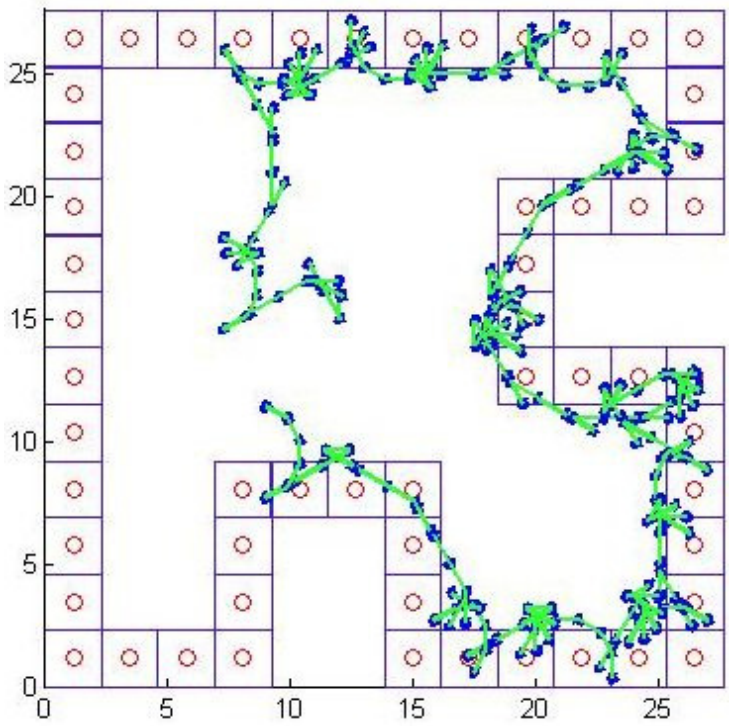
3.1 Primjer izvođenja

Tablica 3.1 prikazuje korištene parametre. Koristiti ćemo sobu sa slike 3.1.

Tablica 3.1 Korišteni parametri

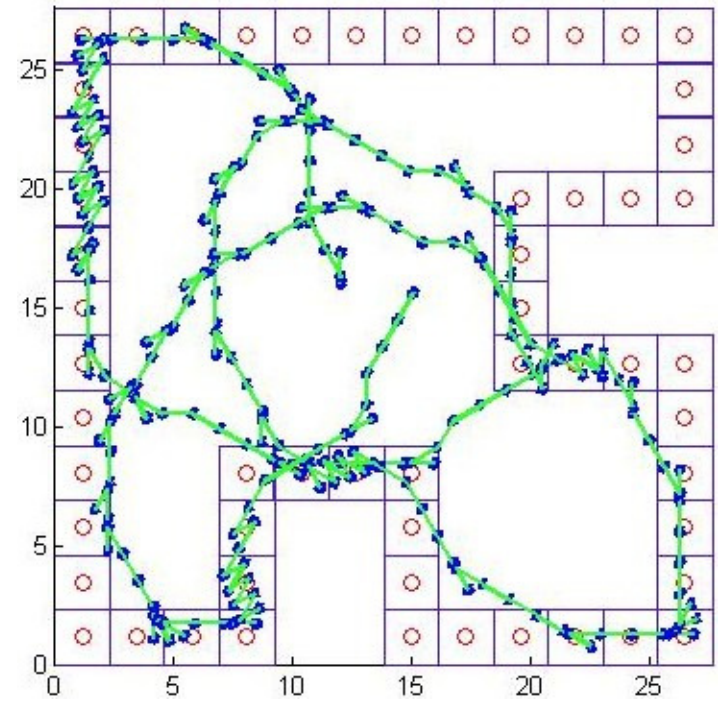
Cilj	Pronaći računalni program koji će navoditi autonomnog mobilnog robota uz rubne površine nepravilne sobe
Skup završnih znakova	Udaljenosti od robota do zida u 12 smjerova, minimalna udaljenost od tih 12, konstante EDG i MSD, te kretanje unaprijed i unatrag i okretanje u lijevo i u desno
Skup operatora	IF LESS THAN OR EQUAL (IFLTE) i PROGN2
Uzorci za učenje	Jedna prostorija
Mjera dobrote	Broj obiđenih rubnih površina (od mogućih 56), te vrijeme potrebno za to
Parametri	Populacija = 1000, broj generacija = 50
Uvjet zaustavljanja	50 generacija
Selekcija	Troturnirska selekcija
Primjena genetskih operatora	1% mutacija, 10% reprodukcija, 89% križanje

U prvoj generaciji čak 506 jedinki ima dobrotu nula, što znači da ne posjete niti jedno rubno polje. Prosječna dobrota je 1,43, a najbolja jedinka obilazi ukupno 31 rubnu ćeliju u 401 vremenskoj jedinici. Slika 3.2 prikazuje putanju te jedinke.



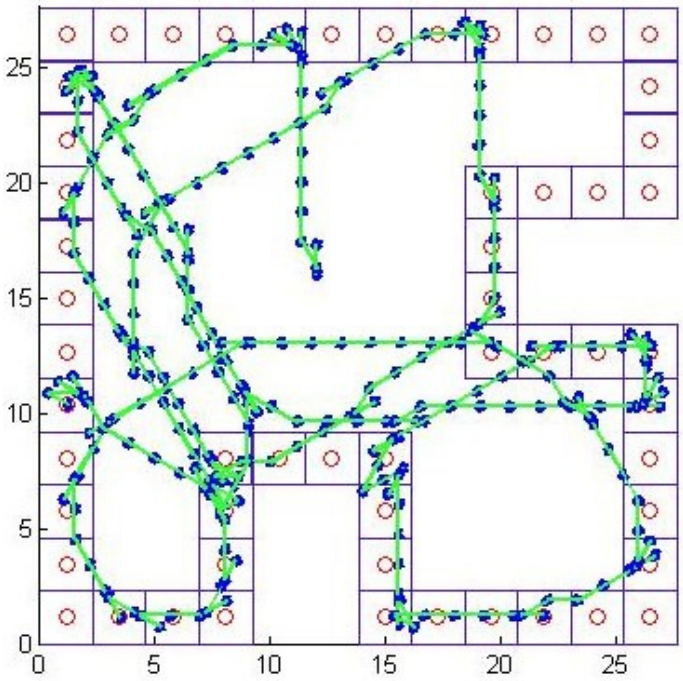
Slika 3.2 Putanja robota u 1. generaciji

U osmoj generaciji najbolja jedinka obilazi 36 ćelija u 407 vremenskih jedinica. Prosječna dobrota te generacije je 11,94. Putanja robota prikazana je slikom 3.3.



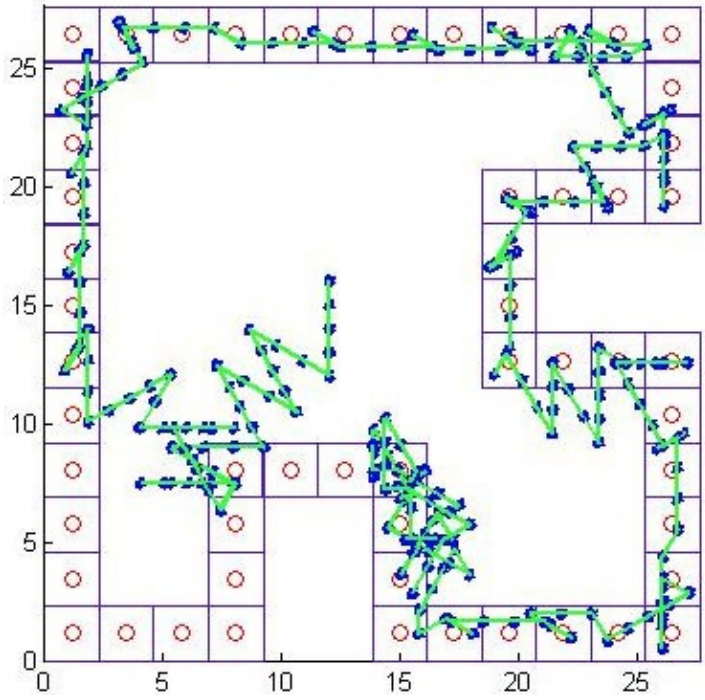
Slika 3.3 Putanja robota u 8. generaciji

U 11. generaciji dobrota najbolje jedinke iznosi 41, a prosječna dobrota je 16,53. Putanju robota prikazuje slika 3.4.



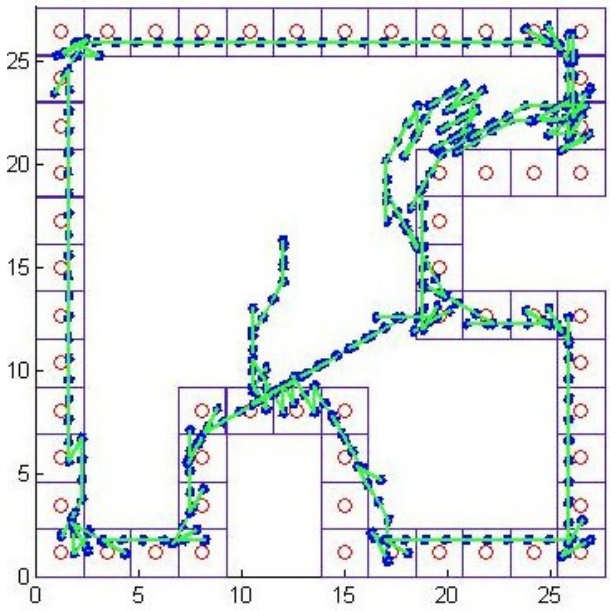
Slika 3.4 Putanja robota u 11. generaciji

Najbolja jedinka 14. generacije, prikazana slikom 3.5, obilazi ukupno 47 rubnih ćelija. Prosječna dobrota iznosi 20,4. Iako je kretanje robota i dalje poriličnо nasumično, vide se blage naznake ponašanja koje obavlja zadatke, posebice onog praćenja zidova.



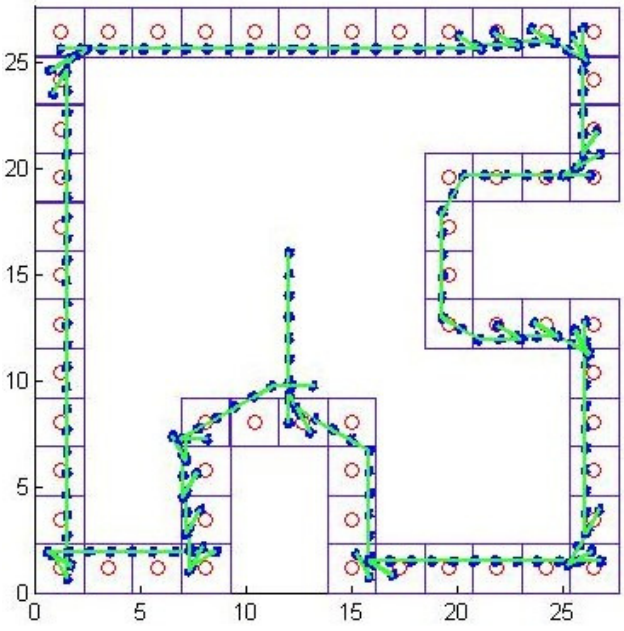
Slika 3.5 Putanja robota u 14. generaciji

Generacija 29 producira upravljački program koji obilazi 53 od maksimalnih 56 rubnih ćelija. Robot ima problema sa poravnavanjem i praćenjem zidova, no nakon što je poravnat bez greške obilazi površine uz ravne zidove. Prosječna dobrota ove generacije iznosi 30,45. Putanja je prikazana slikom 3.6.



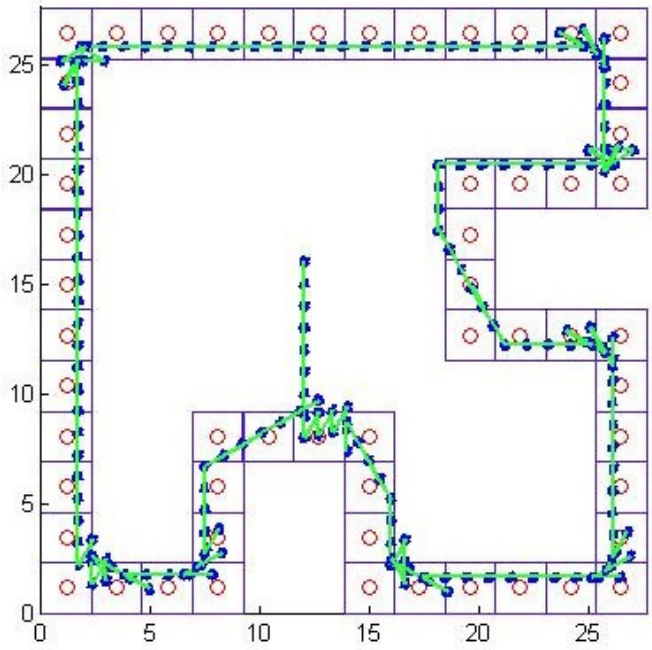
Slika 3.6 Putanja robota u 29. generaciji

Generacija 32 konačno stvara upravljački program koji navodi robota kroz svih 56 rubnih polja, što je prikazano slikom 3.7, i to u samo 250 vremenskih jedinica. Ta jedinka je jedina sa maksimalnom dobrotom u toj generaciji. Robot uspješno pronalazi rubna polja, poravnava se i prati zidove, te skreće uz rubne površine. Prosječna dobrota ove generacije iznosi 33,61.



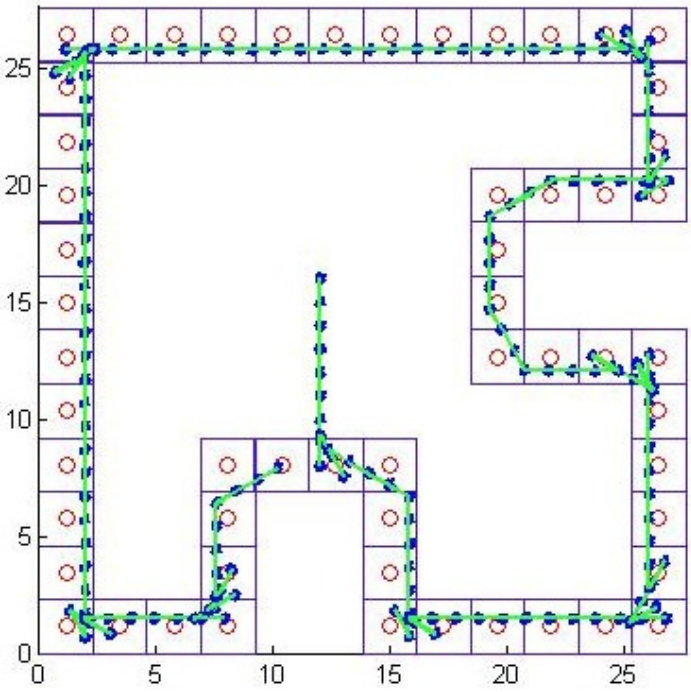
Slika 3.7 Putanja robota u 32. generaciji

Već u slijedećoj generaciji evoluira bolja jedinku, čiju putanju prikazuje slika 3.8. Ovaj put upravljački program provede robota kroz sve rubne ćelije za 5 vremenskih jedinica manje nego u prethodnoj generaciji. Unutar populacije su već 4 jedinke sa maksimalnom dobrotom, a prosječna dobrota iznosi 35,88.



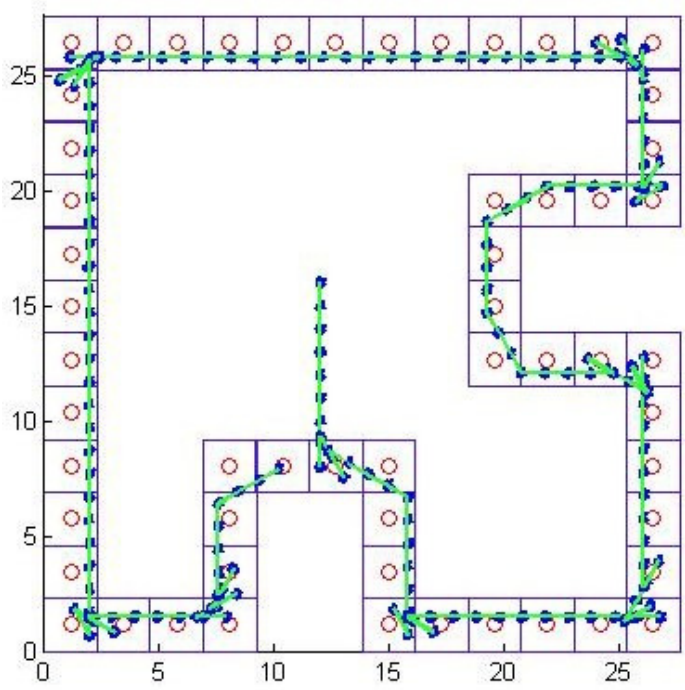
Slika 3.8 Putanja robota u 33. generaciji

Slika 3.9 prikazuje putanju robota iz 39. generacije. On obilazi svih 56 rubnih polja za 214 vremenskih jedinica. Prosječna dobrota generacije iznosi 47,77.



Slika 3.9 Putanja robota u 39. generaciji

U 47. generaciji se stvara ukupno najbolja jedinka, čija je putanja prikazana slikom 3.10. Ona svih 56 rubnih polja obilazi za samo 213 vremenskih jedinica. Može se primijetiti gotovo savršeno gibanje robota, uz tek minimalne suvišne pokrete potrebne za poravnavanje uz zidove nakon skretanja.



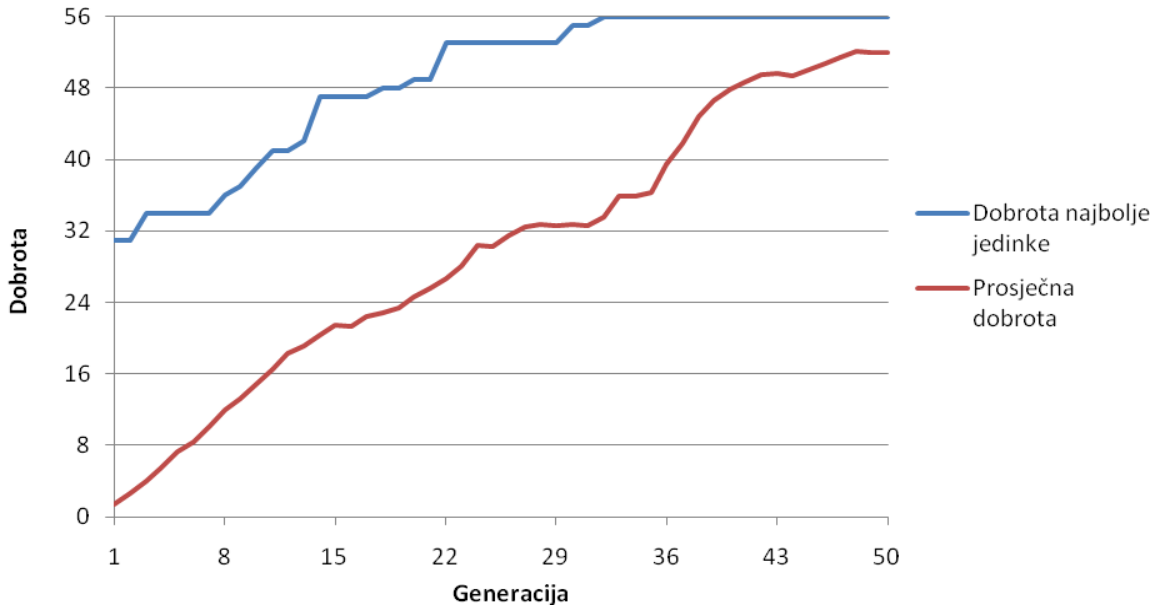
Slika 3.10 Putanja robota u 47. generaciji

U zadnjoj generaciji najbolja jedinka je i dalje ona iz 47 generacije. Čak 799 jedinki ima dobrotu 56. Prosječna dobrota iznosi 51,9.

Najbolja jedinka se sastoji od 172 elementa, što funkcija, što završnih znakova (za usporedbu, upravljački program od Mataric je sadržavao 203 elementa) i prikazuje ju slijedeći program:

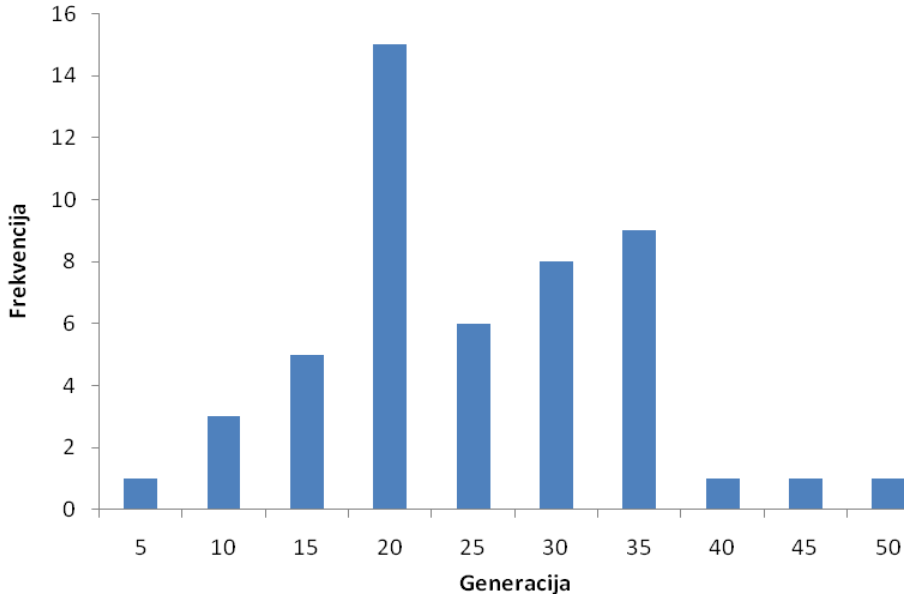
```
PROGN2 IFLTE IFLTE S06 MF S10 S11 IFLTE MF MF S05 S05 S10
IFLTE TL S09 IFLTE S05 S06 MF SS S03 IFLTE MF PROGN2 S11 MSD
PROGN2 MB TL IFLTE S01 S11 PROGN2 SS IFLTE SS MF SS IFLTE S03
IFLTE MF S09 S05 PROGN2 IFLTE S11 TR SS S02 S08 PROGN2 S02
IFLTE S02 EDG S10 SS IFLTE PROGN2 PROGN2 IFLTE MF TL S10 S00
PROGN2 MF MF S11 IFLTE S08 TL MSD S04 PROGN2 SS MSD S07 IFLTE
S09 PROGN2 S06 IFLTE IFLTE S06 IFLTE S01 S11 EDG PROGN2 IFLTE
IFLTE IFLTE S06 S07 TR MF PROGN2 S09 MSD PROGN2 IFLTE S11 S09
S03 MB TL IFLTE S01 S11 S04 IFLTE S09 PROGN2 S09 S11 IFLTE
S09 S01 TR TL S06 TR TR IFLTE S06 MF MF S07 IFLTE SS MF SS
PROGN2 S00 TL TR IFLTE MF S02 S05 S05 PROGN2 S06 MSD PROGN2
MB TL IFLTE S01 S11 S03 IFLTE S09 PROGN2 S09 S03 IFLTE S09
S01 TR S07 S09 IFLTE S01 S01 TR TL S08
```

Slika 3.11 prikazuje progresivni napredak prosječne te najbolje dobrote populacije kroz generacije. Može se primijetiti konstantan raste kvalitete populacije. Na grafu nije ucrtana najgora dobrota svake generacije, jer smo u svakoj generaciji imali jedinku dobrote 0 (zapravo najgoru dobrotu predstavlja apscisa).



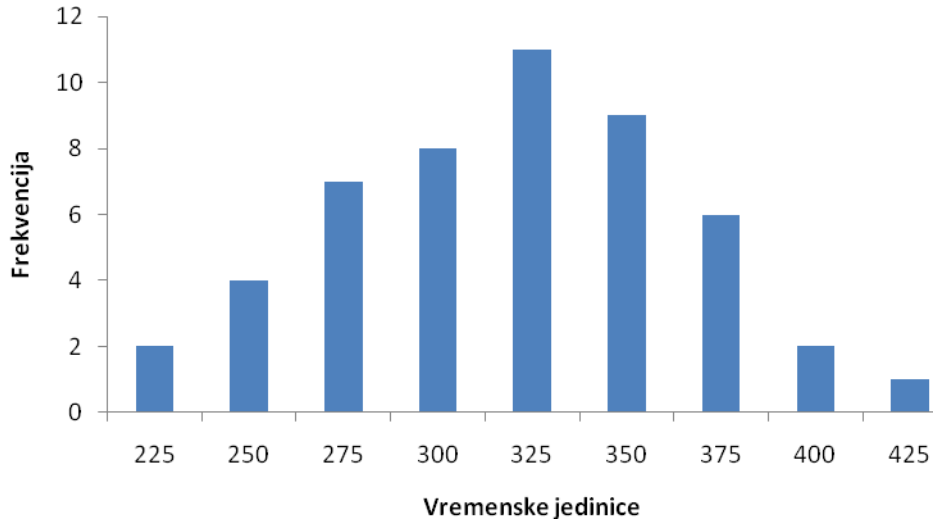
Slika 3.11 Promjena prosječne dorote i dobote najbolje jedinke kroz generacije

Slika 3.12 prikazuje trenutke u kojima su jedinke postigle maksimalnu dobrotu. Graf je napravljen na temelju 100 pokretanja genetskog programa. U prosjeku, taj tren se dogodi u 23. generaciji. U svih 100 pokretanja smo pronašli jedinku maksimalne dobrote unutar 50 generacija.



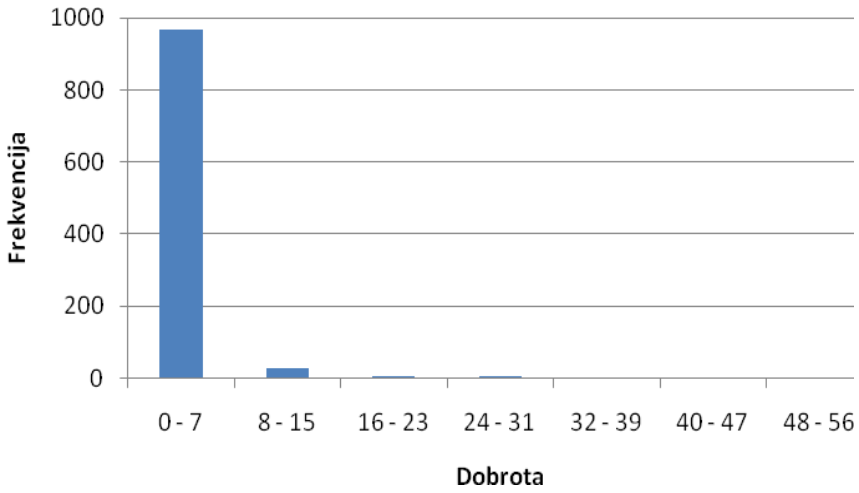
Slika 3.12 Generacije u kojima je postignuta maksimalna vrijednost dobrote

Slika 3.13 prikazuje vremena potrebna za obilazak svih 56 ćelija najboljih jedinki iz 100 pokretanja genetskog programa. Svih 100 jedinki je bilo maksimalne dobrote i u prosjeku su obišle cijelu sobu za 306 vremenskih jedinica.

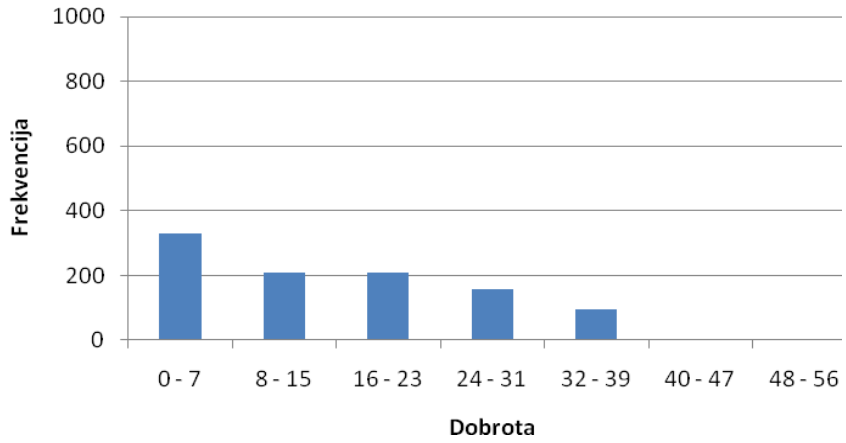


Slika 3.13 Vremena potrebna za obilazak rubnih ćelija najboljih jedinki

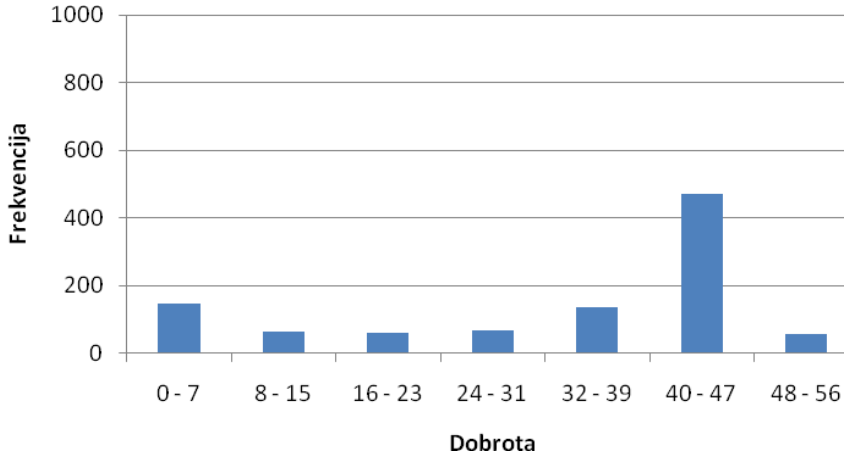
Slika 3.14, slika 3.15, slika 3.16 i slika 3.17 prikazuje raspodjelu svih 1000 jedinki iz 1., 10., 30., i 50. generacije po intervalima dobrote. Primjećuje se napredak kvalitete populacije. U prvoj generaciji preko 900 jedinki ima dobrotu nula. Kroz generacije se kvaliteta popravlja, a u zadnjoj, 50., generaciji čak 900 jedinki obilazi sve rubne ćelije sobe.



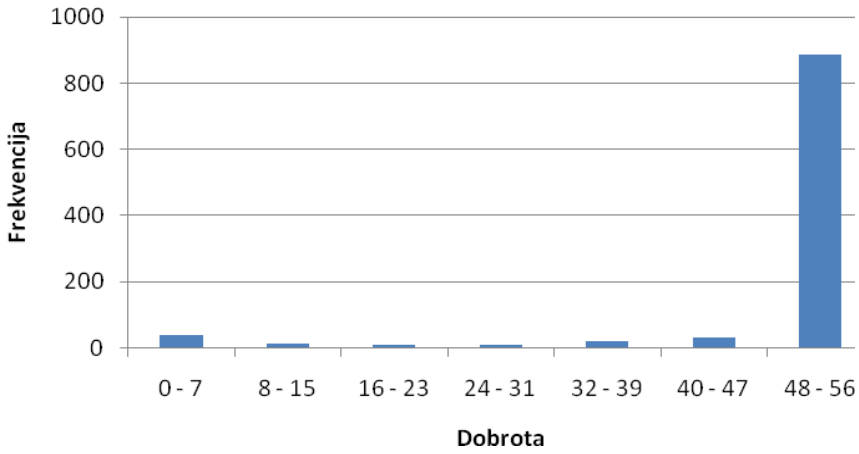
Slika 3.14 Raspodjela jedinki po dobrotu u 1. generaciji



Slika 3.15 Raspodjela jedinki po dobroti u 10. generaciji



Slika 3.16 Raspodjela jedinki po dobroti u 30. generaciji



Slika 3.17 Raspodjela jedinki po dobroti u 50. generaciji

3.2 Utjecaj vremenskog ograničenja simulacije na kvalitetu populacije i rezultata

Prethodno su već spomenuta dva parametra koji ubrzavaju izvođenje simulacije. Jedan je prekidanje izvođenja ako robot nije promijenio lokaciju nakon izvođenja cijelog upravljačkog programa, a drugi je ograničavanje vremena trajanja simulacije.

Vremensko ograničenje, osim ubrzanja izvođenja, ima utjecaj na kvalitetu populacije i rezultata. Svaka prostorija koja se koristi kao primjer za učenje ima svoj minimalan broj vremenskih jedinica potrebnih da simulirani robot obiđe sve rubne ćelije. Ako se vremensko ograničenje postavi jako nisko, oko te minimalne brojke, početna populacija, sa nasumičnim jedinkama, bit će nekvalitetna jer će iznimno malo jedinki stići postići bolji rezultat, te se većom dobrotom diferencirati od ostalih jedinki. Zato je bitno ne postaviti iznimno strogo vremensko ograničenje.

S druge strane, niže vrijednosti vremenskog ograničenja radit će veći pritisak na evolucijski razvoj jedinki, te će postojati veća mogućnosti za razvojem iznimno kvalitetne jedinke. Pokazat će se da će jedinke sa maksimalnom dobrotom obilaziti sobu u vremenu koje je relativno blisko vremenskom ograničenju, te će se rijetko kvalitatativno dalje razvijati. Iz tih razloga je vrlo bitno pravilno odrediti trajanje početnog vremenskog ograničenja.

U ovom eksperimentu koristi se prostorija sa slike 3.1 i parametri iz tablice 3.1. Minimalno trajanje obilaska te prostorije je oko 150 vremenskih jedinica (izračunato u poglavlju 3). Uz već prikazan primjer s ograničenjem od 400 jedinica, u nastavku je prikazano izvođenje s ograničenjem od 200, 300 i 500 jedinica.

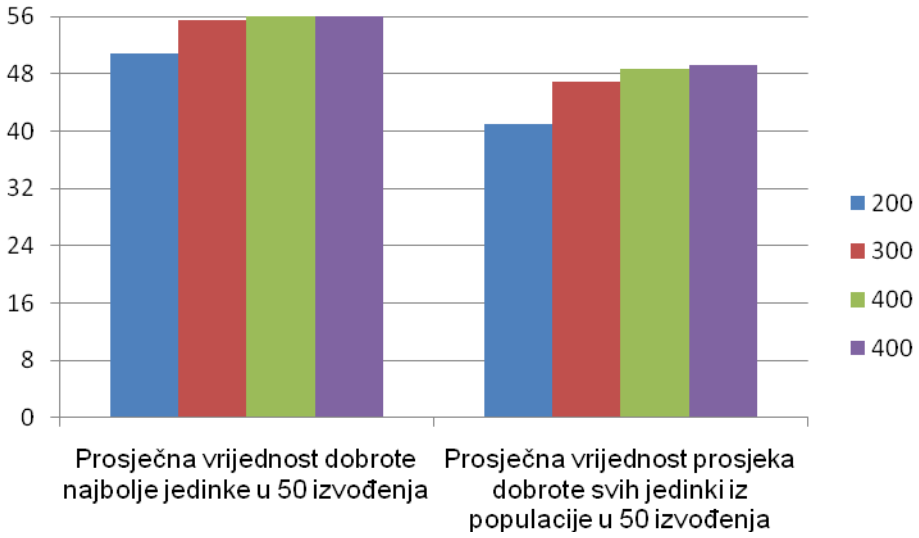
Ako se ograničenje postavi na 200 vremenskih jedinica, od 50 izvođenja samo će 5 puta biti pronađena jedinka maksimalne dobrote od 56 obiđenih rubnih ćelija (uspješnost izvođenja je 10%). Izvjesno je da će tih 5 jedinki biti iznimne kvalitete, jer će obići 56 ćelija u manje od 200 vremenskih jedinica. Prosječno će najbolja jedinka obići 50,9 ćelija, a prosjek prosječne vrijednosti dobrote svih jedinki populacije je samo 40,9. Izvjesno je da se u populaciji nalaze relativno nekvalitetne jedinke iz kojih je teško evoluirati optimalnu jedinku. Jedinke s maksimalnom dobrotom u prosjeku prolaze sobu za 178,8 vremenskih jedinica.

Ako se vremensko ograničenje postavi na 300 jedinki, od 50 izvođenja u samo njih 8 neće biti pronađena jedinka maksimalne dobrote (uspješnost izvođenja 84%), što znači da se izvođenje drastično popravlja naspram prethodnog primjera. Prosječna dobrota najbolje jedinke iznosi 55,5, a prosjek prosječne vrijednosti dobrote svih jedinki iz populacije iznosi 46,86. Jedinke s maksimalnom dobrotom u prosjeku prolaze sobu za 263,9 vremenskih jedinica.

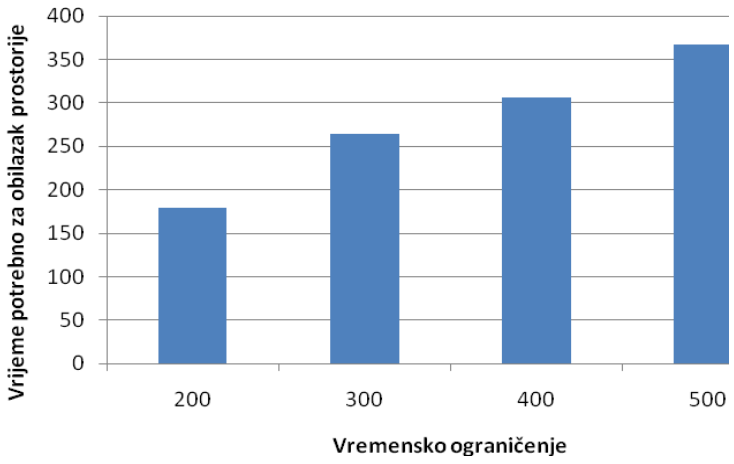
Kada se vremensko ograničenje postavi na 400 jedinica, u svih 50 slučjeva evoluira se jedinka maksimalne dobrote. Te najbolje jedinke će u prosjeku obići svih 56 ćelija u 306,38 vremenskih jedinica. Prosjek prosječne dobrote svih jedinki iz populacije iznosi 48,6.

Ako se vremensko ograničenje postavi na 500 jedinica, što je mnogo više od minimalnog potrebnog broja vremenskih jedinica, logično je da će se u svih 50 izvođenja evoluirati jedinka maksimalne dobrote. Prosjek dobrote najboljih jedinki je, naravno, 56 (svaki put ćemo pronaći jedinku maksimalne dobrote), a prosječno vrijeme obilaska sobe najboljih jedinki iznosi 367 jedinica. Prosjek prosječne vrijednosti dobrote svih jedinica iz populacije iznosi 49,22. To je poprilično visoka brojka, no ona je logična jer roboti imaju iznimno dugo vrijeme za izvođenje programa.

Sve vrijednosti su prikazane na slikom 3.18 i slikom 3.19.



Slika 3.18 Razlike u prosječnim vrijednostima dobrote i kvalitete generacija u ovisnosti o vremenskom ograničenju izvođenja simulacije



Slika 3.19 Prosječno vrijeme potrebno za obilazak prostorije najboljih jedinki

Postaje očito da je vremensko ograničenje bitan parametar. Neophodno je prije pokretanja izvođenja izbrojati minimalan broj vremenskih jedinica potrebnih da robot obiđe sve rubne površine, te ograničenje postaviti na vrijednost višu od minimalne. Ako ograničenje bude

blisko minimalnom, izvođenje će često biti neuspješno, no ako ipak evoluira jedinka maksimalne dobrote, ona će imati izvrsne karakteristike. Što je ograničenje veće, to je izvođenje uspješnije, iako i sporije, no kvaliteta najbolje jedinke opada, a njeno vrijeme potrebno za obilazak prostorije raste.

3.3 Primjer evolucije bez uvjetnih funkcija u skupu operatora

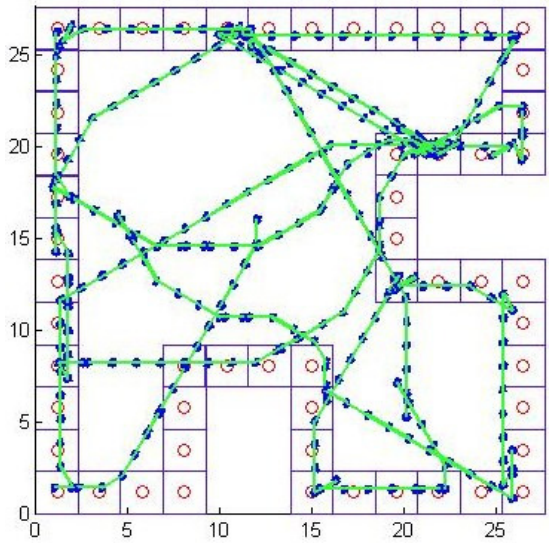
Već je opisano da se supsumpcijska arhitektura može jednostavno prikazati pomoću IF .. ELSE IF uvjetnih funkcija. Iz tog razloga je u skup operatora dodana uvjetna funkciju IFLTE koja omogućuje izgradnju logičke mreže ponašanja u kojoj će jedno ponašanje moći potisnuti drugo. No, što bi se dogodilo kada bi se ta funkciju uklonila iz skupa? Ostala bi samo PROGN2 spojna funkcija čije izvođenje ne ovisi o nikakvim argumentima, već ona samo evaluira dva završna znaka u nizu te vraća vrijednost drugoga. Izvjesno je da se supsumpcijska arhitektitura uopće neće moći razviti, a da će razvijeni upravljački program eventualno davati dobre rezultate isključivo ako ga koristimo na sobi koju smo koristili kao primjer za učenje.

Budući da izvođenje PROGN2, jedine preostale funkcije u skupu operatora, ne ovisi o nikakvim argumentima, senzori na robotu postaju beskorisni, pa se mogu izbaciti iz skupa završnih znakova. Ostaje:

```
F = {PROGN2}; T = {MF, MB, TL, TR}
```

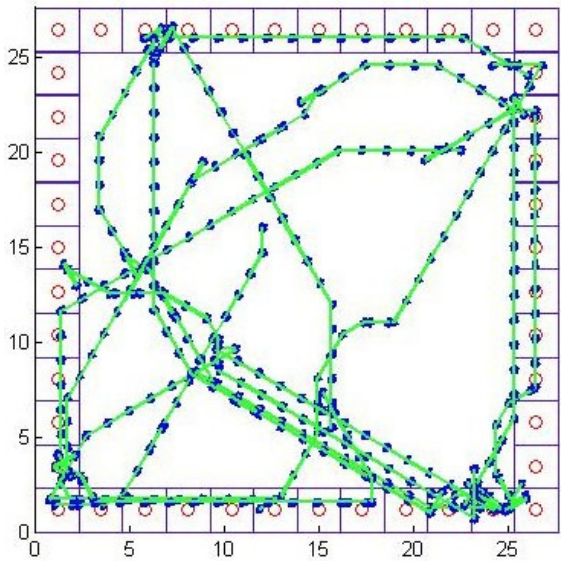
Skup elemenata je zatvoren i potpun.

Nakon pokretanja, sa parametrima iz tablice 3.1, evoluiran je iznimno kompliciran upravljački program, koji uspjeva robota uputiti uz 55 rubnih ćelija dane mu prostorije za učenje. No, po putanji, prikazanoj slikom 3.20, vidi se da je „ponašanje“ robota isključivo nasumično, specifično primjeru za učenje, te nije posljedica pravilno izgrađene upravljačke strukture. Kao takvo je potpuno neupotrebljivo u bilo kojoj drugoj prostoriji.



Slika 3.20 Putanja robota u primjeru za učenje

Ako se robot postavi u drugu prostoriju, čak i mnogo jednostavniju, isključivo sa ravnim zidovima, on će postići mnogo lošije rezultate. Na slici 3.21 je prikazana putanja iz tog primjera, gdje robot obilazi samo 36 od mogućih 44 rubne ćelije.



Slika 3.21 Putanja robota na prostoriji jednostavnijoj od primjera za učenje

Ono što u ovome slučaju robotu omogućava rad je ugrađena zaštita od sudaranja. Ako bi izvršavanje neke od naredbe izazvalo sudar, robot ju jednostavno preskače i izvršava slijedeću naredbu. Bez te zaštite rezultati bi bilo mnogo lošiji i od prikazanih.

3.4 Primjer evolucije s reduciranim brojem senzora

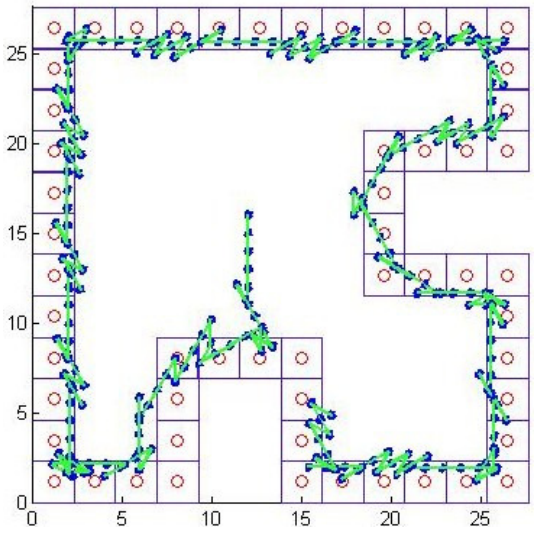
U prethodnim primjerima robot je na raspolaganju imao 12 senzora koji su mjerili udaljenosti iz svih smjerova oko robota, a međusobno su razmaknuti za 30 stupnjeva. Uz njih je robotu bio na raspolaganju i 13 „senzor“, dinamički izračunata najmanja od 12 izmjerenih udaljenosti do zida.

U ovom primjeru će se robotu ograničiti broj senzora na samo četiri, od kojih će dva gledati naprijed i nazad, a dva lijevo i desno. Također, oduzet će mu se i prije spomenuti 13 senzor koji javlja minimalnu udaljenost do zida. Da se još više oteža posao genetskom programu, robotove motorne aktivnosti (MF, ML, TL i TR) više neće vraćati udaljenost tri prednja robotova senzora, već samo nulu. Parametri izvođenja preuzeti su iz tablice 3.1. Jedina promjena je novi skup završnih znakova:

```
T = {S00, S01, S02, S02, EDG, MSD, MF0, MB0, TL0, TR0}
```

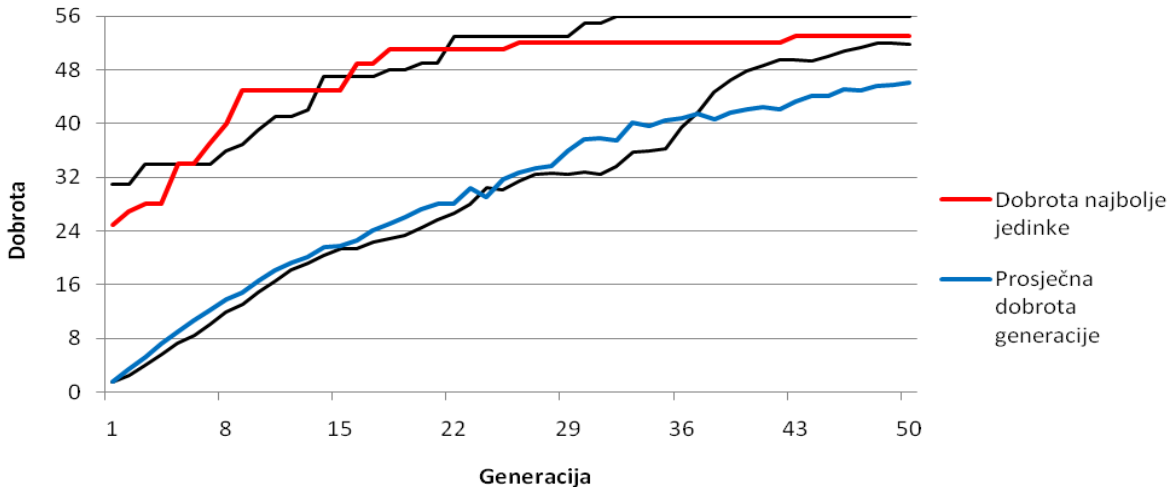
Nakon pokretanja, iznenađuje brz porast dobrote najbolje jedinke i prosječne dobrote populacije iz svake generacije. Očito da pojednostavljeni skup terminala povoljno djeluje na evoluciju. U primjeru iz poglavlja 3.1 u skupu završnih znakova je bilo mnogo više senzora, pa su oni imali i mnogo veći udio u upravljačkim programima iz inicijalnih generacija. Oni ne obavljaju nikakvu motornu funkciju, pa su roboti češće bili neaktivni i time obišli manje rubnih ćelija. No, kroz generacije ta prednost i ubrzani rast pada, te se

počinje osjećati nedostatak senzorskih vrijednosti, a zasigurno u tome ulogu imaju i nova povratna vrijednost motornih funkcija (nula). Iako sustav sa ovakvim parametrima ponekad evoluira kvalitetne jedinke, za to je potrebno mnogo više izvođenja. U pravilu, najbolje jedinke ne postignu niti maksimalnu dobrotu, a kada ju postignu vrijeme potrebno za obilazak prostorije je vrlo blizu vremenskom ograničenju. Slika 3.22 prikazuje putanju robota iz jednog nasumično odabranog pokretanja genetskog programa. Dobrota te jedinke iznosi 54 od maksimalnih 56.



Slika 3.22 Putanja robota iz primjera sa reduciranim brojem senzora

Na slici 3.23 je prikazan porast dobrote najbolje jedinke i prosječne dobrote populacije kroz generacije. Tankom crnom crtom iscrtane su iste vrijednosti i iz primjera iz poglavlja 3.1, u kojem je robot imao na raspolaganju 13 senzora i druge povratne vrijednosti motornih funkcija. Vidi se kvalitetniji start u ovom primjeru, ali i postepeno usporavanje kako generacije prolaze. Ovo je, naravno, samo jedan primjer, no pokazalo se da u prosjeku najčešće javlja upravo ovakva situacija.



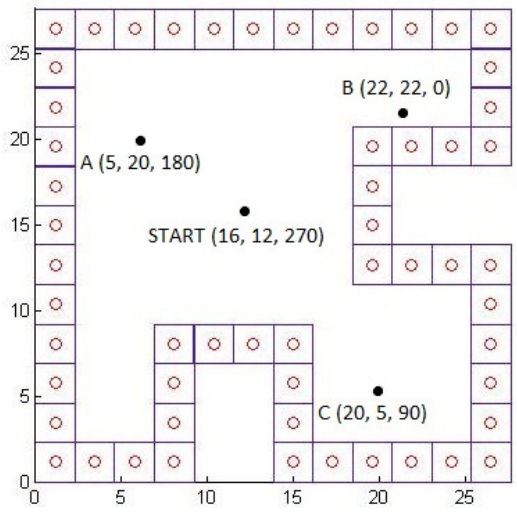
Slika 3.23 Graf napretka dobrote najbolje jedinke i prosječne dobrote generacije primjera sa reduciranim brojem senzora

3.5 Ispitivanje upravljačkih programa na primjerima drugačijim od primjera za učenje

U prethodnim poglavljima je pokazano, na više primjera, da je moguće genetskim programiranjem evoluirati upravljački program koji će navoditi robota uz zidove nepravilno oblikovane prostorije. Kada bi htjeli koristiti robota sa tako razvijenim upravljačkim sustavom u stvarnom svijetu, prostoriju koju bi on obilazio bi postavili kao primjer za učenje, u simulaciji bi trebalo podesiti broj senzora i mogućnosti gibanja robota kao parametre analogno onima iz stvarnog svijeta i rezultirajući evoluirani upravljački program bi zasigurno davao izvrsne rezultate na danoj prostoriji. No, što ako se robota postavi u drugačiju prostoriju? Ili ako bi u danoj prostoriji prostorni razmještaj bio promijenjen (npr. dodan je još jedan ormar uza zid ili je prostorija povećana)? Čak bi i korištenje robota u prostoriji identičnoj primjeru za učenje, ali sa drugačijom početnom pozicijom moglo biti pogubno za njegov rad. Poželjno je razviti program koji će moći odgovoriti na takve promjene bez potrebe za novim evoluiranjem programa, tj. traži se program koji nije specifičan primjeru za učenje i postiže slabe rezultate u drugim uvjetima. Potrebno je razmotriti kako primjeri za učenje i pojedini parametri utječu na razvoj upravljačkih programa.

3.5.1 Ispitivanje upravljačkog programa na primjerima za učenje sa različitim početnim pozicijama

Kada bi bio robot korišten u prostoriji koja je služila kao primjer za učenje, sa početnom pozicijom iz simulacije, za očekivati je da će se robot ponašati kao i u simulaciji. No, što ako se robot koji je učio na nekakvom primjeru ispituje na tom istom primjeru, ali s drugim početnim pozicijama? Za potrebe tog ispitivanja evoluiran je upravljački program, sa parametrima iz tablice 3.1 na prostoriji sa slike 3.1. Jedina varijacija je promjena vremenskog ograničenja izvođenja simulacije, te vremenskog ograničenja izvođenja testova sa drugim startnim pozicijama. Također, koristit će se samo programi koji su davali izvrsne rezultate, konkretno oni sa maksimalnom dobrotom. Slika 3.24 prikazuje početnu poziciju robota iz simulacije, te tri odabrane pozicije sa kojih će robot kretati u testu.



Slika 3.24 Prikaz tri različite početne pozicija robota

U tablici 3.2 su prikazani rezultati. U najlijevijem stupcu su prikazane vrijednosti vremenskog ograničenja izvođenja upravljačkog programa tokom evolucije. U tri velika stupca, sa lijeve strane nalaze se dobrote jedinke za identični primjer, ali sa drugačijim početnim pozicijama. Općenito se iz rezultata zaključuje da razvijeni program, odnosno performanse robota, ne ovise o startnoj poziciji. Primjetna je samo sitna degradacija performansi pri promjeni početne pozicije.

Tablica 3.2 Dobrota jedinki testiranih sa različitim početnim pozicijama

	Pozicija A		**Pozicija B**		**Pozicija C**	
	Vremensko ograničenje izvođenja identično onome iz primjera za učenje	Vremensko ograničenje izvođenja dvostruko veće od onoga iz primjera za učenje	Vremensko ograničenje izvođenja identično onome iz primjera za učenje	Vremensko ograničenje izvođenja dvostruko veće od onoga iz primjera za učenje	Vremensko ograničenje izvođenja identično onome iz primjera za učenje	Vremensko ograničenje izvođenja dvostruko veće od onoga iz primjera za učenje
300	53	53	47	53	44	42
400	53	54	54	55	52	56
500	55	56	56	55	56	56
600	56	55	56	56	55	56

Također, vidi se i da upravljački programi razvijani sa višim vrijednostima ograničenja daju bolje razultate u novim uvjetima, pa je iz istog razloga svaki upravljački program testiran i sa dvostrukom vrijednošću ograničenja. Intuitivno bi bilo da veće vrijeme ostavljeno robotu na obilazak prostorije daje bolje rezultate, no nije se pokazalo tako. Naime, rezultati su samo minorno bolji, pa se može zaključiti da su upravljački programi razvijani sa višom vrijednošću vremenskog ograničenja svestraniji u upotrebu u prostoriji koja jednaka primjeru za učenje, ali ne zbog povećanog vremena izvođenja, nego zbog kvalitetnije razvijenih jedinki. Ako se koriste programi razvijani sa niskim vrijednostima ograničenja, oni će dati ponešto lošije rezultate ako robote stavljamo na početnee pozicije različite od onih iz primjera za učenje.

3.5.2 Ispitivanje upravljačkog programa na primjerima različitim od primjera za učenje

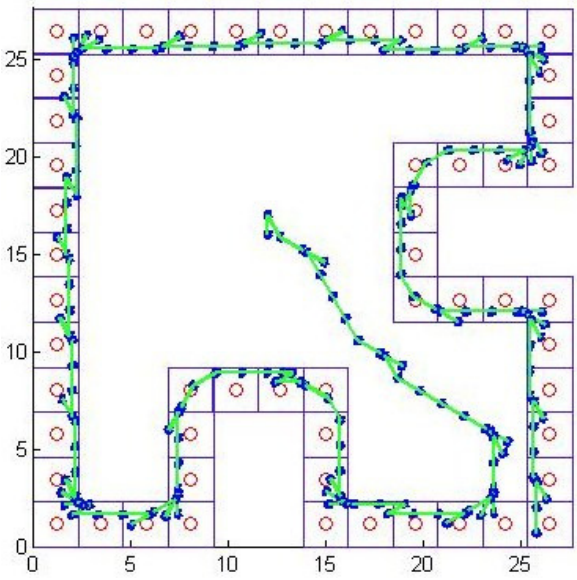
U prethodnom poglavlju je pokazano da promjena početne pozicije ne utječe bitno na rad robota. Preostaje provjeriti da li će se upravljački program snaći ako će biti korišten u prostoriji različitoj od primjera za učenje.

Ovaj korak je bitan jer će se tako moći razlučiti je li upravljački program razvijen isključivo za uvjete na kojima je učio, ili je uistinu razvijena kvalitetna upravljačka arhitektura koja bi bila sposobna snaći se u novim i promijenjenim uvjetima.

Kao primjer za učenje koristit će se prostorija sa slike 3.1 i parametri iz tablice 3.1. Jedina promjena je povećanje vremenskog ograničenja na 500 vremenskih jedinica, jer je pokazano da veće ograničenje u pravilu daje bolje rezultate kada robota koristimo u promijenjenim uvjetima od onih iz primjera za učenje. Kao prostorija za ispitivanje uzeta su tri primjera, jedna potpuno jednostavna kvadratna prostorija, duljine stranice 23 metra,

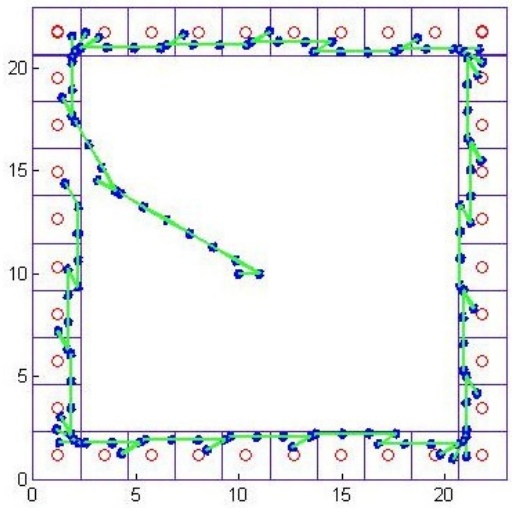
zatim prostorija identična primjeru za učenje, ali zarotirana za 180 stupnjeva oko centra, te jedna nanovo dizajniranu prostorija, „kompliciranija“ od primjera za učenje. Očekuje se da će se upravljački program snaći u jednostavnijim prostorijama, no nepoznato je kakvi će rezultati biti u kompliciranijoj prostoriji.

Za potrebe testiranja odabrana je jednika dobrote 56, čija je putanja na primjeru za učenje prikazana na slici 3.25.



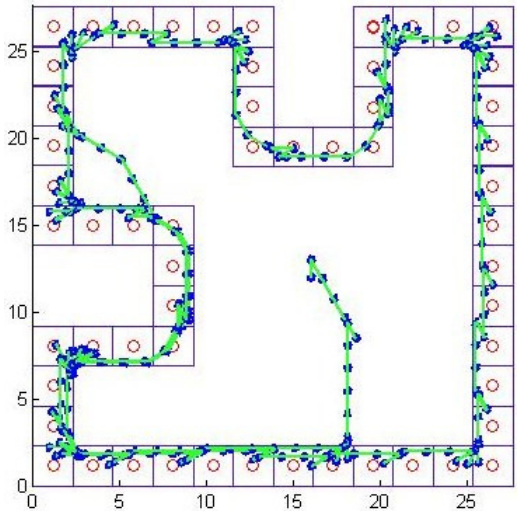
Slika 3.25 Putanja robota na primjeru za učenje

Slika 3.26 prikazuje putanju robota na prvoj od tri ispitne prostorije. Očito je da se robot snašao i više nego dobro, te je obišao sve rubne ćelije prostorije u gotovo minimalnom vremenu. Taj je rezultat očekivan jer je ispitnaa prostorija jednostavnije od primjera za učenje, pred robota stavlja manje zahtjeve. Robot mora pronaći rubne ćelije, te ih potom pratiti uz ravne zidove, te skretati i opet se poravnati uza zid. Testna prostorija je jednostavnija u smislu da nema dvije izbočine, prepreke, uz donji i lijevi zid.



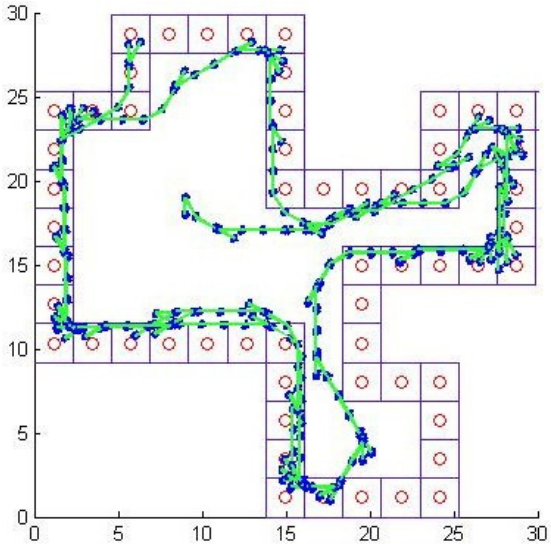
Slika 3.26 Putanja robot na prvoj ispitnoj prostoriji

Slika 3.27 prikazuje putanju robota na ispitnoj prostoriji sličnoj onoj primjeru za učenju, samo zarotiranom za 180 stupnjeva. I opet se upravljački program izvrsno snašao u „novim“ uvjetima. To je očekivani rezultat jer prostorija nema nikakav kompleksniji element od prostorije na kojoj se upravljački program razvijao.



Slika 3.27 Putanja robota na drugoj testnoj prostoriji

Slika 3.28 prikazuje putanju robota na trećoj, najkompleksnijoj ispitnoj prostoriji. Robot ovdje nailazi na probleme, upravljački program se ne snalazi u novoj okolini. Rezultati nisu iznimno loši, ali robot ne obilazi sve rubne ćelije.



Slika 3.28 Putanja robota na trećoj tesnoj prostoriji

Kvalitetni rezultati u prvoj i drugoj ispitnoj prostoriji pokazuju da razvijeni upravljački program nije specifičan primjeru za učenje i da isključivo tamo daje dobre rezultate, no na trećoj ispitnoj prostoriji vidi se da nije odabran optimalan primjer za učenje. Korišteni primjer nije dovoljno reprezentativan da bi se na temelju njega razvio upravljčki program koji bi se mogao koristiti na potpuno nasumičnoj prostoriji, i veličinom i mogućim preprekama, ta na njima davati zadovoljavajuće rezultate.

3.6 Odabir optimalnog primjera za učenje

U prethodnom poglavlju je pokazano da upravljački program razvijan na jednom primjeru za učenje varira kvalitetom upravo u ovisnosti o tom primjeru. Jasno je da će robot uspješno obilaziti zidove prostorije identične primjeru za učenje, no, ako želimo koristiti robota i u drugim prostorijama, bitno je izabrati kvalitetan primjer za učenje. Mora se razmotriti koji su elementi primjera za učenje bitni i moraju biti sadržani u njemu kako bi općenitost razvijene ponašajne arhitekture bila najveća moguća.

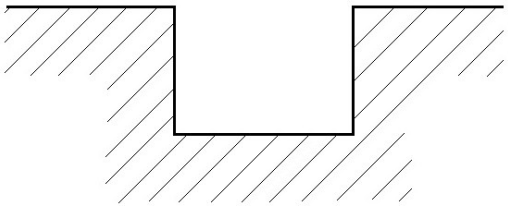
U ovim primjerima koristit će se isključivo prostorije sa jednim i kontinuiranim nizom rubnih ćelija. To su prostorije koje nemaju „otoke“, npr. potporne zidove u sredini. U primjer za učenje nužno je staviti sve elemente koji se mogu pojaviti u takvoj prostoriji. To su:

- ravni zidovi
- kutevi prostorije
- prepreke

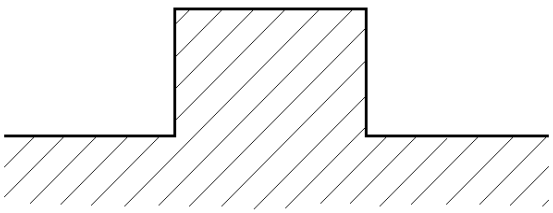
Ravni zidovi su sastavni dio svake prostorije, no u primjeru je potrebno imati barem par potpuno ravnih površina. Ako je primjer za učenje iznimno nepravilan postoji opasnost da se upravljački sustav ne bi snašao u prostorijama sa ravnim zidovima, pa čak i onim jednostavnim poput prostorije sa slike 3.26.

Robot mora biti sposoban skrenuti kada dođe do kuta prostorije, te se poravnati uz drugi zid. Nemoguće je osmisliti prostoriju bez kuteva pa će robot razvijen u praktički bilo kakvoj prostoriji biti sposoban savladati kuteve, te se ovaj element neće dalje razmatrati.

Ono na što robot može naići uz ravne zidove su razne prepreke. No, kada i najkompleksniji splet prepreka razloži na osnovne dijelove ostaju nam samo dva elementa: udubine, poput one sa slike 3.29, i izbočine, sa slike 3.30.

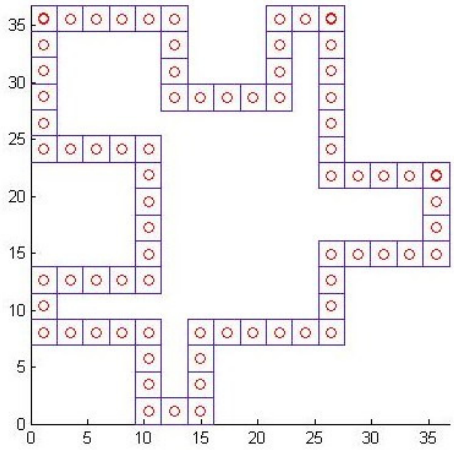


Slika 3.29 Primjer udubine u prostoriji



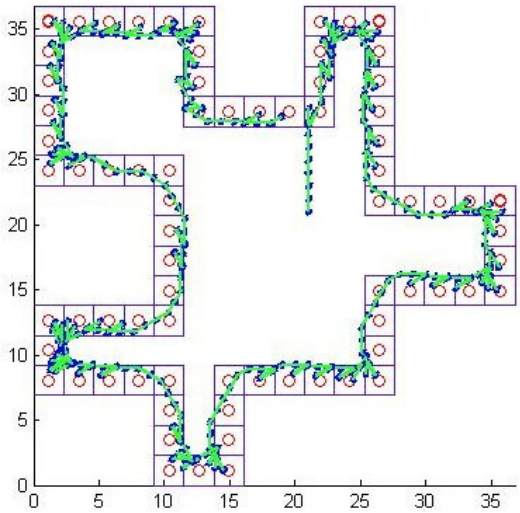
Slika 3.30 Primjer izbočine u prostoriji

Uz tri gore navedena elementa, bitna je i početna pozicija robota. On mora biti u prostoru koji nije uza zid, tj. ne smije biti unutar neke od rubnih ćelija, jer se tako neće razviti ponašanje koje će osigurati robotovo traženje rubnih ćelija. Kada bi u primjeru za učenje robotova početna pozicija bila u jednoj od rubnih ćelija, robot bi odmah po startu simulacije krenuo sa obilaskom zida. Kada bi robota sa istim upravljačkim programom stavili u „prazan“ prostor postoji opasnost da se upravljački program ne bi snašao u novim uvjetima, ne bi znao pronaći rubne ćelije. Na slici 3.31 je osmišljena prostorija. Jasno je da postoje kutevi, a dodani su i dovoljno duge ravne površine zidova. Uz gornji i lijevi zid su dodane dvije izbočine, različite površine, a uz desni i donji zid su umetnute dvije udubine, također različitih površina.



Slika 3.31 Optimalan primjer za učenje

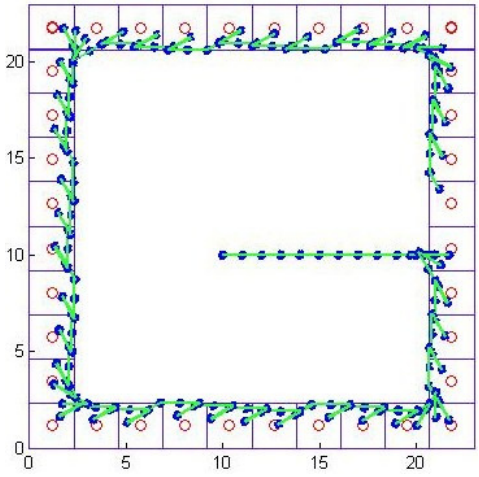
Nužno je staviti prepreke različitih površina jer one veće robot lakše obilazi. Može ih se razložiti na 3 kuta i 3 ravna zida, dok je kod manjih prepreka potreban manji radijus zakretanja, pa je moguće da klasično obilažnje neće biti dovoljno. Parametri izvođenja uzeti su iz tablice 3.1, osim što će trajanje izvođenja simulacije biti 600 vremenskih jedinica. Brzo pronalazimo jedinku maksimalne dobrote čiju putanju prikazuje slika 3.32.



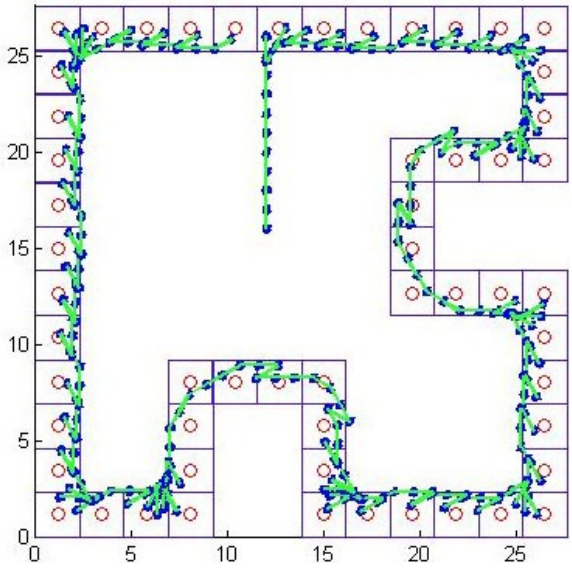
Slika 3.32 Putanja robota na primjeru za učenje

Za ispitivanje se koriste četiri prostorije, a vremensko ograničenje se povećava na 1200 jedinica, što će se kasnije pokazati nepotrebno jer će u sve tri testne prostorije robot obići sve rubne površine u manje od 600 vremenskih jedinica. To je u skladu sa eksperimentom iz poglavlja 3.2, u kojem je pokazano da povećavanje ograničenja pri testiranju ne daje bolje rezultate.

Prve dvije sobe su obična kvadratna prostorija, bez ikakvih prepreka, te prostorija sa slike 3.1. Obje su relativno jednostavne i u obje robot uspijeva u minimalnom vremenu obići sve rubne ćelije. Putanja robota u prvoj prostoriji prikazana je slikom 3.33, a putanja u drugoj prostoriji prikazana je slikom 3.34.

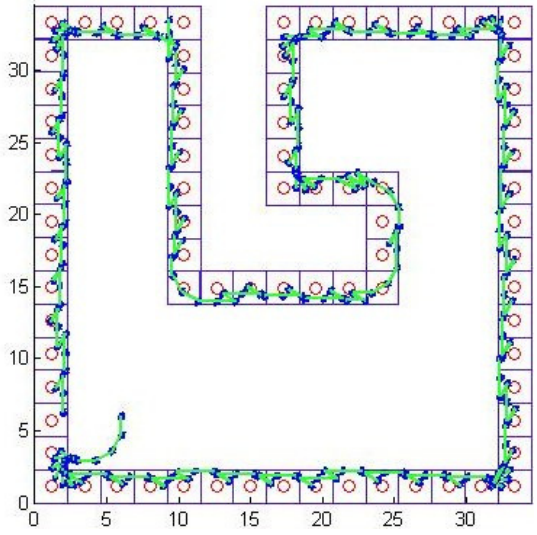


Slika 3.33 Putanja robota u prvoj testnoj prostoriji

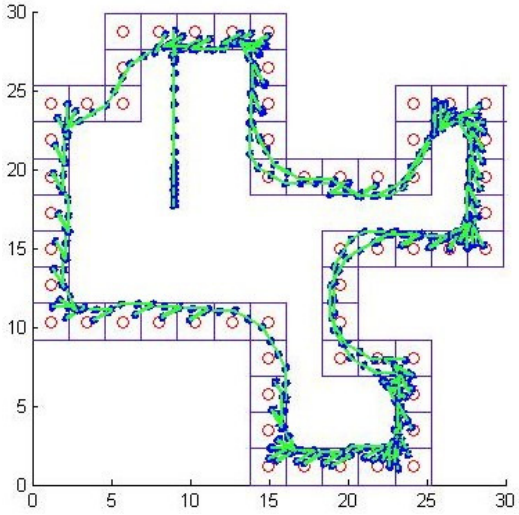


Slika 3.34 Putanja robota u drugoj testnoj prostoriji

Nakon što se upravljački program pokazao uspješan na jednostavnim primjerima, preostaje testirati ga na kompliciranim sobama, za što su dizajnirane dvije prostorije. Putanje robota u tim prostorijama su prikazane slikom 3.35 i slikom 3.36. Upravljački program se izvrsno snalazi u obje prostorije te postiže maksimalnu dobrotu u gotovo minimalnom vremenu.



Slika 3.35 Putanja robota u trećoj testnoj prostoriji



Slika 3.36 Putanja robota u četvrtoj testnoj prostoriji

Iako se dvije prostorije ne čine iznimno komplicirane, one su komplicirane koliko mogu biti. Svaka se prepreka, koliko god komplicirana, može rastaviti na skup izbočina i udubina, a upravljački program je sposoban savladati takve prepreke. Zaključuje se da je primjer za učenje dovoljno reprezentativan i da će upravljački program evoluiran nad tim primjerom biti dovoljno općenit da savlada bilo kakvu prostoriju.

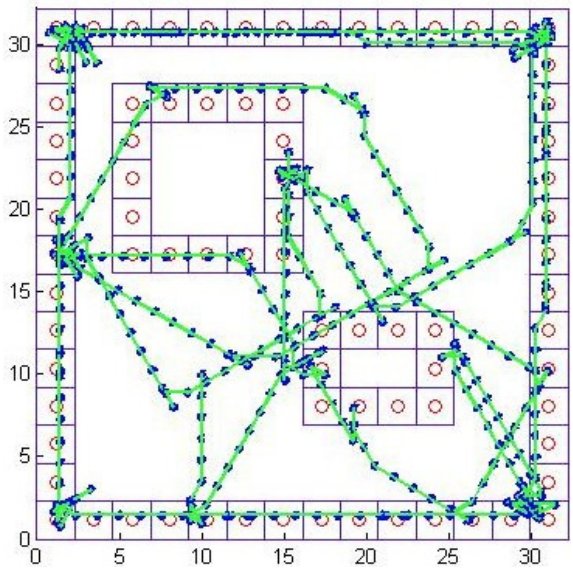
3.7 Optimiranje rada, nedostaci i potencijalni problemi

Do sada je pokazano kako je evolucija supsumpcijske arhitekture genetskim programiranjem ne samo moguća, već i jednostavna, brza i sa izvrsnim rezultatima. Prethodni primjeri su bili opće prirode, bez ulaženja u specifične primjere, no dovoljno općeniti da prikažu sposobnost, kako sustava, tako i rezultirajućih upravljačkih programa. Nužno je razmotriti moguća poboljšanja i dodatne mogućnosti sustava, ali i mane, nedostatke i moguće probleme.

Genetsko programiranje je kompleksan i vremenski zahtjevan proces. Koza je predložio dvije metode ubrzavanja razvoja. Jedna je ograničavanje vremena trajanja simulacije, koje je bilo implementirano opisani sustav iz prethodnih poglavlja. Taj parametar ima snažan utjecaj na razvoj jedniki pa ga je nužno pažljivo odabrati. Uz njega, Koza je predložio da se prije pokretanja slobodna površina sobe diskretizira na sitne ćelije, te se za svaku od njih izmjeri i pohrani 12 udaljenosti koje mjere senzori. Na taj način nije potrebno tokom izvođenja konstantno iznova računati udaljenosti, već se one očitavaju iz baze spremljenih udaljenosti na temelju lokacije robota.

Kvaliteta robotovog praćenja zida može se regulirati povećanjem ili smanjenjem površine rubnih ćelija u prostoriji. Manje rubne ćelije će rezultirati kvalitetnijim praćenjem, ali će usporiti razvoj upravljačkog programa. Podešavanje ovog parametra je odraz potreba iz stvarnog svijeta, gdje će robot biti upotrebljavan. Ako nije potrebna velika preciznost, razvoj upravljačkog programa može se olakšati i ubrzati povećavanjem rubnih ćelija.

Svi primjeri za učenje korišteni u ovome radu su imali samo jednu, kontinuiranu rubnu površinu. Sve rubne ćelije su bile spojene i povezane. Moguće je da prostorija ima „otoke“, krute objekte na svojoj sredini. U prethodnim poglavljima razvijani upravljački programi ne mogu rješiti takav problem. Supsumpcijska arhitektura, u svojoj najjednostavnijoj implementaciji kakva se ovdje razvija, nema internu memoriju. Bez memorije robot ne može znati koje je rubne površine obišao. Kada bi upravljački program mogao pamtiti mapu prostora, onda bi bilo moguće riješiti ovaj problem. Kada robot obiđe neku površinu on ne zna da je sa njom gotov te da treba krenuti dalje tražiti nove neposjećene rubne ćelije. Čak i ako posve slučajno nastavi obilaziti zidove na nekim drugim mjestima, moguće je da se vrati na već posjećene površine, jer ne zna da je tamo bio. Nekoliko izvođenja sa takvim prostorijama je pokrenuto, sa slabim rezultatima. Eventualno je moguće, nakon mnogo pokretanja, pronaći upravljački program koji će na temelju posve slučajnog gibanja obići rubne površine prostorije, no taj program će biti specifičan primjeru za učenje te će biti potpuno neiskoristiv u drugim sobama. Slika 3.37 prikazuje jedan takav primjer. Primjećuje se da robot donekle i prati zidove uz rub prostorije, ali kad treba prijeći na centralne površine potpuno se gubi i samo nasumično luta.



Slika 3.37 Primjer putanje robota u prostoriji sa „otocima“

Zaključak

U ovom radu je pokazano kako je moguće razviti računalni program koji navodi robota uz zidove nepravilne prostorije, koristeći paradigmu genetskog programiranja.

Upravljački programi razvijeni u okviru rada sastoje se od kompozicije uvjetnih funkcija koje ispituju vrijednosti niza senzora, te na temelju njih pokreću razne motorne funkcije robota, poput gibanja unaprijem unazad ili skretanja, a sve sa ciljem praćenja zidova prostorije. Razvijeni programi su u skladu sa supsumpcijskom arhitekturom. Prikazana je evolucija upravljačkog računalnog prgrama u duhu supsumpcijske arhitekture koristeći evolucijski proces upravljan i navođen isključivo mjerom dobrote.

Činjenica da je moguće evoluirati supsumpcijsku arhitekturu koja rješava određeni problem sugerira da je je takav pristup dekompoziciji problema koristan u izgradnji rješenja težih problema, tako da grupiramo ponašanja koja rješavaju zadatke dok problem nije riješen.

U radu je prikazana upotreba raznih metoda ubrzavanja evolucije, utjecaj parametara na razvoj i kvalitetu rješenja, a posebno optimalan odabir primjera za učenje, koji je ključan za stvaranje kvalitetnih upravljačkih programa.

Genetsko programiranje se pokazuje kao iznimno sposobno oruđe, koje uz minimalan trud programera brzo i učinkovito rješava probleme koji bi za čovjeka bili daleko od trivijalnih. Upravljački program autonomnog mobilnog robota u duhu supsumpcijske arhitekture, čiji bi razvoj tražio poprilično vrijeme i trud programera, tehnikom genetskog programiranja izgrađen je brzo i jednostavno. Razvijeni upravljački programi se pokazuju robusnima (rade i bez nekih senzora), brzima (nema izgradnje mape) i učinkovitima (roboti zadaće obavljaju u gotovo minimalnom vremenu).

Literatura

[1] Koza, J. R. *Genetic Programming – On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[2] Mataric, Maja J. *A Distributed Model for Mobile Robot Environment-Learning and Navigation*. MIT Artificial Intelligence Laboratory technical report AI-TR-1228, 1990.

[3] Poli, Langdon, McPhee *A field guide to genetic programming.* Creative Commons, 2008.

[4] Vincent, Brown *Evolutionary game theory, natural selection nad darwinian dynamic.* Cambridge University Press, 2005.