SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 219

EVOLUCIJA AUTOMATSKOG UPRAVLJANJA UPORABOM PARALELNIH EVOLUCIJSKIH ALGORITAMA

Lovro Paić-Antunović

Zagreb, svibanj 2011.

Opisati problem automatskog upravljanja agentima u simuliranoj okolini.
Ostvariti programski sustav za pronalaženje upravljačkog algoritma uporabom evolucijskog računanja. Ispitati mogućnosti paralelizacije sustava na razini više procesa i više dretvi. Ocijeniti učinkovitost dobivenih upravljačkih algoritama i ostvarenih metoda paralelizacije. Radu priložiti algoritme, izvorne tekstove programa i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Sadržaj

Sažetak

U ovom radu objašnjen je problem automatskog upravljanja robotom u proizvoljno zadanoj okolini. Predstavljen je uvod u genetsko programiranje, te je naglasak stavljen na njegove paralelne izvedbe. Opisana je klasifikacija paralelnih implementacija te dano objašnjenje njihovih ključnih karakteristika. Napravljeno je i opisano simulacijsko okruženje za potrebe ispitivanja raznih aspekata genetskog programiranja u primjeni na problem automatskog upravljanja. Primjena memorijskih funkcija je ugrađena te je ispitan njezin utjecaj na pojačanje ekspresivnosti i sposobnost nalaženja rješenja. Također, ispitan je utjecaj evolucijskog učenja na više soba.

Ključne riječi: Genetski algoritmi; Genetsko programiranje; Paralelno programiranje; Automatsko upravljanje robotom;

Summary

In this work we explain the problem of automatic control of a robot in arbitrarily shaped environment. We present an introduction to genetic programming and focus on parallel versions of it. We discuss different varieties of parallel implementations and resulting solutions dependent on their characteristic parameters. A simulation environment is designed with ability to test various aspects of parallel genetic programming applied to the problem of automatic control. Use of memory functions for bolstering expressiveness of learned program is implemented and its impact on quality of reached solution tested. We also test consequences of evolving programs using multiple rooms.

Key words: Genetics algorithms; Genetic programming; Parallel programming; Automatic control of robot;

1. Uvod

Automatsko upravljanje interesantan je koncept omogućen napredcima u tehnologiji i znanosti. Ono uključuje upravljačke programe za sve od autonomnih letjelica do kućnih usisavača ili dodataka za sigurnost u cestovnoj vožnji. Nalaženje robusnih upravljačkih programa predstavlja aktualan problem za mnoga područja, uključujući i računarsku znanost. Takvi roboti obično imaju mogućnost percipirati okruženje koristeći razne senzore pa bi se glavni zadatak upravljačkih programa mogao definirati kao traženje metoda koje odlučuju o sljedećoj akciji na temelju interpretacije primljenih podataka.

Važno je uočiti razliku između dvije verzije navedenog problema. Prva verzija predstavlja automatsku kontrolu u poznatom okruženju, odnosno okruženju za koje postoji prikaz u memoriji robota. Druga uključuje kontroliranje u nikad prije viđenim okruženjima. Na primjer, kada bi robot trebao usisati sobu posjedujući znanje o njezinim dimenzijama i obliku, mogao bi učinkovitije i jednostavnije planirati svoju rutu. U ovom radu naglasak je stavljen na potonju verziju u kojoj se pokušava pronaći kontrolni program primjenjiv na prethodno neviđene sobe, koristeći samo senzore za daljinu. Pri tom se koriste dva zadatka: prvi zadatak zahtjeva od robota da dođe dovoljno blizu svakom zidu ili prepreci u sobi, a drugi predstavlja prolazak po što većoj površini sobe. Prvi zadatak se može koristiti za rekonstrukciju sobe, odnosno prelazak u već navedeni lakši problem kontrole gdje su unaprijed poznati oblik i veličina sobe, budući da bi se, primjerice, točnija mjerenja okoline mogla provesti što bliže je senzor prepreci. Drugi zadatak uveden je prvenstveno zbog demonstracije fleksibilnosti evolucijskog učenja, no on također demonstrira i korisno ponašanje u zadacima poput čišćenja površine ili košenja trave.

U prirodi, evolucija i prirodna selekcija se neprestano ponavljaju. “Prirodna selekcija je proces kojim biološke osobine postaju više ili manje učestale u populaciji zbog konzistentnih utjecaja na preživljavanje ili reprodukciju njihovih nositelja” [1]. Taj proces je zapravo vrlo jednostavan, ali ima vrlo iznenađujuće rezultate. „Evolucijsko računanje je pojam koji obuhvaća nekoliko vrsta postupaka pretraživanja prostora rješenja: genetske algoritme (GA), genetsko programiranje (GP), evolucijske strategije, evolucijsko programiranje i dr.“[2]. U ovom radu fokusirat ćemo se na primjenu genetskog programiranja na izabrani problem. Genetsko programiranje (GP) je podvrsta genetskih algoritama (GA) koja kao rješenja stvara

računalne programe. Genetski algoritmi predstavljaju tehniku strojnog učenja temeljenu na evoluciji i prirodnoj selekciji. Jednostavnije rečeno, to je tehnika bazirana na vjerojatnosti da je kombiniranjem ili malom izmjenom osobina uspješnih roditelja moguće dobiti još uspješnije potomke Detaljniji uvod u genetsko programiranje dan je u 3. poglavlju ovog rada.

Paralelno programiranje je način programiranja u kojem se višestruki dijelovi programa mogu paralelno, odnosno istovremeno, izvoditi. Budući da simuliranje opisanog evolucijskog procesa može biti računalno vrlo skupo, ono predstavlja zanimljivo područje za primjenu paralelnog programiranja. Međutim, korist bržeg računanja nije jedina prednost paralelizacije GA. „Zapravo, oni uspijevaju postići idealni cilj u kojem se paralelni algoritam ponaša bolje nego suma zasebnih dijelova njegovih pod-algoritama“[3]. Četvrto poglavlje je posvećeno različitim načinima paralelizacije genetskih algoritama, te njihovim prednostima i manama.

2. Genetsko programiranje

U ovom poglavlju ukratko su objašnjeni osnovni principi genetskog programiranja. U prvom dijelu objašnjen je princip na kojem se temelje genetski algoritmi općenito, a to je proces evolucije pomoću prirodne selekcije. U drugom dijelu je objašnjena osnovna podatkovna struktura kojom se barata u genetskom programiranju. U trećem i četvrtom dijelu objašnjeni su operatori pomoću kojih se modeliraju procesi iz prirode: reprodukcija, križanje (engl. *crossover*), te mutacija. U konačnom petom dijelu je objašnjena funkcija dobrote (engl. *fitness*) koja definira uspješnost pronađenih rješenja.

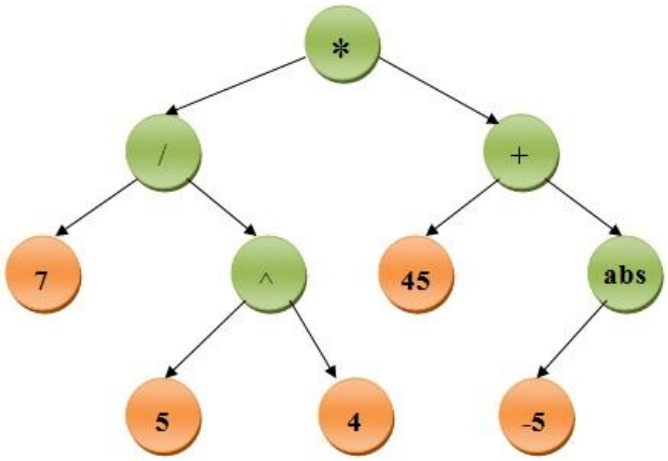
2.1. Temelj genetskog programiranja – evolucija

Genetski algoritmi općenito, pa tako i specifičnije genetsko programiranje temelje se na principu evolucije iz prirode. Evolucija je proces promjene genetskog materijala populacije organizama iz generacije u generaciju. Evolucijom upravlja prirodna selekcija. "Prirodna selekcija je proces kojim biološke osobine postaju više ili manje učestale u populaciji zbog konzistentnih utjecaja na preživljavanje ili reprodukciju njihovih nositelja" [1]. Ovaj proces je konstantan u prirodi i njegovo odvijanje se obavlja automatski jer sposobnije jedinke imaju veću šansu preživljavanja te pronalaska partnera za reprodukciju. Križanje je proces kojim se kombinacijom genetskog materijala dvaju roditelja dobiva genetski materijal potomka. Budući da genetske osobine jedinki imaju znatan utjecaj na njihovu sposobnost preživljavanja, razumno je očekivati da će kombinacijom osobina dvaju uspješnih roditelja nastati i relativno uspješno dijete, iako to ne mora nužno biti slučaj. Također, značajan proces za evoluciju u prirodi predstavlja i genetska mutacija. Mutacija je proces kojim unutar jedne jedinke dolazi do promjene genetskog koda, obično uzrokovane nekim vanjskim utjecajem poput radijacije.

2.2. Struktura kromosoma prikazana stablom

Kako bismo uspješno simulirali navedene genetske procese potrebna nam je struktura koja omogućava zapis teksta programa, ali ujedno omogućava i proces kombiniranja dvaju

tako zapisanih programa. U tu svrhu se u genetskom programiranju obično koriste stablaste strukture koje opisuju željeno ponašanje. Takvo stablo predstavlja usmjereni hijerarhijski graf koji sadrži dvije vrste vrhova - čvorove i listove. Čvorovi reprezentiraju operacije, a listovi njihove operatore. Jedan primjer takvog stabla dan je slikom 2.1.

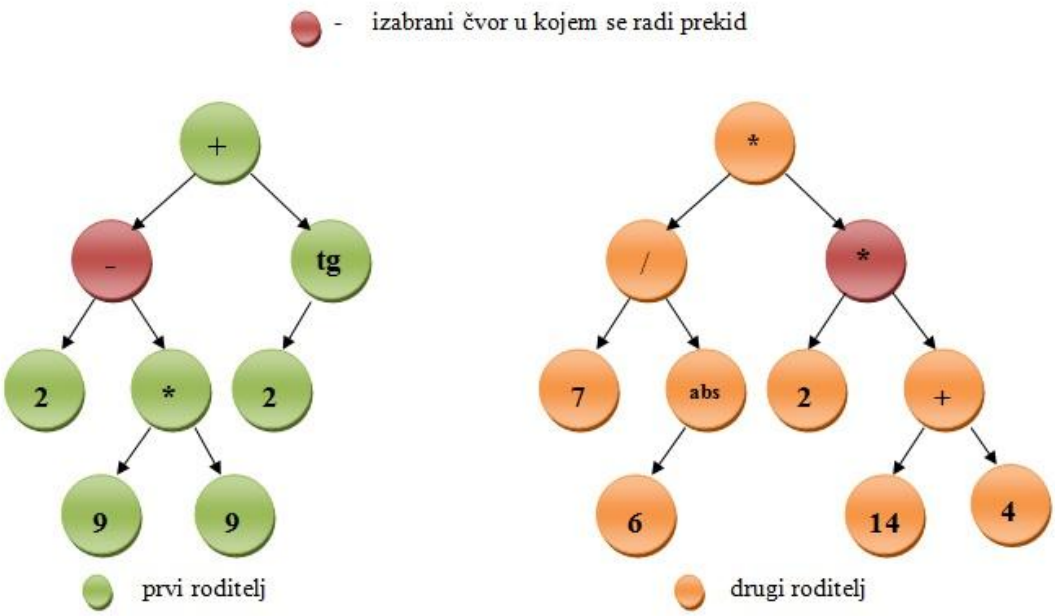


Slika 2.1Stablo koje u *inorder* obilasku predstavlja zapis izraza (7/(5^4))*(45+abs(-5))

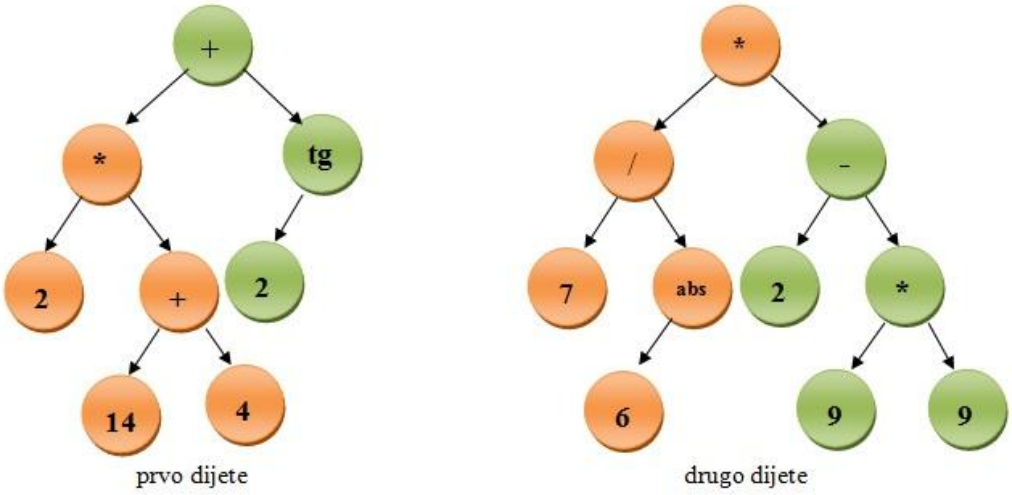
Na prethodnoj slici zelenom su bojom označeni svi čvorovi koji predstavljaju operacije, a narančastom operatori. Navedeni operatori i operacije bili bi pogodni za prikaz matematičke funkcije, no nisu pogodni za prikaz programskog rješenja upravljačkog programa nekog robota. Iz tog je razloga obično potrebno definirati posebne operatore i operacije ovisne o problemu koji se pokušava riješiti. O izabranim operatorima i operacijama za rješavanje automatskog upravljanja biti će više govora u poglavlju 5 ovog rada.

2.3. Križanje

Križanje (engl. *crossover*), u prirodi se javlja kod svake seksualne reprodukcije. To je proces kojim se kombinacijom genotipova dvaju roditelja dobiva novi genotip njihovog potomka. Kao što je navedeno u prethodnom dijelu ovog poglavlja, jedan od najčešćih zapisa genotipa korištenih u genetskom programiranju je upravo struktura stabla, prema tome, križanje bi se moglo definirati kao proces kojim bi se od dva različita stabla dobilo novo, drugačije stablo, koje sadrži karakteristike oba roditelja. Postoje različite metode kojima je moguće ostvariti zadani cilj. Jedna od osnovnih metoda sastoji se od nasumičnog odabira čvora prekida u oba roditelja, te međusobne zamjene njih i njihovih podstabla. Takvo križanje prikazano je slikama 2.2 i 2.3 [4].



Slika 2.2. – Roditelji koji će se križati

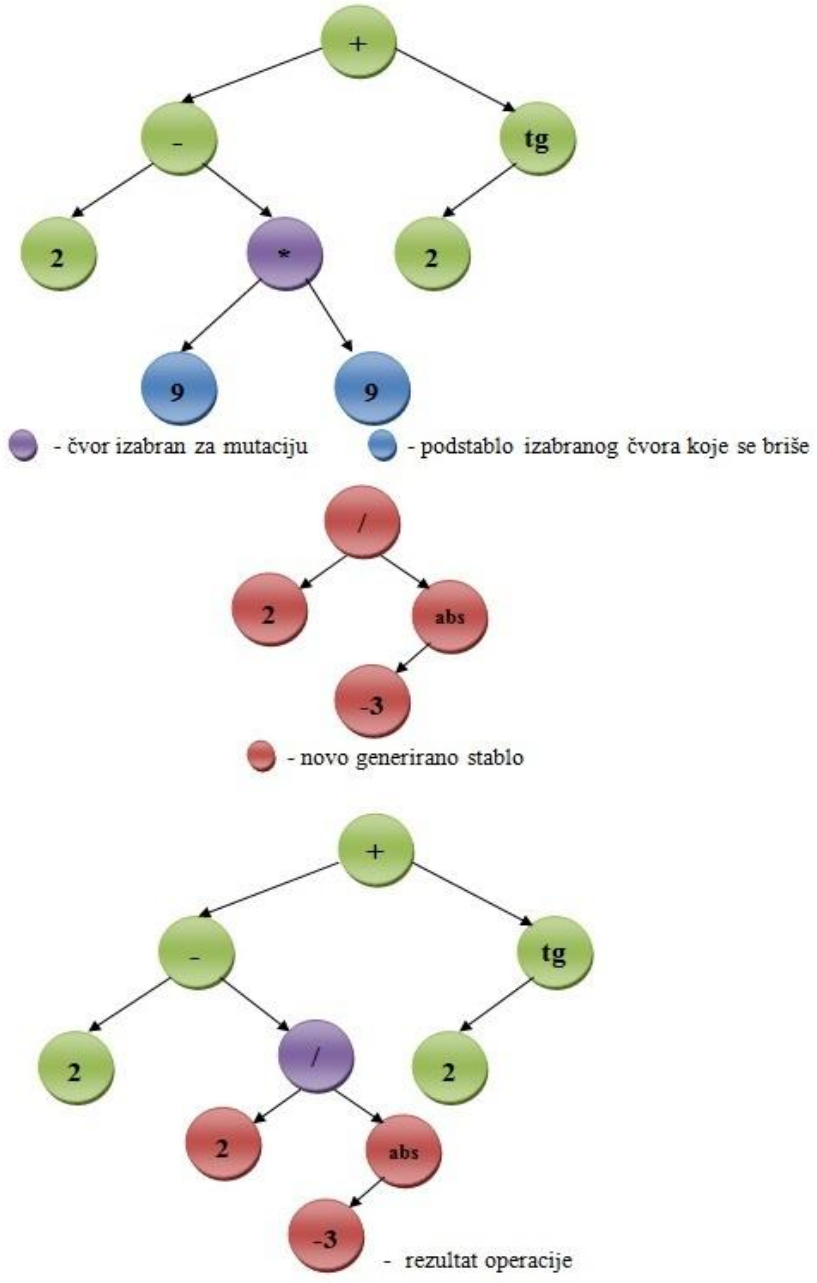


Slika 2.3 Produkt križanja roditelja sa slike 2.2.

2.4. Mutacija

Zadatak mutacije u genetskim algoritmima je uvođenje i očuvanje raznolikosti genetskog materijala prisutnog u populaciji. Ona djeluje na način da mijenja određeni dio izabrane jedinke. U genotipu reprezentiranom stablom to se obično izvodi na način da se

nasumično izabere jedan čvor stabla, te u njemu nasumično generira novo stablo. Primjer takve mutacije dan je slikom 2.4 [4].



Slika 2.4 Primjer mutacije genotipa reprezentiranog zapisom pomoću stabla

2.5. Funkcija dobrote

Funkcija dobrote (engl. *fitness*) predstavlja alat kojim se mjeri uspješnost pojedinog pronađenog rješenja. Ona je ključna za proces evolucijskog učenja jer omogućava simulaciju procesa prirodne selekcije. Pomoću nje se indirektno određuje vjerojatnost kojom će pojedina

jedinka biti izabrana za sudjelovanje u kreiranju populacije sljedeće generacije. O funkciji dobrote najviše ovisi ponašanje pronađenih rješenja, jer je to jedini način definiranja traženog ponašanja. Na primjer ukoliko je cilj napraviti robota koji posjećuje sve prepreke u sobi, funkcija dobrote mogla bi iznositi broj posjećenih prepreka; ukoliko je pak cilj doći što dalje u sobi od početne točke, funkcija dobrote bi mogla biti zračna udaljenost konačne i početne točke, itd. Bez funkcije dobrote algoritam ne bi znao razlikovati poželjno od nepoželjnog ponašanja.

3. Parelelno genetsko programiranje

Postoje mnogi razlozi zašto ima smisla paralelizirati proces evolucijskog učenja. Za početak, evolucijsko učenje može biti poprilično računalno skup proces. Osim toga, sam algoritam spada u kategoriju trivijalno paralelnih (engl. *embarrassingly parallel*) algoritama vrlo pogodnih za paralelizaciju [5]. Prema tome, samo ubrzanje koje dolazi kao produkt paralelizacije je već dostatan razlog za njegovu primjenu. Međutim, ubrzanje nije jedina prednost paralelnih verzija genetskih algoritama: „Dokazi o većoj učinkovitosti, boljem očuvanju raznolikosti, dodatnoj dostupnosti memorije i procesorske snage, te mogućnosti višestrukih rješenja osnažuju važnost istraživačkih napredaka na području paralelnih genetskih algoritama.“ [3].

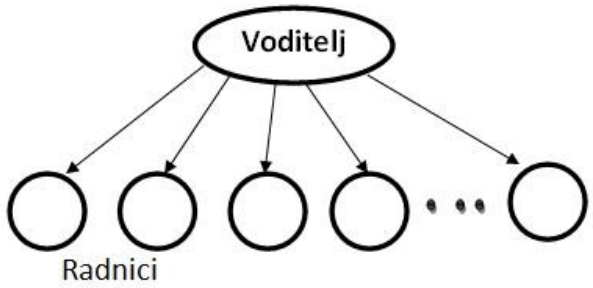
3.1. Klasifikacija

Postoje brojni načini ostvarenja paralelne paradigme genetskih algoritama koji se u literaturi obično svrstavaju u 4 osnovne kategorije[6],[7], makar postoje i druge klasifikacije:

1. Jednopopulacijski GA, poznati i kao voditelj-radnik GA (engl. *master-slave*) ili globalno paralelni GA (GPGA)
2. Višepopulacijski GA poznati i kao raspodijeljeni GA (DGA)
3. Fino granulirani GA poznati i kao masivno paralelni GA (MPGA)
4. Hijerarhijski GA

Osim ovih kategorija, neki izvori navode još kategorije: hibridni GA[6], koji uključuju uporabu neke druge metode optimiranja, i trivijalno paralelni GA[7], koji odgovara neovisnom pokretanju na različitim procesorima.

Jedno-populacijski GA su vrlo slični slijednoj verziji algoritma s razlikom da se obično samo evaluacija funkcije dobrote odvija paralelno, makar je moguće paralelizirati i genetske operatore. Ovakav pristup ima smisla ukoliko je glavni cilj paralelizacije ubrzanje algoritma. Jedno-populacijski GA se obično implementiraju pomoću voditelj-radnik odnosa prikazanog slikom 3.1.



Slika 3.1 – Voditelj-radnik GA

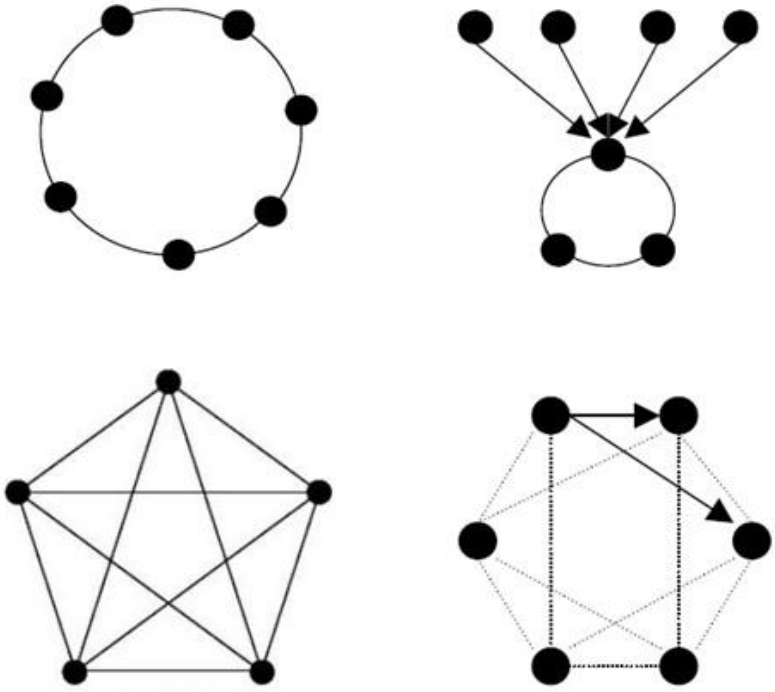
U tom odnosu jedan čvor kojeg nazivamo voditeljem sadrži cijelu populaciju i obično vodi proces evolucije, delegirajući zadatke poput računanja vrijednosti funkcije dobrote radnicima. Svaka jedinka iz populacije se može križati s bilo kojom drugom jedinkom, baš kao i u slijednoj verziji algoritma. Postoje dvije osnovne vrste GPGA: sinkroni i asinkroni. U sinkronoj verziji algoritam čeka primitak odgovora za sve jedinke u generaciji prije prelaska u iduću. U asinkronoj verziji nema čekanja, već se konstantno odvija proces evolucije. Sinkrona je verzija stoga primjerenija generacijskim GA, dok je asinkrona primjerenija eliminacijskim (engl. *steady-state*) GA. U ovom su radu implementirane i ispitane dvije verzije jedno-populacijskih sinkronih GA. Detaljnije informacije o implementiranim verzijama mogu se naći u idućem, 4. poglavlju, dok je usporedba performansi navedenih inačica dana u 5. poglavlju.

Više-populacijski GA su najpopularnija metoda paralelizacije GA [6]. Oni se sastoje od više slijednih GA od kojih svaki posjeduje vlastitu populaciju. U određenim trenutcima izmjenjuju se određene jedinke između odvojenih populacija, što ovaj algoritam čini uvelike različitim od neovisnog pokretanja GA na različitim procesorima ili računalima. Više-populacijski GA uvode 7 novih parametara koji definiraju rad algoritma [6],[7]. To su:

1. *Migracijski interval* koji definira učestalost razmjene jedinki dviju odvojenih populacija
2. *Migracijska stopa* koja definira broj jedinki koje se razmjenjuju
3. *Strategija odabira jedinki* koje će se slati drugim populacijama
4. *Strategija odabira zamjene* jedinki primljenim jedinkama iz drugih populacija
5. *Topologija migracije* koja definira koje populacije smiju slati kojima jedinke (neke poznatije topologije prikazane su slikom 3.2.)

6. *Broj odvojenih populacija*
7. *Veličina odvojenih populacija*

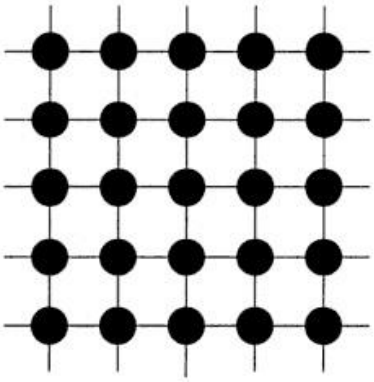
Više-populacijski GA vrlo su pogodni za implementaciju na više računala, jer time iskorištavaju višestruku količinu dostupne memorije i procesorske moći, dok se određivanjem topologije te intervala i stope može definirati trošak komunikacije.



Slika 3.2 Redom od gore lijevo : prstenasta topologija, „injection island“ topologija, potpuno povezana topologija i prstenasta sa 2 susjeda (sve slike osim potpunog grafa preuzete iz [6]).

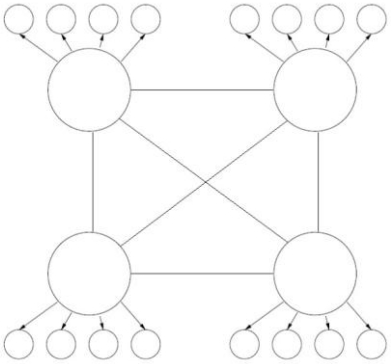
Detalji o implementaciji više-populacijskog GA u ovom radu mogu se naći u 5. poglavlju, dok se ispitivanja izabranih specifičnih parametara mogu naći u 5. poglavlju.

Fino granulirani GA (slika 3.3) sastoje se od jedne prostorno strukturirane populacije u kojoj se svaka jedinka smije kombinirati samo s neposrednim susjedima. „Susjedstva se preklapaju, pa se u konačnici dobra svojstva superiorne jedinke mogu prenijeti na cijelu populaciju“ [5]. Fino granulirani GA nisu detaljnije istraživani u ovom radu.

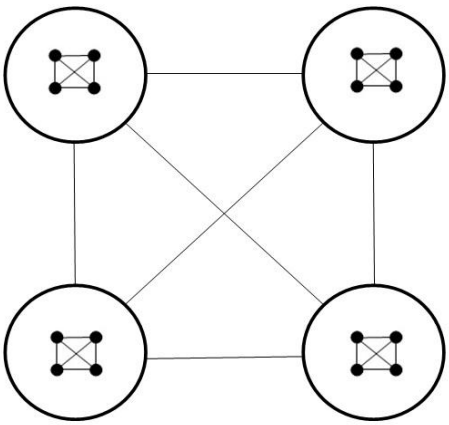


Slika 3.3 Fino granulirani GA (slika preuzeta iz [5])

Hijerarhijski GA predstavljaju kombinaciju bilo kojih od navedenih metoda paralelizacije. Na primjer, više-populacijski GA u kojem je svaka populacija realizirana paralelnim jedno-populacijskim GA (slika 3.4) ili više-populacijski GA u kojem je svaka populacija realizirana više-populacijskim GA (slika 3.5). U ovom radu su implementirane obje navedene verzije, s dodatkom da je zapravo moguće napraviti i više-populacijski GA u kojem je svaka populacija realizirana više-populacijskim GA u kojem je svaka populacija realizirana paralelnim jedno-populacijskim GA.



Slika 3.4 Hijerarhijski GA sastavljen od više populacijskog GA u kojem je svaka populacija realizirana jedno-populacijskim GA



Slika 3.5 Hijerarhijski GA sastavljen od više populacijskog GA u kojem je svaka populacija realizirana više populacijskim GA.

Trivijalni GA su pojam koji se koristi za višestruko, nepovezano pokretanje slijednog GA na više računala ili procesora.

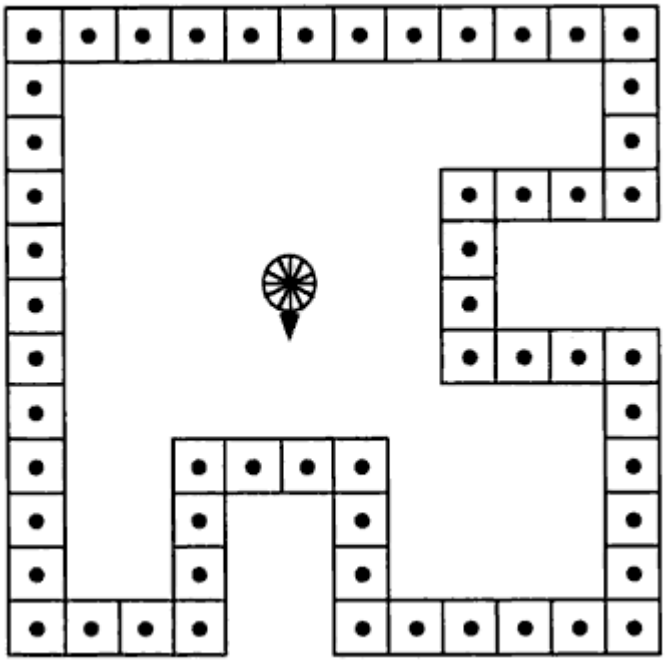
„Hibridni GA kombiniraju paralelne GA sa nekom od klasičnih metoda optimizacija, primjerice, s lokalnom „hill climbing“ metodom“ [6].

4. Simulacijsko okruženje

Ovo poglavlje bavit će se implementacijskim detaljima ostvarenog simulacijskog okruženja. Prvi dio poglavlja posvećen je opisu izabranog problema te načinu na koji se navedeni problem rješava genetskim programiranjem. Drugi dio bavi se dvjema implementiranim verzijama jedno-populacijskih GP, dok treći govori o više-populacijskim i hijerarhijskim verzijama GP. U četvrtom dijelu se opisuju motivacija i način ostvarenja učenja ponašanja na više soba te koevolucija. U zadnjem, petom poglavlju, govori se o ugrađenim memorijskim funkcijama. Za izradu programske izvedbe korišten je Microsoftov C# programski jezik te „Visual Studio Ultimate 2010“ razvojna okolina, Microsoftov High Performance Cluster Pack, te MPI.NET. Za izradu korisničkog sučelja korištena je WPF (Windows Presentation Foundation) tehnologija. Za pokretanje programa potrebno je imati instaliran Microsoftov .net framework verziju 4. Za pokretanje učenja je potrebno imati instaliran Microsoftov HPC.

4.1. Problem automatskog upravljanja pomoću GP

Ovaj radi bavi se dvama različitim zadacima autonomnog upravljanja robotom. Prvi zadatak je problem praćenja zidova kojeg je 1992. John R. Koza opisao u svojoj knjizi *„Genetic programming: On the programming of computers by means of natural selection*“. U tom zadatku cilj je približiti se dovoljno blizu svim zidovima u proizvoljno zadanoj sobi. Slika 4.1 prikazuje sobu na kojoj je John R. Koza ispitivao svoj GP. U ovom radu izabrana je teža verzija zadanog problema. Naime, u Kozinoj verziji je zadatak obići rub sobe u kojoj ne postoji niti jedna dodatna prepreka osim vanjskih zidova, dok se u ovdje implementiranoj verziji smiju pojaviti i prepreke koje ne predstavljaju vanjski zid sobe već objekte kroz koje se ne može prolazit i koji su postavljeni unutar sobe.



Slika 4.1 Soba sa 56 rubnih područja na kojoj je J. Koza ispitivao svoj GP.

Drugi zadatak kojim se bavi ovaj rad jest problem prekrivanja cijele površine sobe, kao što bi to, primjerice, radio autonomni usisavač. U ovoj „originalnoj“ verziji problema jedine informacije dostupne robotu u bilo kojem trenutku su mjere 12 senzora, koji mjere udaljenost do prve prepreke u smjeru u kojem su orijentirani, te dinamički generirani 13. senzor koji vraća najmanju udaljenost. U ovom radu istražena je još mogućnost pamćenja nekih akcija kako bi se povećala izražajnost generiranih programa i dostupnost informacija, o kojima će više govora biti u petom dijelu ovog poglavlja. Nakon definiranja problema koji

želimo riješiti, moramo definirati ključne parametre koji ostvaruju paradigmu genetskog programiranja [8]:

1. Skup završnih znakova (listova u strukturi stabla),
2. Skup funkcijskih znakova (čvorova),
3. Funkciju dobrote,
4. Parametre i varijable koji kontroliraju pokretanje,
5. Kriterij zaustavljanja.

U ovom radu izabrani su isti parametri koje je koristio i John R. Koza u svojem radu (uz dodatak memorijskih funkcija).

Skup završnih znakova je definiran kao T= {S00, S01, S02, S03,…, S11, SS, MSD, EDG}.

„S00“ do „S11“ su listovi koji vraćaju vrijednost pripadajućih senzora. SS je dinamički generirani senzor koji vraća najmanju udaljenost prethodnih 12 senzora, dok su „MSD“ (engl. *minimmum safe distance*) i EDG (engl. *edging distance*) dodatni numerički parametri koji samo vraćaju svoju vrijednost.

Skup funkcijskih znakova je definiran kao F={TR, TL, MF, MB, IFLTE, PROGN2, IMEM1, IMEM2}.

„TR“ i „TL“ su akronimi od „Turn Right“ i „Turn Left“, odnosno funkcije koje rotiraju robota oko njegovog centra za 30 stupnjeva u smjeru kazaljke na satu ili obrnuto.

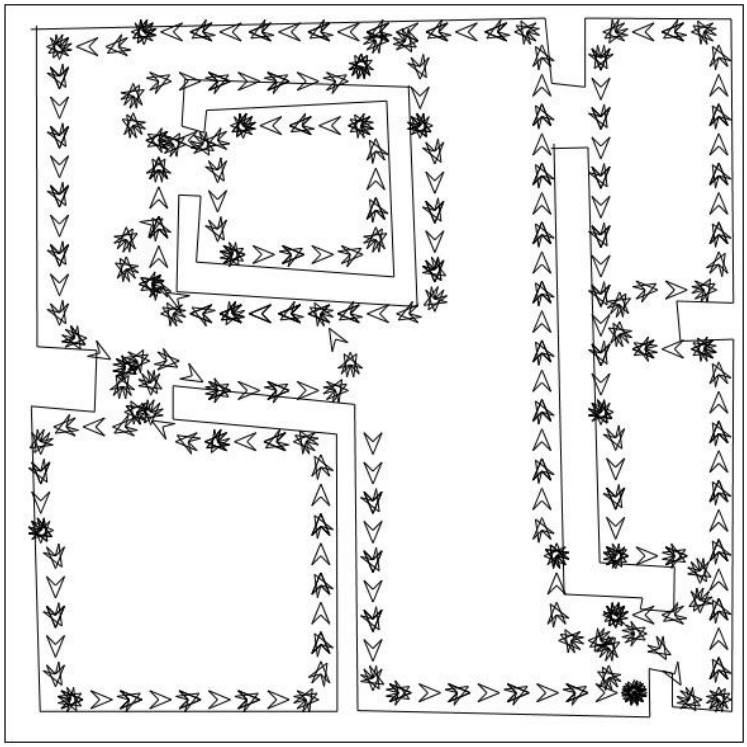
„MF“ i „MB“ su akronimi od „Move Forward“ i „Move Backward“, odnosno funkcije koje pomiču robota za korak unaprijed ili korak unazad. U ovoj implementaciji moguće je zadati veličinu koraka. Prije obavljanja ovih funkcija ispituje se bi li njihovo izvođenje rezultiralo kolizijom, te ukoliko bi, funkcija se ne izvodi. Ovaj mehanizam je uveden kako ne bi dolazilo do sudaranja robota sa zidom. U originalnoj verziji Johna R. Koze koristi se vrijednost od 110% veličine koraka, ali zbog diskretizacije (objašnjeno u nastavku) u ovoj verziji bi takva implementacija rezultirala kolizijama.

„IFLTE“ znači „IF Less Than or Equal“ i predstavlja jedinu kontrolnu funkciju koju je koristio Koza u svojem radu. „IFLTE“ je funkcijski čvor koji prima 4 parametra. Ukoliko je vrijednost prvog parametra manja ili jednaka drugom parametru, evaluira se i vraća vrijednost trećeg parametra, dok se u suprotnom evaluira i vraća vrijednost četvrtog.

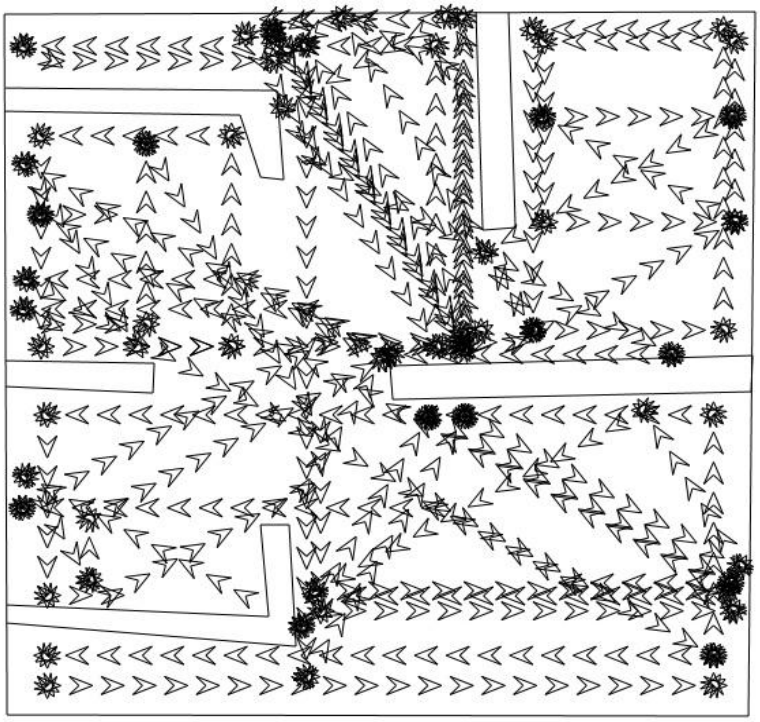
„PROGN2“ je funkcija koja spaja dvije funkcije na način da evaluira obje, te vrati vrijednost druge.

„IMEM1“ i „IMEM2“ su objašnjeni u petom dijelu ovog poglavlja.

U oba odabrana zadatka je cilj robota proći nekim područjem u sobi, stoga nam je potrebna funkcija dobrote koja pridjeljuje numeričku vrijednost jedinkama ovisno o količini posječenih traženih područja. Iz tog razloga se u diskretizirano polje koje predstavlja sobu, ovisno o zadatku, na područja koja želimo da robot posjeti, postavljaju „jabuke“ koje se skupljaju kada im robot dođe dovoljno blizu. Funkcija dobrote je jednaka u oba zadatka koji se obrađuju, a definirana je kao broj skupljenih „jabuka“. Nakon skupljanja svih jabuka u sobi prekida se izvođenje i bilježi se broj izvršenih motornih funkcija koji predstavlja vrijeme obilaženja sobe. U slučajevima kada se ne uspiju skupiti sve jabuke se simulacija izvršava dok ne istekne definirano maksimalno vrijeme izvođenja (parametar *Max time)*. U posebnom slučaju kada je broj skupljenih jabuka između dvije jedinke jednak, bolja je ona koja ih uspijeva skupiti u kraćem vremenu Razlog zbog kojeg se ipak istom funkcijom dobrote uspijevaju postići različiti ciljevi je sam razmještaj jabuka. Sve što je potrebno je da u slučaju praćenja zidova jabuke postavimo samo na zidove odnosno prepreke, dok u slučaju prekrivanja površine jabuke stavimo po cijeloj površini. Slika 4.2 prikazuje jedan uspješan primjer obilaženja ruba sobe, dok slika 4.3 ilustrira uspješno obavljen drugi zadatak.



Slika 4.2 Uspješno riješen zadatak obilaska svih prepreka sobe



Slika 4.3 Uspješno obavljen zadatak posjećivanja cijele površine sobe

Radi ubrzanja rada algoritma, proizvoljno nacrtana soba uvijek se predprocesira na način da se kreira dvodimenzionalna matrica polja, te se za svako polje unaprijed izračunaju vrijednosti udaljenosti do zidova. Pomoću parametara „Precision X“ i „Precision Y“ moguće je definirati veličinu tog polja, ali za potrebe ovog rada uvijek će se koristit matrica 100x100. Unutar programskog okruženja ostvarenog u sklopu ovog rada moguće je crtati sobe proizvoljnog oblika koje ne ovise o veličini dvodimenzionalne matrice, odnosno bilo koju sobu je moguće prikazati bilo kojom veličinom matrice.

4.2. Paralelni jedno-populacijski GP

Kao što je već prije rečeno, implementirane su dvije verzije paralelnog jednopopulacijskog GP.

Prva verzija odgovara standardnom voditelj-radnik modelu u kojem voditelj čuva populaciju i obavlja genetske operacije, a posao računanja vrijednosti funkcije dobrote delegira radnicima. Ova verzija ostvarena je pomoću MPI tehnologije, na način da jedan proces odgovara jednom radniku, dok je posebni nulti proces voditelj. U ovom slučaju radnici ne moraju znati ništa o genetskom algoritmu, već samo sadrže klasu „RoboSimulator“ pomoću koje simuliraju

kretanje robota u zadanoj sobi ili sobama, te vraćaju broj pokupljenih jabuka i proteklo vrijeme.

Druga verzija implementira paralelnost stvaranjem proizvoljnog broja dretvi. Broj dretvi je moguće zadati na dva načina. Jedan način broji dostupne fizičke jezgre te taj broj množi podesivim faktorom „threads per cpu“, dok drugi način dozvoljava eksplicitno definiranje brojem. Paralelizacija se postiže koristeći sljedeća pravila:

1. Razlikuju se memorijske pozicije jedinki parnih i neparnih generacija.
2. Ukoliko je tekuća generacija (ona u kojoj stvaramo jedinke) parna, ne mijenja se memorijska struktura prethodne generacije koja je bila nužno neparna. Isto pravilo vrijedi i obrnuto, ukoliko je tekuća generacija neparna, ne mijenjaju se jedinke prethodne parne generacije.
3. Ukoliko je tekuća generacija parna, sve dretve imaju pristup za čitanje svim jedinkama u prethodnoj neparnoj generaciji. Ovo pravilo također vrijedi obrnuto.
4. Ukoliko je tekuća generacija parna, svaka dretva smije mijenjati samo podskup memorijskih lokacija jedinki parne tekuće generacije za koje je od početka rada algoritma zadužena. Kao i prethodna, i ovo pravilo naravno vrijedi i u drugom smjeru.
5. Svaka dretva posjeduje vlastite memorijske lokacije za potrebe obavljanja genetskih operacija (križanja, mutacije i reprodukcije).
6. Ne prelazi se u sljedeću generaciju dok sve dretve ne popune njima dodijeljene memorijske pozicije jedinki.

Koristeći ova pravila dobiva se paralelna verzija algoritma sa svim drugim svojstvima identičnim slijednoj verziji. Ovim načinom paralelizirani su i genetski operatori i računanje funkcije dobrote, uz očuvanje velikog izbora jedinki koje pružaju jedno-populacijski GA.

4.3. Paralelni više-populacijski i hijerarhijski GP

Više-populacijski GP u ovom je radu implementiran koristeći MPI tehnologiju na način da jedan proces odgovara jednoj odvojenoj populaciji. Migracijski interval implementiran je na način da se parametrom „Rounds till saturation“ može podesiti broj

generacija koji mora proći bez promjene najboljeg pronađenog programa unutar jednog procesa, prije nego se od drugih procesa traže njihove jedinke. Ovaj pristup ima smisla zato što je obično baš migracija pokretač promjene kvalitete jedinki. „Ova teorija (op. a.: teorija „preciznog ekvilibrijuma“, engl.: *punctuated equilibria*) predlaže da većinu vremena nema značajnih promjena u populaciji (znači da je u ekvilibrijumu), ali neki događaji okidaju brze evolucijske promjene. Cohoon i suradnici su primijetili da migracija može biti takav događaj.“[5]. Migracijska stopa je također podesivi parametar, a zadaje se kao postotak ukupne populacije jednog otoka (populacije jednog procesa). Strategija odabira jedinki za slanje definirana je na način da se šalju najbolje jedinke iz cijele populacije, dok je strategija odabira jedinki za zamjenu definirana na način da se izbacuju najlošije jedinke u populaciji. Broj odvojenih populacija moguće je zadati parametrom „Num of processes“, a veličinu populacije svakog otoka parametrom „Population size“.

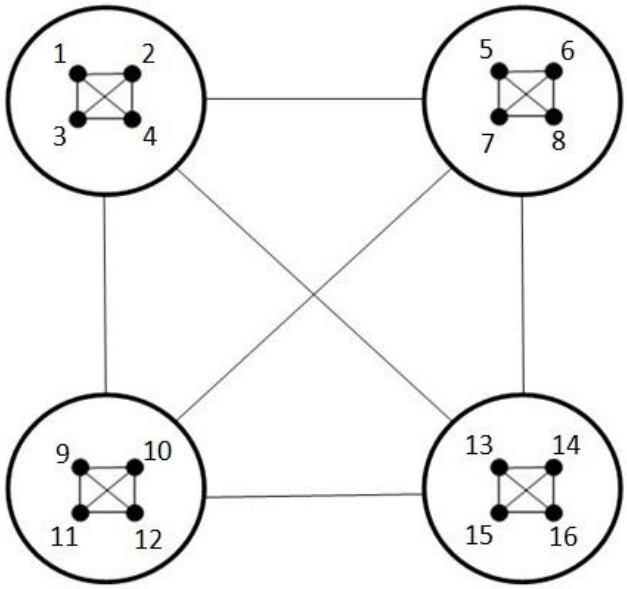
Za potrebe definiranja topologije napravljena je struktura koja omogućava njezino potpuno proizvoljno definiranje. To je ostvareno definiranjem susjedstva svakog procesa pomoću tekstualne datoteke. Na primjer, tekstualna datoteka prstenaste dvosmjerne strukture sa 4 procesa bi izgledala ovako:

```
1 T2 2
2 T2 1
2 T2 3
3 T2 2
3 T2 4
4 T2 3
4 T2 1
1 T2 4
```

Prvi red ove datoteke predstavlja pravilo koje govori da proces 1 smije slati jedinke procesu 2, odnosno, zbog načina implementacije migracijskog intervala, da proces 2 smije tražiti jedinke od procesa 1. Drugi red definira pravilo da proces 2 smije slati jedinke procesu 1, odnosno da proces 1 smije tražiti jedinke od procesa 2. Na ovaj način moguće je definirati bilo koji graf, pa je stoga moguće definirati i bilo koju topologiju. Makar se u algoritmu uvijek radi s ovako definiranom datotekom, pri pokretanju je moguće izabrati jednu od dobro poznatih, unaprijed definiranih topologija, nakon čega program sam generira ovakvu datoteku, ovisno o zadanim parametrima topologije. Ugrađene dostupne topologije su: potpuno povezana, jednosmjerna i dvosmjerna prstenasta struktura, „Injection island“ te prstenasta struktura s 2 susjeda.

Također, posebno se može zadati parametar želi li se pri svakom okidanju migracije tražiti jedinke od svih dostupnih susjeda, ili se nasumično bira samo jedan od njih.

Hijerarhijski pristup ostvarivanja više-populacijskog GP u kojem je svaka populacija definirana više-populacijskim GP omogućen je dodavanjem dodatnog parametra migracijskog intervala za migracije između procesa koji ne pripadaju istom otoku na višoj razini.



Slika 4.4 Hijerarhijski više-populacijski GP u kojem je svaka populacija ostvarena više-populacijskim GP.

Slika 4.4 ilustrira podjelu na više i niže razine. Procesi 1 i 2 predstavljaju različite otoke na nižoj razini koji međutim spadaju u isti otok u višoj razini. Procesi 1 i 5 pripadaju različitim otocima u višoj razini. Jedina preostala potrebna modifikacija je mogućnost definiranja odnosa pripadnosti istom otoku u višoj razini. To je ostvareno koristeći oznake T1 (od „tier 1“) ili T2 u definiranju topologije susjedstva pomoću već opisanih tekstualnih datoteka. Definiranjem odnosa kao T2 zapravo se govori da dva korespondentna procesa pripadaju istom otoku u višoj razini, dok suprotno, odnos T1 govori da navedeni procesi pripadaju drugim skupinama otoka u višoj razini.

I standardni i hijerarhijski više-populacijski pristup mogu još iskorištavati mogućnost podjele na dretve opisane u prethodnom dijelu ovog poglavlja, čime se omogućuju dvije nove verzije hijerarhijskog pristupa: više-populacijski GP u kojem je svaka populacija ostvarena paralelnim jedno-populacijskim GP, te više-populacijski GP u kojem je svaka populacija ostvarena više-populacijskim GP u kojem je svaka populacija ostvarena paralelnim jedno-populacijskim GP.

4.4. Učenje na više soba i koevolucija

Motivacija za uvođenje učenja na više soba je dvojaka. Jedan razlog potiče iz činjenice da se za neke jednostavnije sobe, poput na primjer one na kojoj je John R. Koza ispitivao svoj GP, rješenja koja uspijevaju pokupiti sve jabuke u sobi dosta često znaju naći već u nultoj generaciji, odnosno običnim nasumičnim generiranjem programa. Tokom provedenih ispitivanja se čak znalo dogoditi da najbolji program nulte generacije uspije obaviti dani zadatak i u znatno kraćem vremenu od maksimalnog. Ispitivanja pokazuju da se uvođenjem učenja na više soba znatno otežava problem pronalaska rješenja. Razlog tome je što postoji puno veći broj zadovoljavajućih programa koji obilaze samo jednu sobu, nego što postoji generaliziranijih koji funkcioniraju u općenitijem slučaju. Budući da je jedna od glavnih uloga izabranog problema automatskog upravljanja robotom u ovom radu bila omogućiti ispitnu okolinu pomoću koje se mogu ispitati razni aspekti GP, otežavanje problema bilo je nužan uvjet. Drugi razlog je specijaliziranost nađenih rješenja, odnosno, prijašnjim istraživanjem [4] utvrđeno je da programi naučeni u jednoj sobi često potpuno zakazuju u drugačije definiranim sobama. Iz tog se razloga postavlja pitanje bi li programi naučili općenitije ponašanje ukoliko bi funkcija dobrote ovisila o simulaciji na više soba. Ovo pitanje odgovoreno je u 5. poglavlju ovog rada. Uvođenje učenja na više soba zahtjeva blagu izmjenu funkcije dobrote. Jednostavno se generirani program pokrene na svim sobama umjesto na jednoj, te se zbrajaju skupljene jabuke i potrošeno vrijeme.

„Kao i sa klasičnim genetskim algoritmima, koncept koevolucijskih algoritama dolazi iz promatranja prirode. Uistinu, priroda je sačinjena od nekoliko vrsta koje koevoluiraju. I umjesto evoluiranja populacije sličnih jedinki koje reprezentiraju globalno rješenje (kao u klasičnim genetskim algoritmima), mi razmatramo koevoluciju pod-populacija jedinki koje reprezentiraju specifične dijelove globalnog rješenja.“ Koevolucija u ovom radu je implementirana na način da je za različite procese i u jedno-populacijskim i više-populacijskim verzijama moguće specifično zadati sobe na kojima će evoluirati jedinke. Ispitivanje učinkovitosti ovog pristupa za izabrani problem je zanimljivo zbog činjenice da učenje na više soba umnogostručuje trajanje algoritma jer se funkcija dobrote mora računati onoliko puta koliko ima soba. Korištenjem koevolucije možemo izbjeći takvo višestruko računanje, jer možemo zadati da svaki proces i dalje evaluira funkciju dobrote na samo jednoj sobi.

4.5. Ugrađene memorijske funkcije

Potreba za memorijom proizlazi iz činjenice da simulirani robot niti u jednom trenutku ne zna je li već prošao nekim dijelom sobe ili nije. Drugim riječima, koristeći samo senzore za udaljenost, robot će na istoj poziciji sa istom orijentacijom uvijek jednako reagirati. Takvo ponašanje predstavlja problem za sobe koje sadrže prepreke odvojene od rubnih zidova, jer će se robot u situaciji gdje je moguće ići u 2 smjera iz određene točke, uvijek odlučiti za isti smjer. Ukoliko je ta točka na primjer bila jedina iz koje se može krenuti u drugom, još neposjećenom smjeru, ili robot u svim točkama iz kojih je taj novi smjer dostupan uvijek odlučuje poći već posjećenim smjerom, jedan dio sobe ostati će neposjećen.

S ciljem rješavanja tog problema, dizajnirane su dvije pomoćne memorijske funkcije. Tako definirane funkcije, nazvane „IMEM1“ i „IMEM2“ dodane su skupu funkcijskih znakova generiranih programa. Obje funkcije su upravljačke kao i „IFLTE“, odnosno, ovisno o rezultatu njihovog izvođenja izvode se različiti dijelovi programa.

Korištenje „IMEM1“ funkcije zahtjeva da robot pri svakom pomicanju provjerava je li došao blizu zida, te ukoliko jest, pamti koordinate tog zida. Ovakvo ponašanje moglo bi se u stvarnosti implementirati na način da robot svoju startnu poziciju pri paljenju smatra, primjerice, ishodištem koordinatnog sustava, te zatim poznajući svoju brzinu kretanja te brzinu zaokretanja procjenjuje trenutnu poziciju u kojoj se nalazi. Izvođenje „IMEM1“ funkcije tada funkcionira na način da robot pretražuje svoj zapis posjećenih zidova, te ukoliko je trenutno orijentiran direktno prema jednom takvom zidu evaluira prvo podstablo, a u suprotnom slučaju drugo.

Korištenje „IMEM2“ funkcije zahtjeva od robota pamćenje svih akcija i koordinata na kojima se u tom trenutku nalazio. „IMEM2“ čvor prima tri argumenta. Prvo ispituje postoji li za trenutnu poziciju akcija prvog podstabla u memoriji, te ukoliko postoji, evaluira drugo podstablo, a u suprotnom, treće.

5. Rezultati ispitivanja

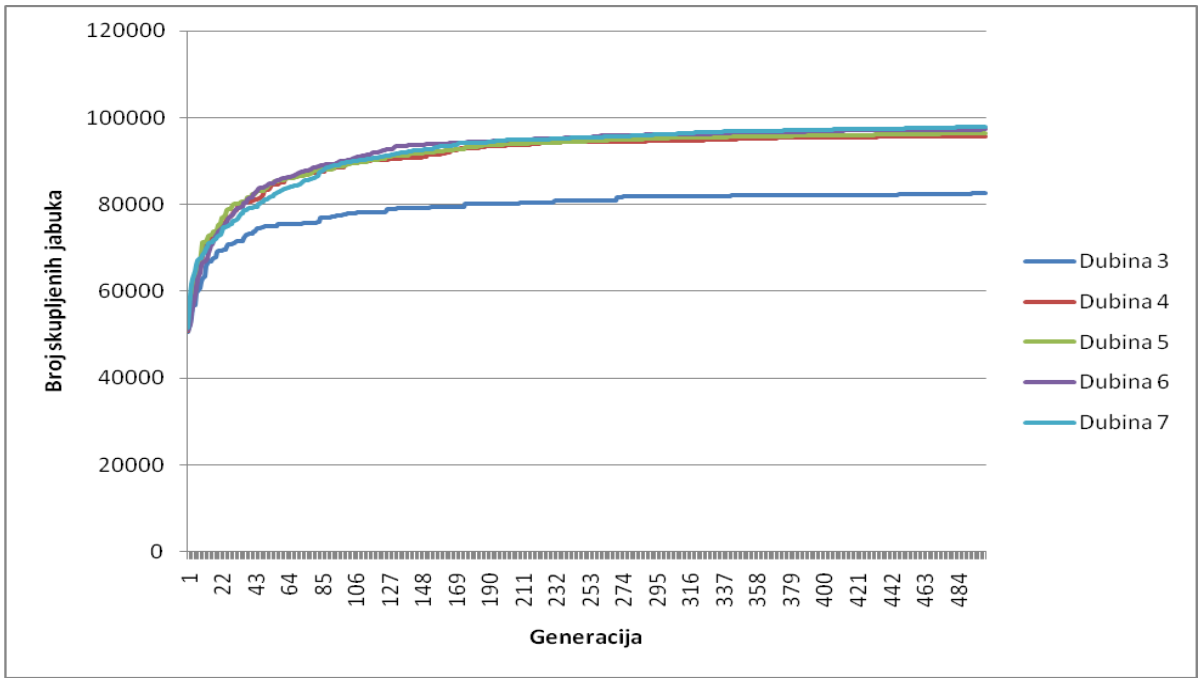
Ovaj dio rada posvećen je ispitivanju različitih aspekata GP.

5.1. Utjecaj dubine generiranih stabala

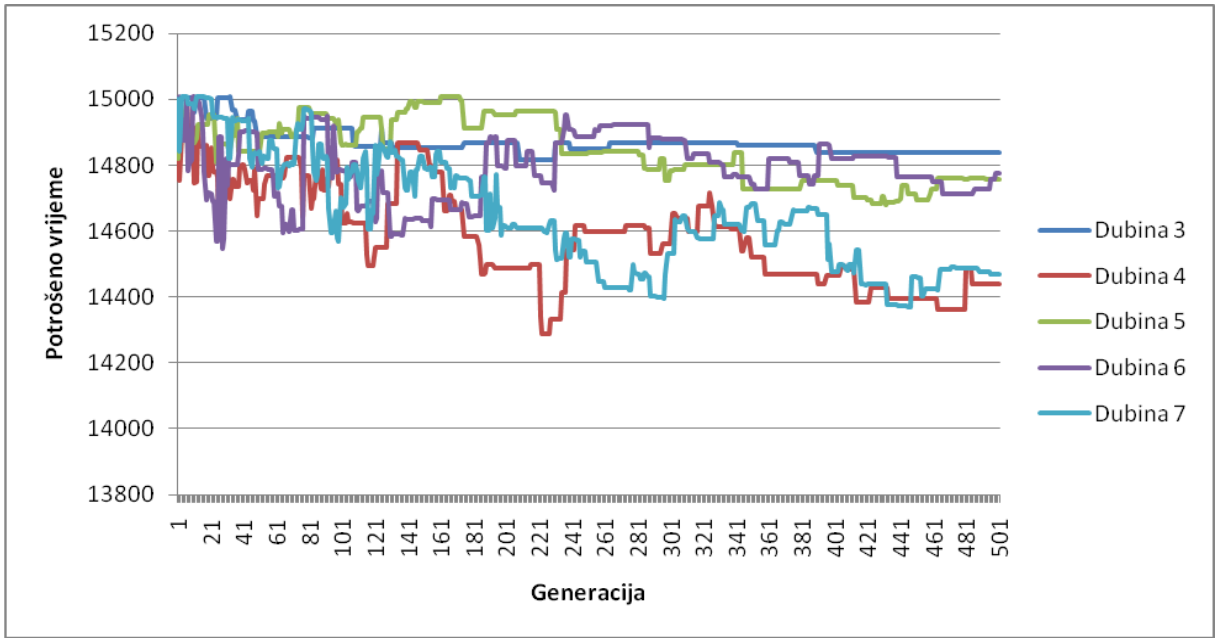
Pitanje na koje želimo odgovoriti ovim pokusom glasi: kakav je utjecaj veličine programa generiranih u inicijalnoj populaciji na kvalitetu postignutih rješenja pri učenju na više soba. Važno je primijetiti da programi mogu tokom evolucijskog procesa, zbog operatora križanja, postići veću veličinu nego što je to zadano ovim parametrom, međutim, utvrđeno je da veličina konačnih rješenja uvelike ovisi o veličinama korištenim u inicijalnoj populaciji. Odgovor na ovo pitanje bitan je i za neka daljnja ispitivanja, jer bi neoprezan izbor parametra dubine mogao znatno utjecati na rezultate. Naime, premala dubina stabla onemogućuje izražavanje kompleksnih izraza, dok bi prevelika dubina stabla rezultirala puno većim troškovima komunikacije, većom potrošnjom memorije, te ponešto usporenijim radom algoritma. Također, s obzirom na eksponencijalni rast, problem bi mogao uzrokovati i preveliki prostor pretraživanja. Maksimalni stupanj funkcijskih čvorova je 4, pa na primjer na dubini 3 možemo imati ukupno 4^3=64 listova, a budući da postoji 19 različitih elemenata u skupu završnih znakova, to znači da je moguće prikazati 19^{64} različitih programa. Taj broj je u stvarnosti još i veći, jer ovaj izračun pokriva samo savršeno balansirana stabla, u kojima su svi unutarnji čvorovi „IFLTE“. Za usporedbu, na dubini 6, primjerice, moguće je prikazati ukupno 19^{4096} različitih programa (s istim ograničenjem kao i u prethodnom primjeru).

Budući da je cilj ovog ispitivanja bio odrediti utjecaj zadane dubine na najbolja pronađena rješenja, za ispitivanje je izabran vrlo težak problem. Konkretno, izabrano je učenje na 10 ručno izabranih soba (izabranih po kriteriju procijenjene raznolikosti). Svaki pokus je pokretan kroz 500 generacija, evoluirajući populaciju od 1000 jedinki. Ispitivane su dubine od 3 do 7, te je za svaku dubinu pokus pokretan 10 puta. Prosječne vrijednosti pokusa prikazane su slikama 5.1 (broj skupljenih jabuka) i 5.2 (potrošeno vrijeme). Tablica 5.1 prikazuje vrijednosti u 500. generaciji. Za ovaj pokus je korištena normalizacija, odnosno za svaku sobu je broj maksimalnih jabuka skaliran na 10000. Normalizacija se koristi kako bi se ujednačila važnost svih soba, odnosno kako se nebi događalo da je soba sa više zidova „vrjednija“ za robota. Budući da se pokus pokretao na 10 soba, a svaka soba može dati

maksimalni rezultat od 10000, maksimalni mogući broj skupljenih jabuka u tom slučaju iznosi 100000, međutim u stvarnosti je taj broj nešto manji, jer su neki zidovi nedostupni.



Slika 5.1: Broj skupljenih jabuka najboljih jedinki kroz generacije, ovisno o dubini inicijalno generiranih stabala.



Slika 5.2: Potrošeno vrijeme na obilazak sobe

Dubina	3	4	5	6	7
Broj skupljenih jabuka	82499.6	95757.2	96512.8	97275.9	97778.2
Potrošeno vrijeme	14838.4	14439	14758.6	14776.2	14468

Tablica 5.1 Prosječne vrijednosti u 500. generaciji

Rezultati pokazuju da dok god je izabrana dubina veća od 3, mogu se očekivati zadovoljavajuća rješenja. Programi s inicijalnom dubinom 6 i 7 uspjeli su pronaći blago uspješnije jedinke nego programi dubine 4, ali razlog tome mogla bi biti mogućnost prikazivanja specijaliziranijih rješenja, koja pokušavamo izbjeći. Također, valja uzeti u obzir i veličinu konačno dobivenih rješenja, prosječna veličina najboljih jedinki iz ovog ispitivanja za dubinu 4 iznosi 494 bajta, dok je prosječna veličina za dubinu 7 jednaka 6.64KB. Iz navedenih razloga je za daljnja ispitivanja odabrana dubina 4.

5.2. Utjecaj memorijskih funkcija

U ovom dijelu ispitat ćemo imaju li implementirane memorijske funkcije „IMEM1“ i „IMEM2“ ikakvog utjecaja na rad algoritma. Za ovo ispitivanje izabran je identičan problem kao i u prošlom zadatku, budući da se želi ispitati isti utjecaj. Izabrana dubina početno generiranih jedinki je 4. Ispituju se 4 načina rada s obzirom na korištene memorijske funkcije:

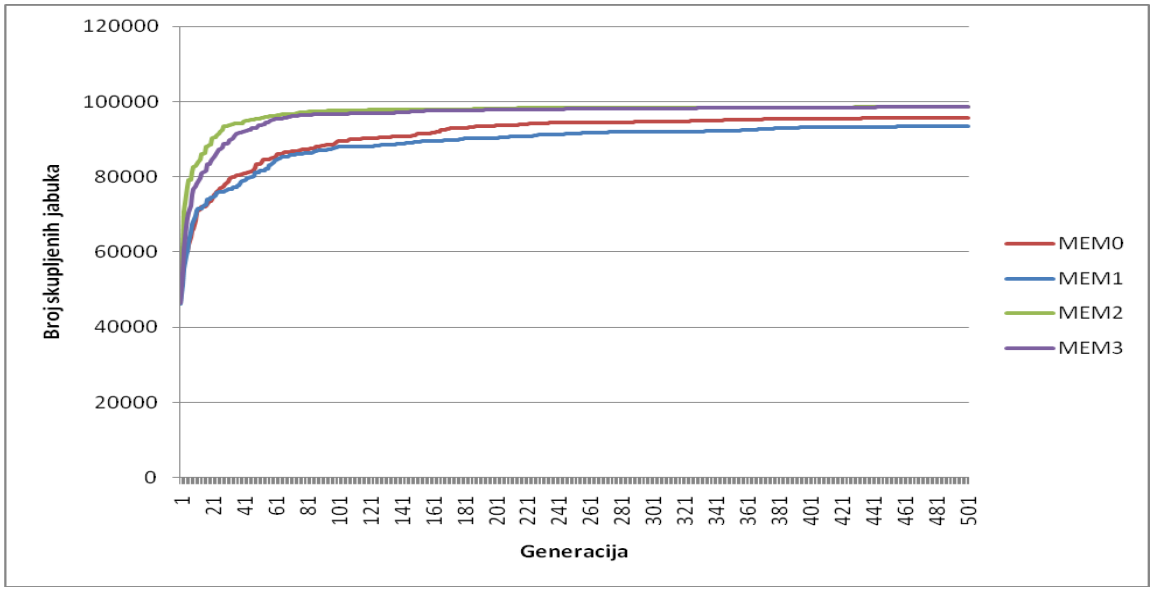
1. MEM0 - ne koristi niti jednu memorijsku funkciju; odgovara pokusu za dubinu 4 iz prethodnog ispitivanja (poglavlje 5.1)

2. MEM1 - uz sve standardne funkcije koristi još i „IMEM1“

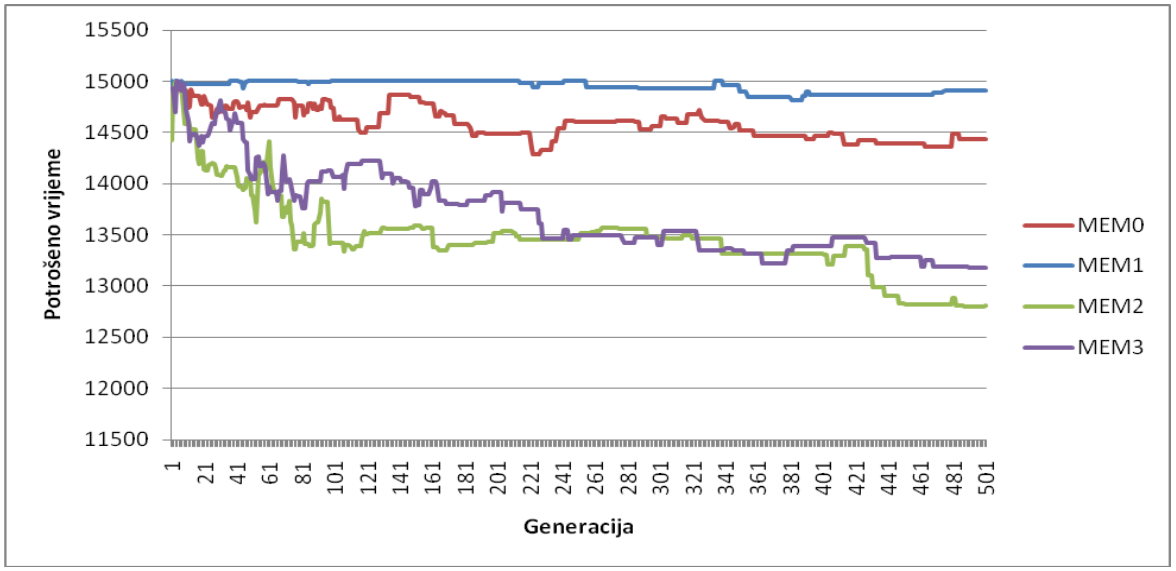
3. MEM2 - sve standardne funkcije plus „IMEM2“

4. MEM3 – sve standardne funkcije plus „IMEM1“ i „IMEM2“

Ispitivanje za svaki od navedena 4 načina rada je pokretano 10 puta te su uzete prosječne vrijednosti koje su prikazane slikama 5.3 i 5.4. Tablica 5.2 prikazuje prosječne vrijednosti zadnje generacije.



Slika 5.3 Prosječan broj skupljenih jabuka najboljih jedinki kroz generacije, ovisno o korištenom memorijskom modu



Slika 5.4 Prosječno trajanje obilaska soba najboljih jedinki kroz generacije, ovisno o korištenom memorijskom modu

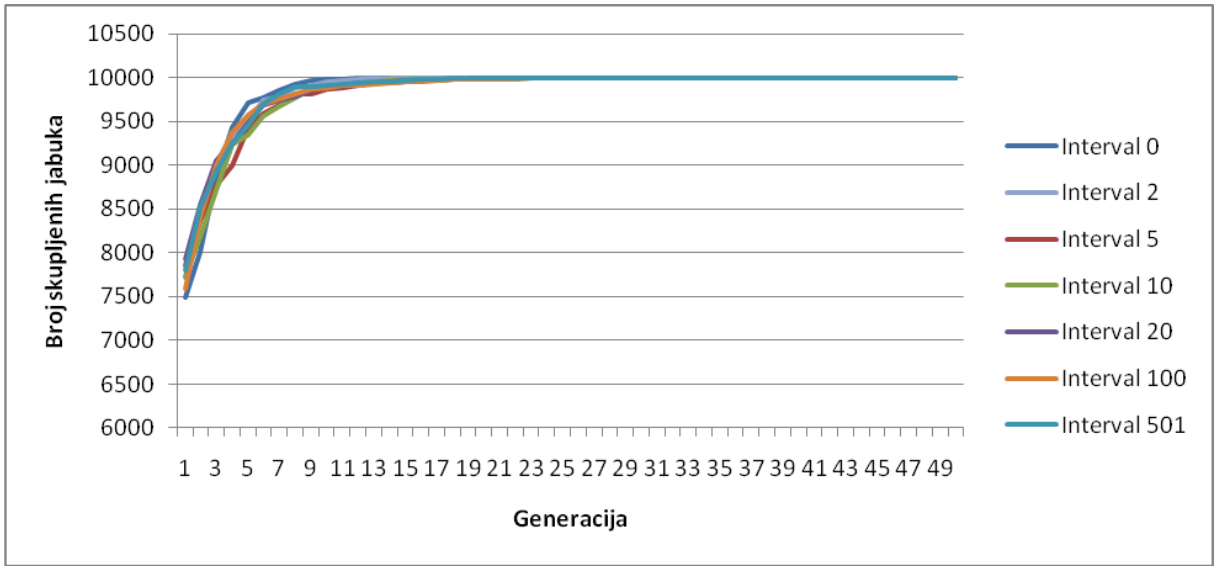
Način rada	MEM0	MEM1	MEM2	MEM3
Broj skupljenih jabuka	95757.2	93537.9	98681	98578.6
Potrošeno vrijeme	14439	14906.9	12810.6	13178.9

Tablica 5.2 Prosječne vrijednosti u 500. generaciji

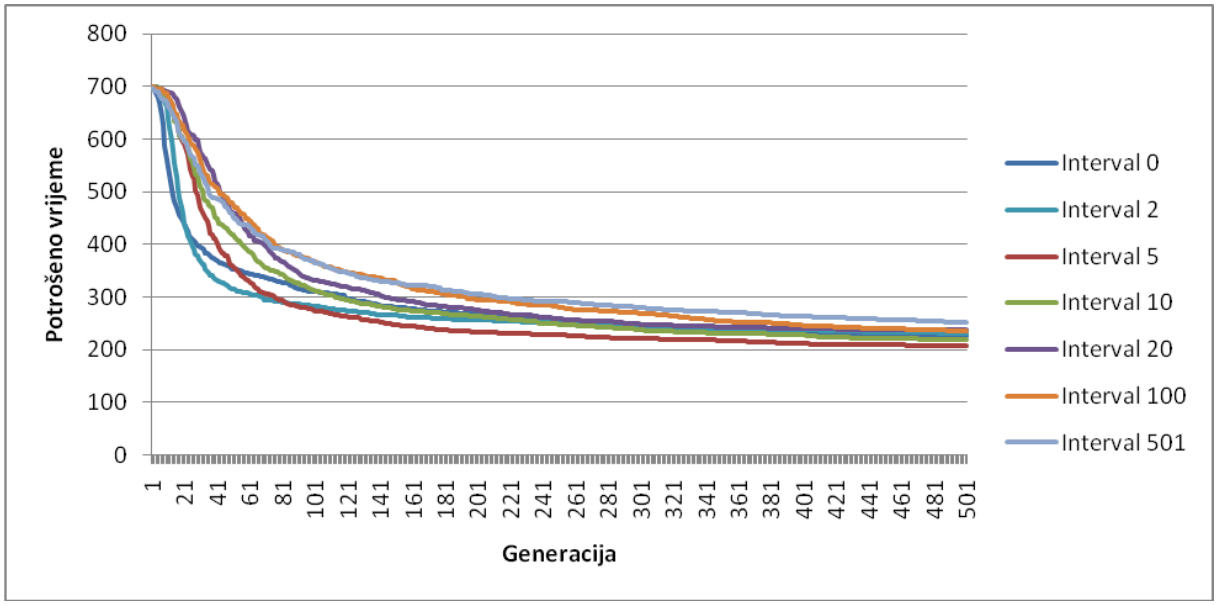
Rezultati pokazuju da funkcija „IMEM2“ predstavlja vrlo koristan dodatak skupu funkcijskih znakova. Njeno korištenje znatno utječe i na brzinu konvergencije, i na kvalitetu pronađenih rješenja. Zanimljivo za primijetiti jest da niti jednim pokretanjem iz prošlog ispitivanja, za bilo koju dubinu stabla, nije postignuto vrijeme izvođenja kao što se postiže koristeći „IMEM2“. S druge strane, funkcija „IMEM1“ pokazala se čak kao blago štetan dodatak s nepovoljnim utjecajem i na konvergenciju i na kvalitetu rješenja. Razlog zbog kojeg evolucija s uključenim „IMEM1“ daje nešto slabije rezultate je najvjerojatnije povećavanje prostora pretraživanja.

5.3. Utjecaj migracijskog intervala

Migracijski interval je jedan od ključnih parametara više-populacijskih GA. Osim što uvelike određuje troškove komuniciranja, očekuje se da ima i znatan pozitivan utjecaj na brzinu konvergencije. Ono što nije u potpunosti jasno bez ispitivanja jest ima li ta brza konvergencija pozitivan ili negativan utjecaj na kvalitetu pronađenih rješenja. Naime, ono što bi se moglo dogodit jest da određeni bitni dijelovi lošijih jedinki ne prežive dovoljno dugo da bi imali ikakvog utjecaja, jer bivaju brzo zamjenjeni naizgled boljim jedinkama. Drugim riječima, kraći migracijski interval mogao bi imati štetan utjecaj na raznolikost genetskog materijala jedinki. Ispitivanje je provedeno koristeći 8 otoka povezanih u topologiju oblika dvostruko usmjerenog prstena. Koristio se način rada u kojem se od svih dostupnih susjeda istovremeno traže jedinke. Svaki otok sadržavao je 400 jedinki te se evolucija pokretala kroz 500 generacija. Izabrana soba za ispitivanje je istog oblika kao ona na kojoj je John R. Koza ispitivao svoj GP. Ispitivani migracijski intervali su: 0 (nakon svake generacije se traže jedinke, bez obzira je li došlo do promjene najbolje jedinke), 2, 5, 10, 20, 100 i 501 (otoci uopće ne komuniciraju). Za svaki interval provedeno je 50 pokusa. Slike 5.5 i 5.6 pokazuju prosječan broj sakupljenih jabuka te prosječno potrošeno vrijeme.



Slike 5.5 Prosječan broj sakupljenih jabuka kroz prvih 50 generacija



Slika 5.6 Prosječno potrošeno vrijeme kroz generacije

Rezultati potvrđuju očekivanja u vezi brzine konvergencije, što se zbog izbora relativno lakog problema bolje vidi na prosječno potrošenim vremenima nego na broju sakupljenih jabuka. Tablica 5.3 prikazuje prosječno potrošeno vrijeme u 500. generaciji.

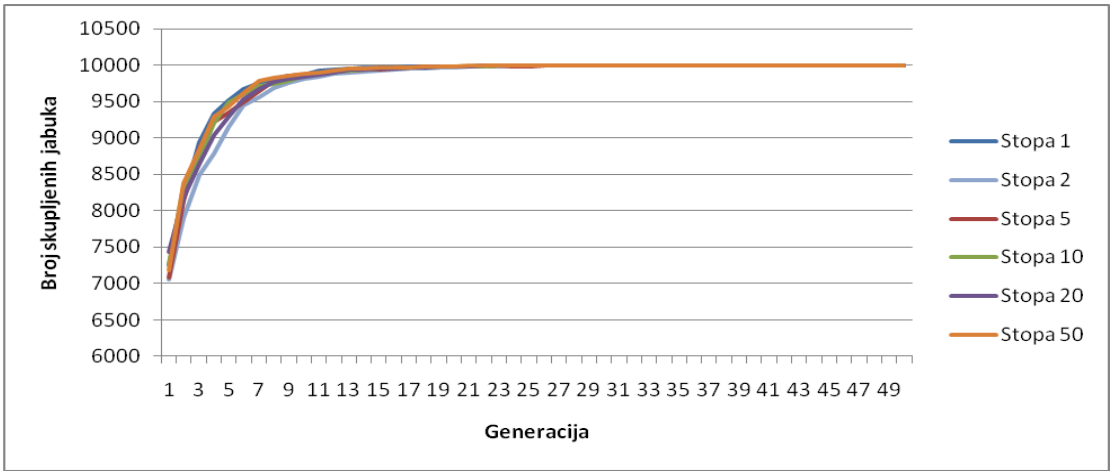
Interval	0	2	5	10	20	100	501
Vrijeme	227.2	231.9	208.2	219.6	237.5	236.1	252.1

Tablica 5.3 prosječno potrošeno vrijeme na obilazak sobe najboljih jedinki u 500. Generaciji

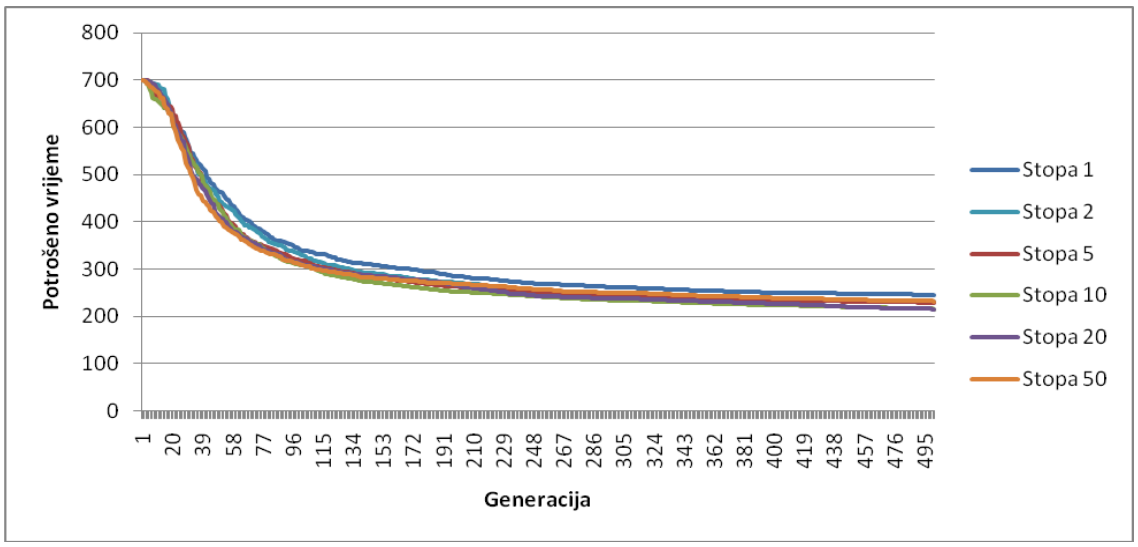
Iz tablice i slike se može također vidjeti i da znatno kratki intervali imaju donekle negativan utjecaj na kvalitetu konačnog rješenja.

5.4. Utjecaj migracijske stope

Migracijska stopa predstavlja drugi vrlo važan parametar za troškove komuniciranja u višepopulacijskim GA. Pitanje na koje ovim ispitivanjem želimo pronaći odgovor je: koliko bi niska stopa migracije smjela biti, a da nema znatnih negativnih utjecaja na rad algoritma. Prijašnja istraživanja su potvrdila da kvaliteta rješenja raste sa povećanjem migracijske stope [9]. Za ispitivanje ovog parametra izabran je isti problem kao u ispitivanju migracijskog intervala, s tim da je interval fiksiran na 10 generacija. Stope migracije koje se ispituju, izražene u postotcima inicijalne veličine otoka, su redom: 1,2, 5, 10, 20 i 50%. Ispitivanje za svaku dubinu provedeno je 50 puta. Prosječan broj sakupljenih jabuka te prosječno potrošeno vrijeme dano je slikama 5.7 i 5.8., a tablica 5.4 prikazuje prosječno potrošeno vrijeme u 500. generaciji.



Slika 5.7 Prosječan broj sakupljenih jabuka kroz prvih 50 generacija



Slika 5.8 Prosječno potrošeno vrijeme najboljih jedinki

Stopa	1	2	5	10	20	50
Vrijeme	245.9	230.6	229.8	215.4	215.5	232.2

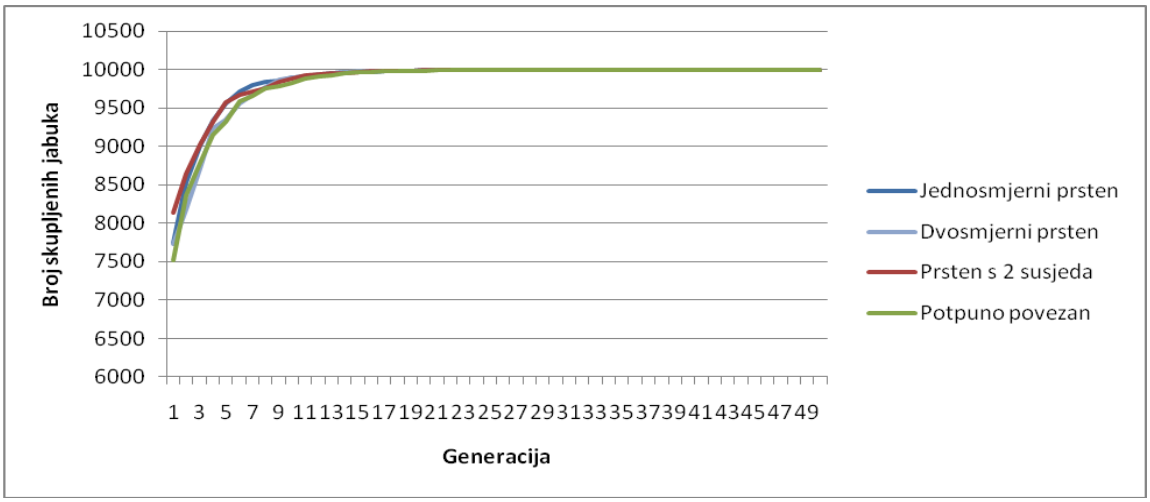
Tablica 5.4 Prosječno potrošeno vrijeme na obilazak sobe najboljih jedinki u 500. generaciji

Dobiveni rezultati potvrđuju da kvaliteta rješenja raste sa migracijskom stopom, kao što je i očekivano. Važno je primjetiti da se rast kvalitete za stopu od 50% niti nije očekivao, jer stopa od 50% podrazumjeva uništavanje pola postojeće populacije otoka pri svakom primanju jedinki.

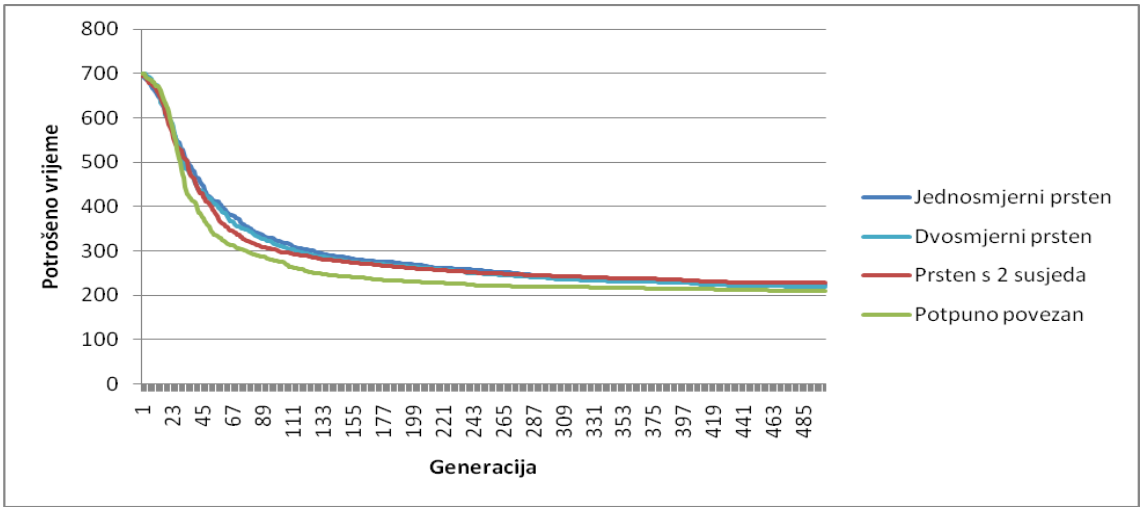
5.5. Utjecaj topologije

U ovom djelu ispitivat ćemo razne topologije kako bismo odredili njihov utjecaj na kvalitetu pronađenih rješenja, te konvergenciju. Zaključak drugih istraživača [9] je da generalno kvalitete rješenja rastu s povećanjem stupnja njihovih čvorova, te da se različite topologije s istim stupnjem čvorova međusobno ne razlikuju. Za ovo ispitivanje izabrani su isti parametri kao u protekla 2 ispitivanja, s fiksiranom vrijednosti stope slanja na 10 posto i

migracijskog intervala na 10 generacija. Slike 5.9 i 5.10 prikazuju dobivene rezultate. Tablica 5.5 prikazuje prosječno potrošeno vrijeme najboljih jedinki u zadnjoj generaciji.



Slika 5.9 Prosječan broj skupljenih jabuka kroz prvih 50 generacija



Slika 5.10 Prosječno vrijeme obilaska

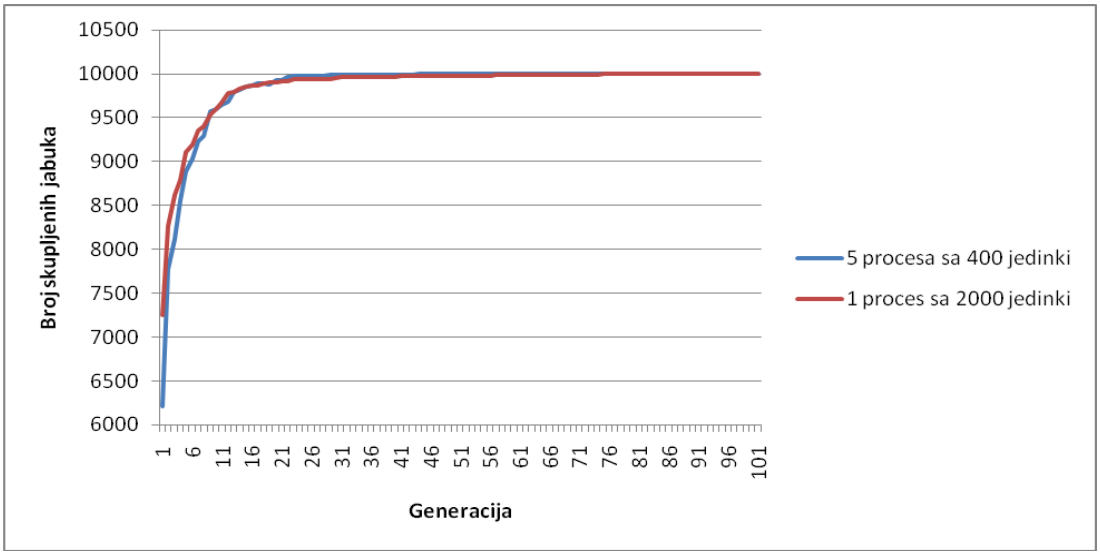
Topologija	Prosječno potrošeno vrijeme najbolje jedinke
Jednosmjerni prsten	225
Dvosmjerni prsten	219.6
Prsten s 2 susjeda	227.7
Potpuno povezan	210.4

Tablica 5.5 Prosječno potrošeno vrijeme na obilazak sobe najboljih jedinki u 500. Generaciji

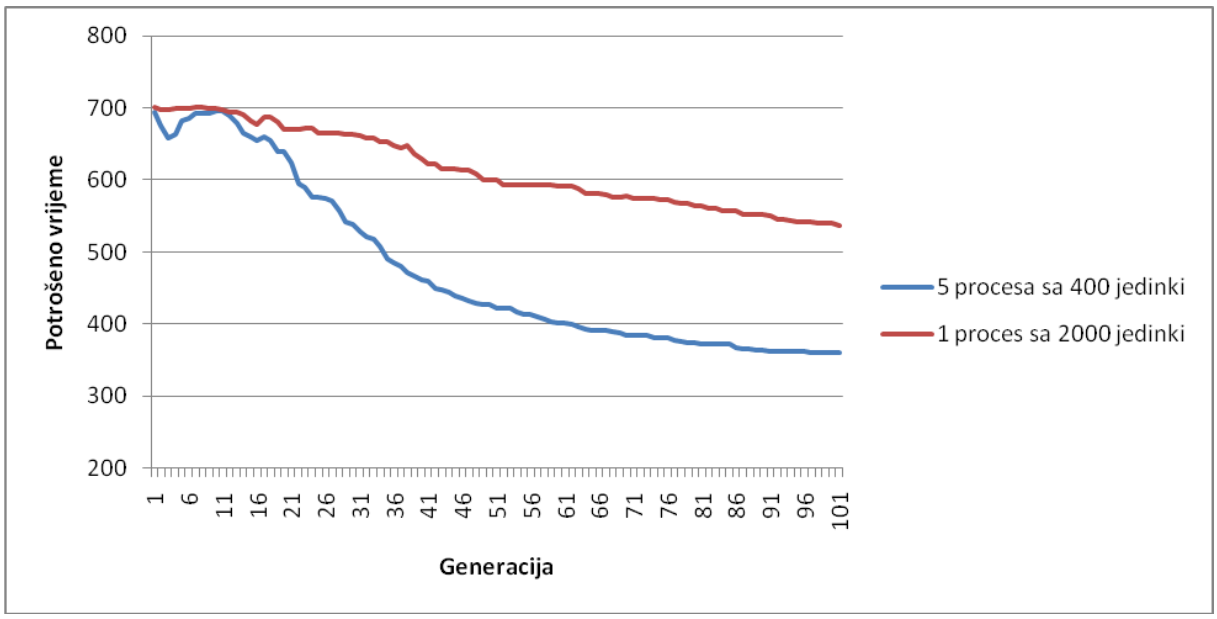
Rezultati jasno pokazuju da povećanje stupnja povezanosti pozitivno utječe na brzinu konvergencije. Također, primjećuje se i razlika u najboljem rješenju kod potpuno povezane strukture, koja jedina ima znatno veći stupanj povezanosti čvorova od drugih ispitanih struktura.

5.6. Usporedba jedno-populacijskog i više-populacijskog GP

U ovom dijelu želimo ispitati kakav utjecaj na kvalitetu rješenja ima korištenje jednakog broja jedinki razdijeljenog na otoke u usporedbi s postojanjem jedne centralne populacije, odnosno koja je metoda korisnija za jednaki broj računanja funkcije dobrote. Uspoređivat će se učenje na jednom otoku sa 2000 jedinki, s 5 otoka veličine 400. Za interval komunikacija kod više-populacijskog GP izabrano je 5 generacija, za količinu prijenosa 10 posto veličine otoka, a za topologiju potuno povezan graf. Slike 5.11 i 5.12, te tablica 5.6 prikazuju rezultate.



Slike 5.11 Broj sakupljenih jabuka



Slike 5.12 Prosječno potrošeno vrijeme na obilazak sobe

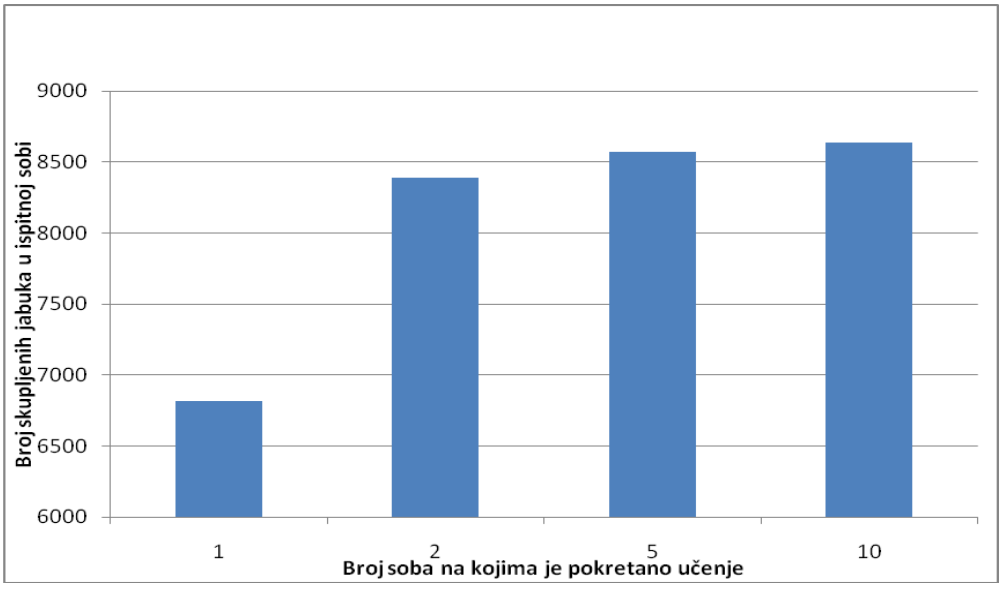
	5 procesa sa 400 jedinki	1 proces sa 2000 jedinki
Broj skupljenih jabuka	9997.9	9994.6
Potrošeno vrijeme	359.6	537

Tablica 5.6 Prosječno potrošeno vrijeme na obilazak sobe najboljih jedinki u 500. Generaciji

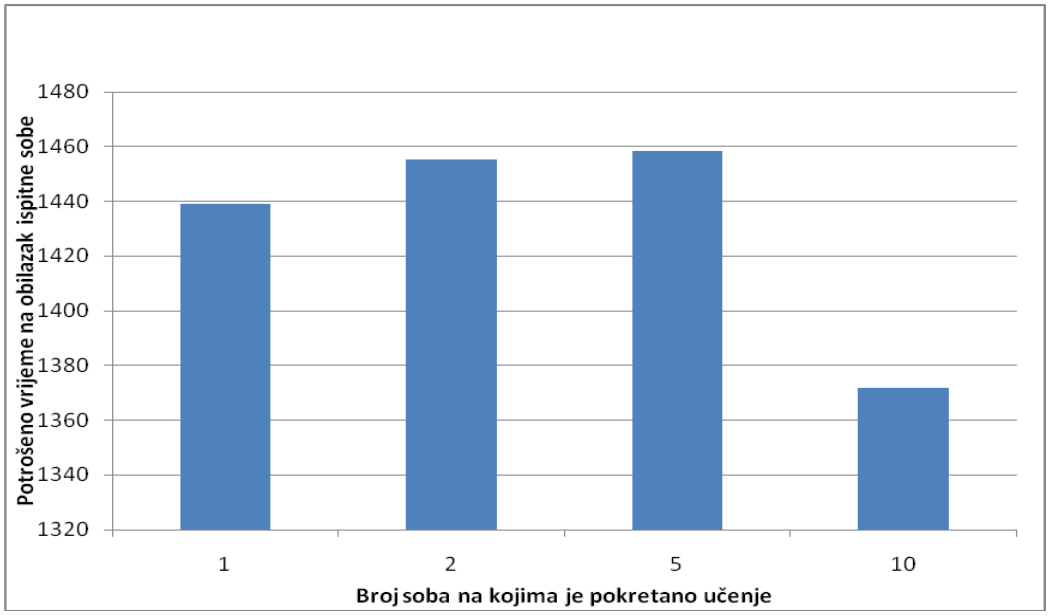
Rezultati potvrđuju nalaze [3] koji govore da više-populacijski GA/GP nalaze bolja rješenja od ekvivalentnih jedno-populacijskih metoda.

5.7. Utjecaj učenja na više soba

Učenje na više soba pokazalo se kao znatno teži problem od učenja na jednoj sobi, pitanje koje želimo odgovoriti u ovom dijelu je ima li ta dodatna cijena opravdanje u sposobnosti generaliziranja robota. Odnosno, ukoliko robot uči ponašanje na nekih 10 soba, hoće li imati bolje šanse dobro se ponašati na nekoj novoj, jedanaestoj sobi, nego da je učio na samo jednoj sobi. Ispitivanje je provođeno na sljedeći način: definirano je 60 soba koje se mogu koristiti. Od tih 60 soba nasumično se izvlači M soba za učenje, gdje je M promatrani parametar, i jedna dodatna za ispitivanje nakon što program prođe 100 generacija. Ispitivano je učenje na 1, 2, 5 i 10 soba. Slike 5.13 i 5.14 prikazuju rezultate.



Slika 5.13 Prosječan broj sakupljenih jabuka u ispitnoj sobi, ovisno o broju soba na kojima je robot učio

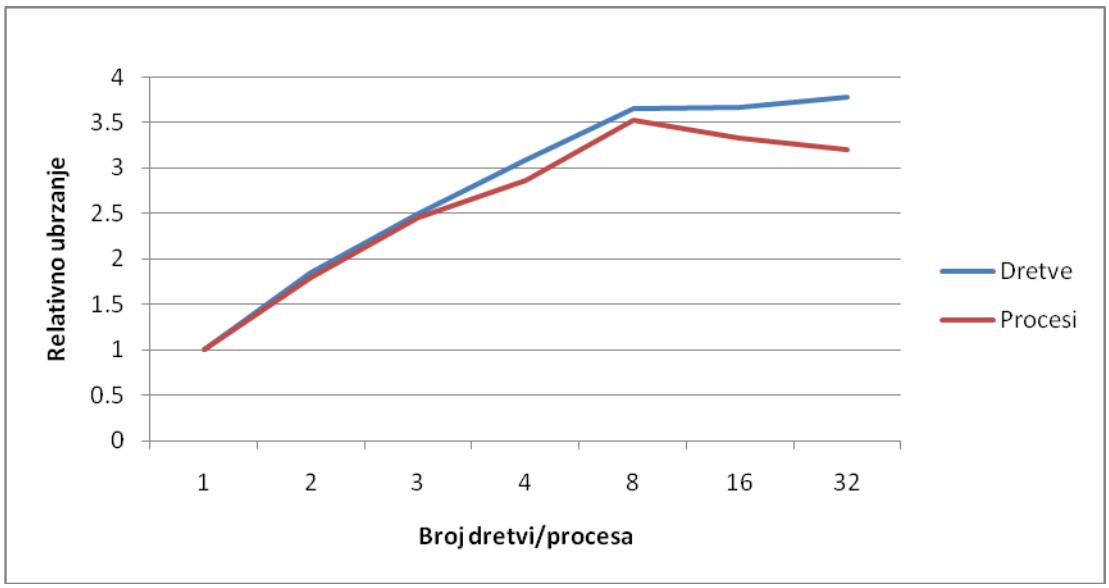


Slika 5.14 Prosječno vrijeme obilaska ispitne sobe

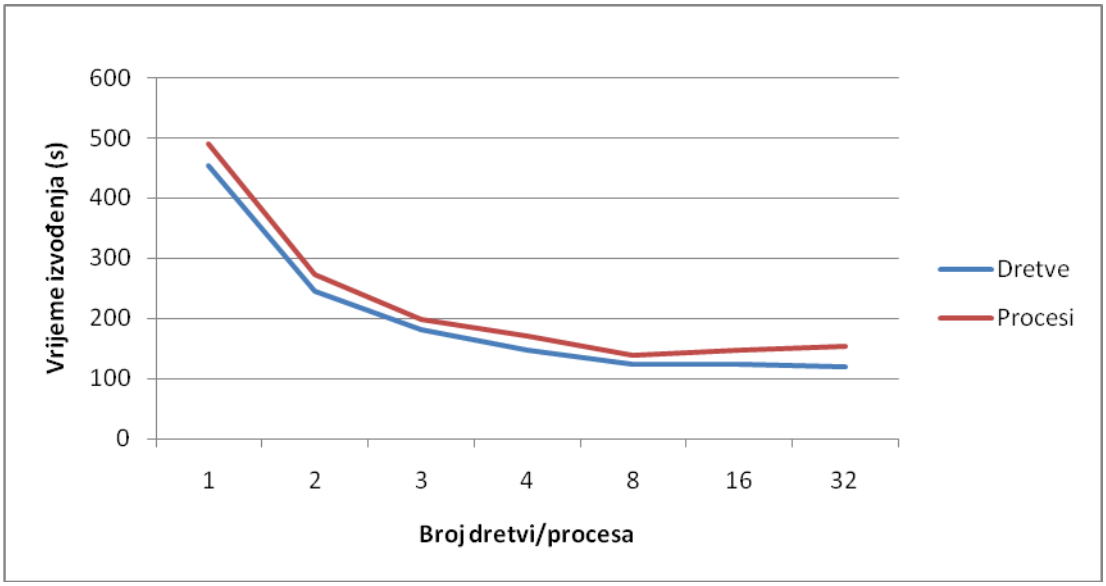
Rezultati jasno pokazuju da su roboti učeni na više soba naučili generaliziranije ponašanje, odnosno da uspijevaju obići rubove u više slučajeva nego roboti učeni na manje soba.

5.8. Usporedba paralelnih implementacija jedno-populacijskog GP

U ovom poglavlju ispitujemo postignuto ubrzanje paralelizacijom jednopopulacijskog GP. Uspoređuju se dvije implementirane verzije: verzija koja paralelizira i genetske operatore i računanje funkcije dobrote, temeljena na dretvama, te verzija koja paralelizira samo računanje funkcije dobrote, temeljena na komuniciranju pomoću MPI biblioteke. Za ispitivanje je izabrano učenje na 10 soba, kako bi se maksimizirao trošak računanja funkcije dobrote. Slike 5.15 i 5.16 prikazuju relativna ubrzanja i prosječno vrijeme izvođenja zadatka. Ispitivanje je rađeno na računalu sa i7 860 procesorom sa 4 jezgre. Izabrana veličina populacije je 1000, a broj generacija 100. Namjerno je drastično smanjen broj „Max time“, odnosno maksimalan broj motornih akcija koje robot smije napraviti, zato da se osigura što veća sličnost kompleksnosti izvođenja dviju jedinki, jer se bolje evoluirane jedinke koje uspjevaju sakupiti jabuke puno prije isteka roka izvode znatno brže.



Slika 5.15 Relativno ubrzanje

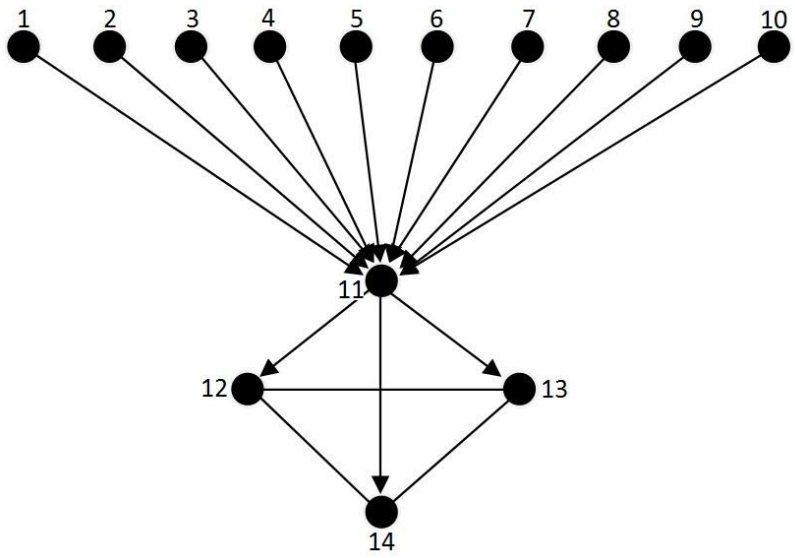


Slika 5.16 Vrijeme izvođenja u sekundama

Rezultati pokazuju da obje implementacije uspijevaju dosta učinkovito ubrzati rad algoritma. U slučaju implementacije sa dretvama to ubrzanje je nešto izraženije te postiže maksimalnu vrijednost od 3.77 za 32 dretve. Valja uzeti u obzir da računalo na kojem su pokretana ispitivanja ima samo 4 jezgre, te da ipak mora trošiti dio resursa na pokretanje operacijskog sustava. Maksimalno postignuto ubrzanje koristeći MPI biblioteku za komuniciranje je 3.52 za 8 procesa. Ono što je također bitno za primijetiti je znatno bolja skalabilnost rješenja s dretvama. Opažamo da korištenjem više od 8 procesa performanse počinju slabiti, dok se kod rezultata za implementaciju s dretvama vidi da je dodatni trošak računanja zbog broja dretvi zanemariv.

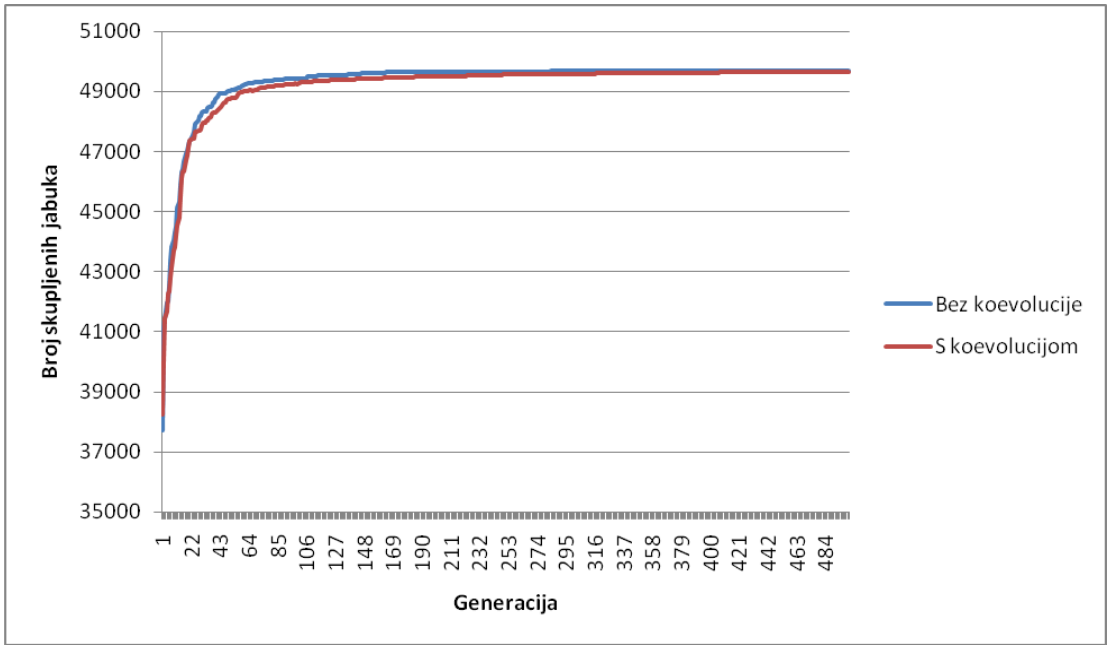
5.9. Utjecaj koevolucije

U ovom djelu ispitujemo utjecaj odvojene evolucije specijaliziranijih jedinki na kvalitetu pronađenih rješenja. Pitanje na koje želimo pronaći odgovor je: isplati li se kod problema učenja na više soba za neke procese odrediti učenje na pojedinačnim sobama. Uspoređivat će se učenje 6 otoka od kojih svaki sadrži 400 jedinki u strukturi potpuno povezanog grafa bez korištenja koevolucije, sa odabranom topologijom prikazanom slikom 5.17 uz korištenje koevolucije. U slučaju potpuno povezane topologije koristila se migracijska stopa od 10% i migracijski interval od 10 generacija bez promjene najbolje jedinke. Za ispitivanje sa koevolucijom je izabrana veća migracijska stopa od 20% zbog velikog broja procesa koji istovremeno šalju jedinke procesu 11.

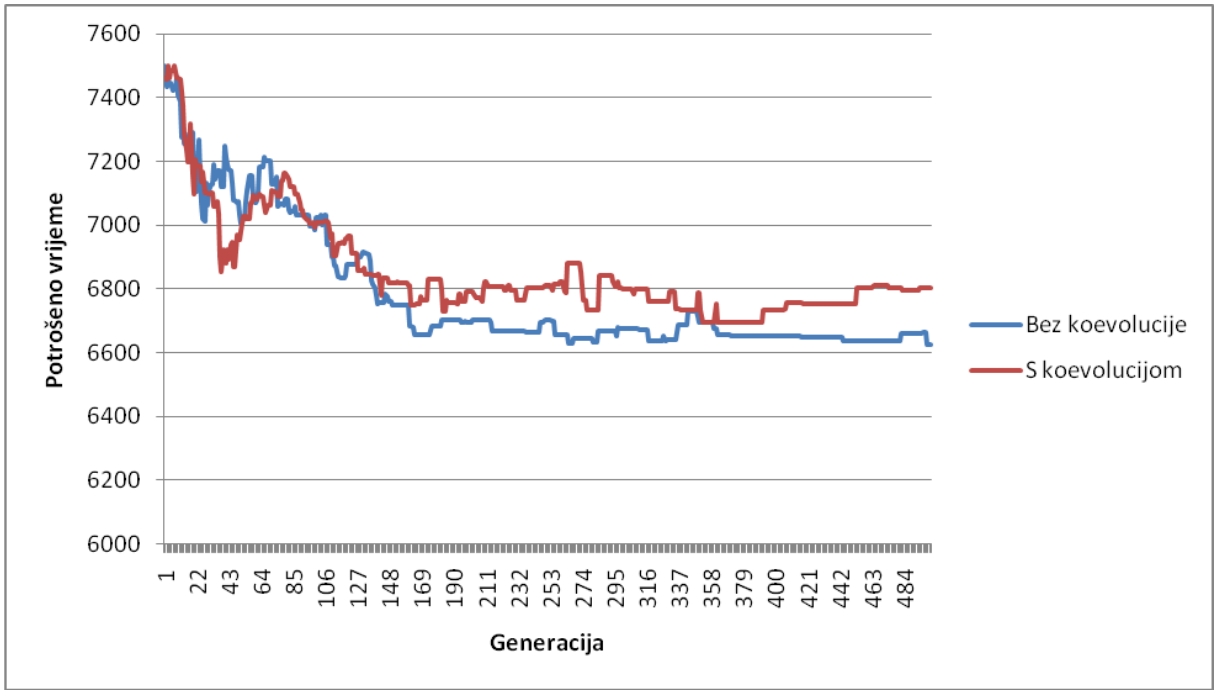


Slika 5.17 Korištena topologija za ispitivanje

Procesima označenim sa 1-10 dodijeljena je po jedna od 5 soba na kojima se uči u procesima 11-14. Preciznije, svaka od 5 soba je dodijeljena na točno 2 procesa u skupu procesa od 1 do 10 i svim procesima u skupu 11-14. Na ovaj način uspoređuju se učenja koja sadržavaju jednak broj evaluacija funkcije dobrote. Slike 5.18 i 5.19 te tablica 5.7 prikazuju dobivene rezultate.



Slika 5.18 Prosječan broj skupljenih jabuka najboljih jedinki kroz generacije



Slika 5.19 Prosječno potrošeno vrijeme

	Broj skupljenih jabuka	Potrošeno vrijeme
Bez koevolucije	49694.7	6625.1
S koevolucijom	49657.2	6804.2

Tablica 5.7 Prosječan broj skupljenih jabuka i prosječno potrošeno vrijeme najboljih jedinki u 500. Generaciji

Rezultati pokazuju da koevolucija nije isplativa u ovom slučaju, odnosno da se bolja rješenja dobivaju ukoliko ne specijaliziramo neke procese. Najvjerojatniji razlog je slaba dobrota primljenih specijaliziranih jedinki na druge sobe.

6. Zaključak

Genetsko programiranje predstavlja jak i fleksibilan način evoluiranja automatskog upravljanja robotom u okolini. Paralelno genetsko programiranje predstavlja vrlo korisno i učinkovito poboljšanje slijednog algoritma. Više-populacijski GP uspješnije rješavaju zadani problem, nego slijedni jedno-populacijski GP jednake veličine. Paralelizacijom jedno-populacijskog GP postignuta su zadovoljavajuća ubrzanja. Korištenje memorijske funkcije koja upravlja tijekom izvođenja programa ovisno o zapamćenim akcijama robota za danu poziciju pokazalo se kao vrlo koristan dodatak skupu funkcija. Pokazano je da je učenje na više soba znatno teži problem, nego učenje na jednoj sobi, što može biti vrlo korisno ukoliko se dani problem želi koristiti za ispitivanje raznih aspekata GP. Također, potvrđeno je da se učenjem na više soba dolazi do upravljačkih programa koji u prosjeku mogu bolje obilaziti neviđene sobe. Potvrđeno je da kvaliteta rješenja raste sa stupnjem povezanosti otoka. Također je potvrđeno da smanjenje intervala znatno utječe i na brzinu konvergencije i na kvalitetu pronađenih rješenja.

Literatura

[1] Wikipedia, the free encyclopedia, Natural selection, s interneta, http://en.wikipedia.org/wiki/Natural_selection, datum pristupa: 24.april 2011.

[2] Jakobović, D.: Predavanja iz kolegija Paralelno programiranje

[3] Alba, E., Troya, J.M.: A Survey of Parallel Distributed Genetic Algorithms, Complexity, vol. 4, no. 4, 1999.

[4] Paić-Antunović, L: Automatsko upravljanje u simuliranoj okolini

[5] Cantú-Paz, E.:Efficient and accurate parallel genetic allgorithms, Kluwer Academic Publishers, Massachusetts, 2001.

[6] Golub, M., Budin, L.: An Asynshcronous Model of Global Parallel Genertic Algorithms, Second ICSC Symposium on Engineering of Intelligent Systems EIS 2000, University of Paisley, Scotland, UK, 2000.

[7] Bilješke s predavanja, paralelno programiranje

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

[9] Cantú-Paz, E.: Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms, In Banzhaf et al., editors, Proceedings of the Genetic and Evolutionary Computation Conference, volume 1, pages 91-98.

[10] Danoy, G, Bouvry, P, Alba, E.: Distributed Coevolutionary Genetic Algorithm for Optimal Design of Ad Hoc Injection Networks, In special session on Parallel and Grid Computing. Pragues, June, 2007.