SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1883

Ocjena učinkovitosti asinkronih paralelnih evolucijskih algoritama

Bruno Alfirević

Zagreb, veljača 2011.

Sažetak

Ovaj diplomski rad opisuje različite modele paralelizacije genetskih algoritama. Istražena je primjena genetskog programiranja na razvijanje automatizirane računalne strategije za igranje kartaške igre Ajnc i provedena analiza učinkovitosti različitih strategija paralelizacije na istom problemu.

Ključne riječi

evolucijsko računanje, genetski algoritmi, genetsko programiranje, paralelno računanje, strojno učenje, Ajnc

Abstract

This diploma thesis describes various models for parallelization of genetic algorithms. Application of genetic programming to evolving automated computer Blackjack player is investigated. Thesis presents analysis of efficiency for several parallelization strategies applied to the same problem.

Keywords

evolutionary computing, genetic algorithms, genetic programming, parallel computing, machine learning, Blackjack

Sadržaj

1. Uvod

Ovaj diplomski rad se temelji na istraživanju učinkovitosti različitih modela paralelizacije evolucijskih algoritama, s konkretnom primjenom u području strojnog učenja. Postoji nekoliko pristupa paralelizaciji evolucijskih algoritama, sa različitim prednostima i nedostacima, kao što su stupanj ubrzanja pri povećanju broja procesora na kojima se algoritam izvodi, lakoća programskog ostvarenja paralelnog algoritma te prilagođenost pojedinim vrstama problema. U ovom radu je opisana primjena evolucijskih algoritama na razvijanje automatizirane računalne strategije za igranje popularne kartaške igre Ajnc. Obavljeno je eksperimentiranje sa različitim paralelnim algoritmima i parametrima paralelizacije, te varijacijama u rješenju samog problema evoluiranja strategija igranja Ajnca.

Rad se sastoji od osam poglavlja. Nakon uvodnog, slijedi poglavlje o evolucijskim algoritmima. Kratko je opisana njihova inspiracija biološkom evolucijom, nakon čega slijedi pregled načina rada genetskih algoritama, metodologija njihovog korištenja na stvarnim problemima, te opis alata kojima se postiže evolucija rješenja problema prema onom koje nas zadovoljava.

Treće poglavlje se bavi opisom paralelnih evolucijskih algoritama i različitih modela njihove paralelizacije. Opisane su karakteristike različitih inačica paralelnih evolucijskih algoritama i njihova primjena u različitim situacijama.

Četvrto poglavlje opisuje primjenu evolucijskih algoritama na probleme strojnog učenja. Dan je kratak pregled različitih pristupa strojnom učenju s naglaskom na tehniku genetskog programiranja.

Peti dio se bavi primjenom evolucijskih algoritama na problem računalnog igranja kartaške igre Ajnc. Dan je kratak opis igre i predstavljeno rješenje problema tehnikom genetskog programiranja.

Šesti dio predstavlja rezultate testiranja učinkovitosti različitih modela paralelizacije evolucijskih algoritama. Opisana je metodologija mjerenja, te dana analiza osjetljivosti različitih inačica algoritama s obzirom na parametre paralelizacije i karakteristike problema.

Sedmi dio daje zaključak cijelog rada.

Osmo poglavlje daje popis korištene literature.

2. Evolucijski algoritmi

Evolucijski algoritmi su heuristička metoda optimiranja koja imitira prirodni proces evolucije. Po načinu rada spadaju u metode usmjerenog slučajnog pretraživanja prostora rješenja (engl. *guided random search techniques*) koje traže globalni optimum [1]. Evolucijski algoritmi služe za rješavanje teških optimizacijskih problema za koje ne postoji egzaktna matematička metoda rješavanja, ili su *NP*-teški pa se za veći broj nepoznanica ne mogu riješiti u zadovoljavajućem vremenu [5].

Učinkovitost ovakvih metoda proizlazi iz činjenice da su sposobne odrediti položaj globalnog optimuma u prostoru rješenja s više lokalnih ekstrema (tzv. višemodalnom prostoru) [1]. Za razliku od klasičnih deterministčkih metoda, koje će se uvijek kretati prema lokalnom minimumu ili maksimumu, stohastičke metode mogu s određenom vjerojatnošću pronaći globalni optimum zadane ciljne funkcije. Međutim, za rezultat rada evolucijskog algoritma nismo u mogućnosti sa sigurnošću reći radi li se o globalnom ili samo lokalnom optimumu, te s kolikom je preciznošću taj optimum određen. Zbog toga se koristi ponavljanje procesa rješavanja, čime kod stohastičkih metoda možemo povećati sigurnost rezultata.

Evolucijski algoritmi spadaju u skupinu metaheurističkih algoritama. To znači da je način rada algoritma neovisan o samom sadržaju i osobinama problema na koji se primjenjuje. Naravno, učinkovitost varira u ovisnosti o karakteristikama problema, ali čak i za probleme na kojima su evolucijski algoritmi posebno učinkoviti, detalji problema nisu ni na koji način ugrađeni u algoritam. Metaheuristički algoritmi (pa tako i evolucijski algoritmi) su se općenito pokazali učinkovitima u rješavanju problema sa velikim pretraživim prostorom mogućih rješenja.

Zahvaljujući nevjerojatno brzom rastu računalne snage u posljednjoj četvrtini 20. stoljeća i početkom 21. stoljeća, evolucijski algoritmi pokazali se vrlo učinkovitim i široko primjenjivim alatom za rješavanje mnogih problema iz inženjerske prakse. Objašnjenje leži u njihovoj jednostavnosti – kako same ideje, tako i praktične primjene.

2.1 Inspiracija biološkim procesom evolucije i prirodnog odabira

2.1.1 Darwinova istraživanja

Jedno od najvećih otkrića u povijesti biologije i znanosti u cjelini je teorija evolucije. Nju je prvi razvio Charles Darwin u svom djelu *O podrijetlu vrsta*, 1859. godine. Tu je iznio svoje tezu koja kaže da su se živi organizmi razvili postupno kroz milijarde godina, počevši od jednostaničnih organizama pa sve raznolikosti današnjeg živog svijeta.

Charles Darwin je ustvrdio da je razvoj populacije (evolucija) omogućen procesom prirodnog odabira. Primijetio je da veću šansu za preživljavanje i ostavljanje potomaka imaju jedinke sa boljim svojstvima, dok one sa lošijim svojstvima češće odumiru bez ostavljanja potomaka. Zbog toga populacija lagano napreduje po različitim značajkama njenih pripadnika (npr. brzini, otpornosti na bolesti, razvijenosti osjetila, snalažljivosti itd.). Nakon dužeg vremena odvijanja ovakvog procesa dolazi do formiranja potpuno novih vrsta.

Dva su glavna pokretača evolucije: [3]

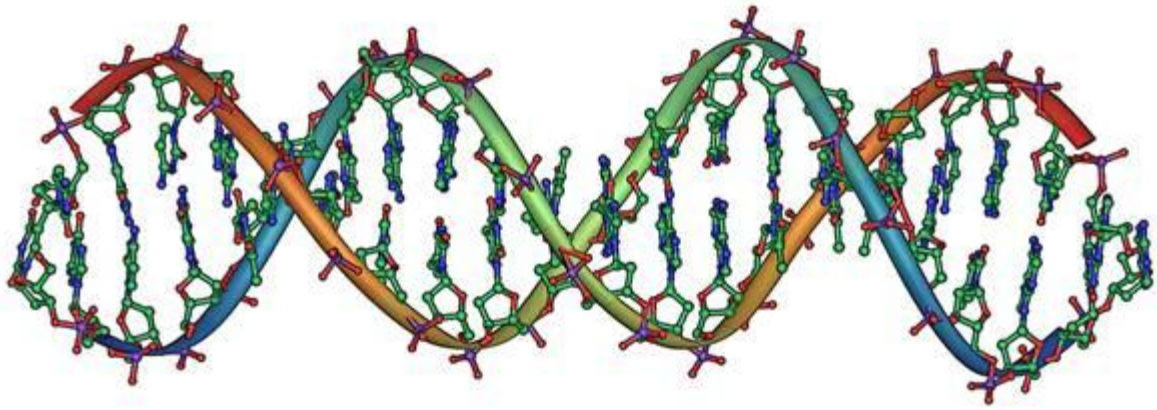
- Prirodni odabir - način na koji priroda izdvaja dobre jedinke za preživljavanje u borbi za opstanak.
- Spolno razmnožavanje – proces kojim se čuva raznolikost populacije koja generaciji omogućuje široki raspon značajki koje će im povećati šansu za preživljavanje u sljedećoj generaciji.

2.1.2 Moderna genetika

Ono što Darwin nije poznavao jest mehanizam kojim se svojstva roditelja prenose na djecu. Prve rezultate na tom području ostvario je Gregor Mendel, moravski redovnik iz 19. stoljeća. On je uzgajanjem i proučavanjem razvoja i razmnožavanja stabljika graha otkrio da se svojstva roditelja prenose na potomke u obliku diskretnih, nedjeljivih značajki (npr. za boju očiju, broj zubi...). Do tada se smatralo da se svako pojedino svojstvo jednog roditelja pomiješa sa pripadnim svojstvom drugog roditelja i onda se nekakva vrsta „prosjeka“ prenese na potomka. Biološki nosioci ovih

diskretnih značajki poslije su prozvani geni, a u vrijeme Mendela još nije bio poznat biološki mehanizam njihovog funkcioniranja.

Daljnji napredah tehnologije omogućio je uvid u strukturu stanice i njene jezgre te otkriće molekule DNK (deoksiribonukleinska kiselina) koja služi kao nositelj svih gena jedinke – njenog genotipa (Slika 2.1). Strukturu molekule DNK u obliku dvostruke, spiralne zavojnice opisala su 1953. godine dva engleska znanstvenika, James D. Watson i Francis Creek.



Slika 2.1 Djelić strukture molekule DNK

U prostoru se molekula DNK dijeli na manje lance i namata u strukture koje nazivamo kromosomima. Kod spolnog razmnožavanja, genetski materijal potomka se dobije iz kromosoma oba roditelja putem procesa rekombinacije. Rekombinacijom se pojedini dijelovi kromosoma jednog roditelja zamijene dijelovima kromosoma drugog roditelja. Trenutak razmjene dijelova kromosoma naziva se križanje (engl. *crossing over*) i predstavlja jedan način promjene genetske poruke pri stvaranju nove generacije u populaciji.

Drugi način promjene genetske poruke je mutacija. To je nepredvidljiva i slučajna promjena genetskog materijala na nekom od kromosoma koja nastaje kao rezultat malo vjerojatne anomalije u procesu diobe stanice. Posljedica mutacije je najčešće negativna, ali može biti i pozitivna. Upravo je pozitivnim mutacijama dodatno ubrzano napredovanje populacije u evoluciji, jer se neka nova i dobra značajka jako brzo širi kroz cijelu populaciju (npr. izrazito velika otpornost na neku bolest daje veću

šansu za preživljavanjem te jedinke i njenih potomaka). U slučaju negativne mutacije, jedinka vrlo vjerojatno umire bez stvaranja potomstva.

Procesima prirodne selekcije, križanja i mutacije odvija se spori proces evolucije koji je doveo do razvoja života na planetu Zemlji kakav je danas poznat.

2.2 Princip rada genetskih algoritama

Genetski algoritmi su jedna od metoda koje ubrajamo pod širu kategoriju evolucijskih algoritama. Genetski algoritam je prvi put predložio John H. Holland sedamdesetih godina dvadesetog stoljeća [2]. On ga je zamislio kao računalni proces koji imitira evolucijski proces u prirodi i primjenjuje ga na apstraktne jedinke.

Svaka apstraktna jedinka predstavlja rješenje problema koji se optimizira, a sve su jedinke predstavljene istom podatkovnom strukturom. Ovisno o problemu koji se obrađuje, odabrana podatkovna struktura za jedinku može biti broj, niz, matrica, stablo, graf itd. Jedinke genetskog algoritma se, u skladu s analogijom u prirodi, nazivaju kromosomima. Svaka jedinka (tj. svako potencijalno rješenje optimizacijskog problema) ima vlastitu vrijednost funkcije cilja ili dobrote. Bolje jedinke imaju bolji rezultat funkcije cilja što im daje veću vjerojatnost preživljavanja. Broj jedinki za vrijeme izvršavanja genetskog algoritma je stalan, a skup svih jedinki se naziva populacija.

Genetski algoritam se izvršava iterativno. Počevši od početne populacije, za svaku jedinku se računa njena vrijednost funkcije cilja i na temelju dobivene vrijednosti se, primjenom operatora selekcije, biraju jedinke koje će sudjelovati u formiranju nove generacije populacije. Selekcija oponaša borbu za opstanak u prirodi gdje dobre jedinke imaju veliku vjerojatnost opstanka, a loše jedinke malu vjerojatnost. Na „preživjele“ jedinke se zatim primjenjuje operator križanja. Križanjem jedinki i razmjenom genetskog koda formiraju se nove jedinke koje će zamijeniti one koje su otpale u procesu selekcije. Također se, s malom vjerojatnošću, primjenjuje operator mutacije koji pospješuje raznolikost genetskog materijala i pomaže u izbjegavanju lokalnog optimuma. Na ovaj način se dobije nova generacija populacije i genetski algoritam je spreman za novu iteraciju ili zaustavljanje.

2.2.1 Inicijalizacija i uvjet završetka evolucijskog procesa

Početna generacija populacije se može zadati na nekoliko načina: [1]

1) Slučajnim odabirom se iz prostora rješenja formiraju jedinke. Ovo je najčešći postupak u praksi.
2) Jedinke se formiraju na temelju rješenja neke druge optimizacijske metode (za koje pretpostavljamo da su bliže optimumu od slučajno generiranih).
3) Odaberu se sve međusobno istovjetne jedinke. Ovaj način nije naročito popularan jer je potreban velik broj generacija i primjena genetskih operatora za postizanje genetske raznolikosti.

Za zaustavljanje genetskog algoritma može postojati više kriterija. Neki od njih su:

1) Unaprijed zadan broj iteracija
2) Prag dobrote rješenja nakon kojeg proglašavamo rješenje zadovoljavajućim i zaustavljamo genetski algoritam
3) Nije došlo do poboljšanja rješenja kroz neki određen broj iteracija
4) U nekim primjenama algoritam se zaustavlja i ako je velik broj članova populacije dovoljno blizu u prostoru rješenja

2.2.2 Jednostavni genetski algoritam

Jednostavni genetski algoritam koristi binarni prikaz jedinke. U svakoj generaciji se primjenjuje jednostavan operator selekcije. Nakon selekcije, na jedinke se primjenjuje operator križanja s jednom točkom prekida, a zatim se na jedinke primjenjuje jednostavna mutacija.

```
Genetski_algoritam
{
    t = 0;
    generiraj početnu populaciju potencijalnih rješenja P(0);
    sve dok nije zadovoljen uvjet završetka evolucijskog procesa
    {
        t = t + 1;
        selektiraj P'(t) iz P(t-1);
        križaj jedinke iz P'(t) i djecu spremi u P(t);
        mutiraj jedinke iz P(t);
    }
    ispiši rješenje;
}
```

Slika 2.2 Pseudokod jednostavnog genetskog algoritma

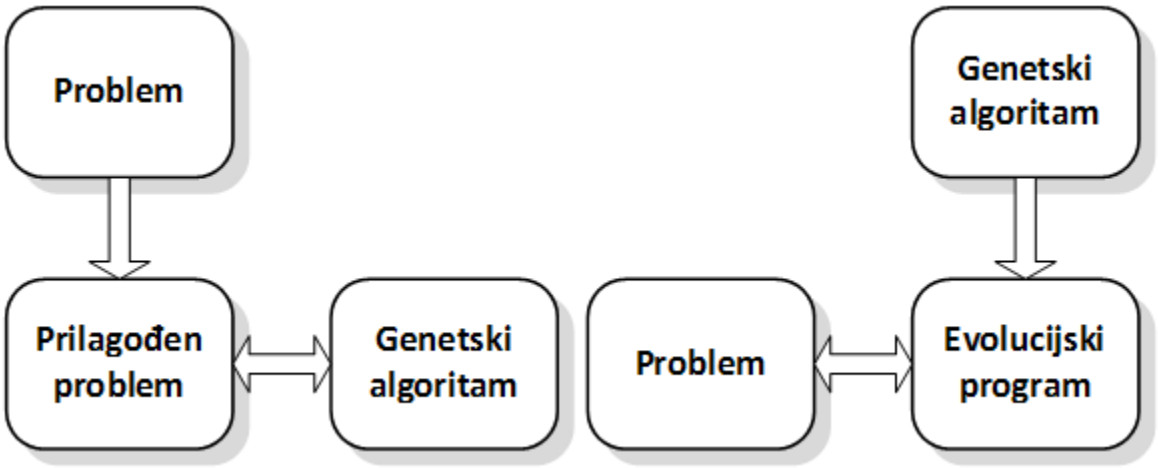
Slika 2.2 prikazuje strukturu jednostavnog genetskog algoritma kroz pseudokod. Algoritam započinje generiranjem početne populacije potencijalnih rješenja. Nakon inicijalizacije populacije, algoritam ulazi u petlju koja će se izvršiti konačan broj puta. Na kraju algoritam ispisuje najbolje dobiveno rješenje.

2.2.3 Metodologija rješavanja problema uporabom genetskog algoritma

Postoje dva pristupa rješavanju problema pomoću genetskog algoritma: [1]

1) Prilagoditi problem genetskom algoritmu
2) Genetski algoritam prilagoditi specifičnostima problema

Korištenjem prvog pristupa, potrebno je napraviti prilagodbu problema genetskom algoritmu na način da se za prikaz rješenja koriste općenite strukture podataka s kojima genetski algoritam zna raditi (npr. niz bitova, stablo...). To znači da će postojati preslikavanje općenitog prikaza rješenja u stvarno rješenje. Međutim, zbog preslikavanja može doći do situacije u kojoj velik broj jedinki (predstavljenih općenitim zapisom) ne daje ispravno rješenje, tj. ne mogu se preslikati u stvarno rješenje. Također, postoje problemi čija se rješenja ne mogu modelirati na ovaj način (npr. problem rasporeda se vrlo teško može prikazati nizom bitova).



Slika 2.3 Dva pristupa rješavanju problema pomoću genetskog algoritma

U drugom pristupu, algoritam rukuje veličinama svojstvenima određenom problemu. Ovo najčešće znači korištenje posebnih struktura podataka za prikaz jedinki, te korištenje specijaliziranih genetskih operatora. Ovim pristupom dobije se visoko specijalizirani genetski algoritam, koji se tada obično naziva *evolucijskim programom*. Djelotvornost takvih evolucijskih programa je najčešće izvrsna, ali je rješenje problemski ovisno i zahtjeva dosta rada na prilagođavanju.

2.3 Načini prikaza kromosoma

Za prikaz kromosoma unutar populacije koristi se niz različitih struktura podataka [4]. Zbog širokog spektra problema na koje su genetski algoritmi primjenjivi, strukture podataka koje se obično koriste su raznolike, a opet dovoljno općenite kako bi svaki način prikaza kromosoma bio upotrebljiv na što većem broju stvarnih problema. Kako bi bio koristan za genetski algoritam, svaki način prikaza kromosoma mora biti popraćen odgovarajućim operatorima križanja i mutacije.

2.3.1 Binarni prikaz

Binarni prikaz kromosoma je najstariji i najčešće korišten prikaz. U binarnom prikazu, svaki kromosom je niz bitova sa vrijednošću 0 ili 1.

Tablica 2.1 Primjer binarnog prikaza kromosoma

Kromosom A	101100101100101011100101
Kromosom B	111111100000110000011111

Ovakav prikaz omogućuje zapis velikog broja rješenja uz relativno malu veličinu kromosoma. S druge strane, binarni prikaz rješenja je često neprirodan za mnoge probleme i ponekad zahtjeva popravak kromosoma nakon križanja ili mutacije kako bi se izbjegla neispravna rješenja.

2.3.2 Permutacijski prikaz

Permutacijski prikaz kromosoma se najčešće koristi kod problema čije je rješenje neka vrsta redoslijeda (npr. gradova, zadataka...). Tipičan primjer takvog problema je problem trgovačkog putnika (engl. *travelling salesman problem*).

Tablica 2.2 Primjer permutacijskog prikaza kromosoma

Kromosom A	1 5 3 2 6 4 7 9 8
Kromosom B	8 5 6 7 2 3 1 4 9

Iako je ovakav prikaz dobro prilagođen spomenutim problemima, ipak je ponekad potrebno napraviti ispravke na kromosomu nakon križanja ili mutacije kako bi se dobilo ispravno rješenje.

2.3.3 Vrijednosni prikaz

U problemima gdje je rješenje predstavljeno nekom vrijednošću poput realnog broja, jedna takva vrijednost ili niz vrijednosti mogu biti direktno pohranjeni u kromosomu. Korištenje binarnog prikaza za ovakve probleme je moguće i tradicionalno je u praksi to često i bio slučaj, međutim direktno pohranjivanje vrijednosti unutar kromosoma često pruža puno veću lakoću korištenja.

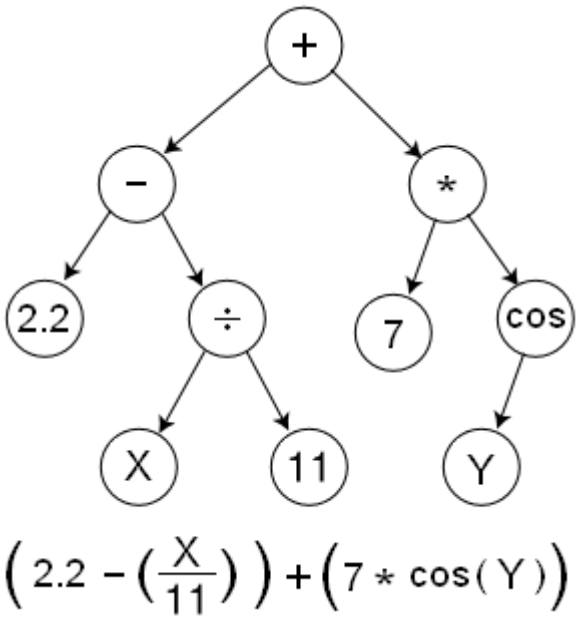
Tablica 2.3 Primjer vrijednosnog prikaza kromosoma

Kromosom A	1.2324 5.3243 0.4556 2.3293 2.4545
Kromosom B	ABDJEIFJDHDIERJFDLDFLFEGT
Kromosom C	(natrag), (natrag), (desno), (naprijed), (lijevo)

Nedostatak ovog pristupa je što je za postizanje dobrih rezultata često potrebno razviti prilagođene operatore križanja i mutacije za pojedini problem.

2.3.4 Stablasti prikaz

Stablasti prikaz kromosoma se prvenstveno koristi za evoluciju programa ili aritmetičkih izraza u područjima genetskog programiranja i strojnog učenja. Kod ovakvog prikaza, svaki kromosom predstavlja stablo objekata kao što su funkcije ili naredbe nekog programskog jezika.



Slika 2.4 Primjer stablastog prikaza kromosoma

Programski jezik Lisp, zahvaljujući prirodnoj stablastoj strukturi njegovih programa, se često koristi kao odredišni jezik čiji se programi evoluiraju.

2.4 Genetski operatori

2.4.1 Evaluacija

Mjera kvalitete svake jedinke određuje se primjenom operatora evaluacije. Sam genetski kod nije na jednostavan način povezan s kvalitetom rješenja kojeg predstavlja, već se dobrota mora izračunati. Za to je potrebno odrediti funkciju dobrote (funkcija cilja, engl. *fitness function*), koja je usko povezana s samom funkcijom iz stvarnog života koju želimo optimizirati. U najjednostavnijem slučaju, funkcija dobrote i funkcija koju optimiziramo su jednake i tada vrijedi:

$$dobrota(v) = f(x)$$

gdje *v* predstavlja kromosom a *x* je preslika prikaza kromosoma u stvarno rješenje.

Pošto funkcija dobrote ima neka ograničenja koja omogućuju da genetski algoritam ispravno funkcionira, a funkcija *f* je funkcija iz stvarnog svijeta koju želimo optimizirati i često je posve proizvoljna, ovakva jednostavna veza među njima često nije moguća. U tom slučaju, potrebno je mnogo pažnje posvetiti odabiru funkcije dobrote jer njena pogrešna ocjena često dovede do evolucije populacije u krivom smjeru i nezadovoljavajućim rezultatima genetskog algoritma.

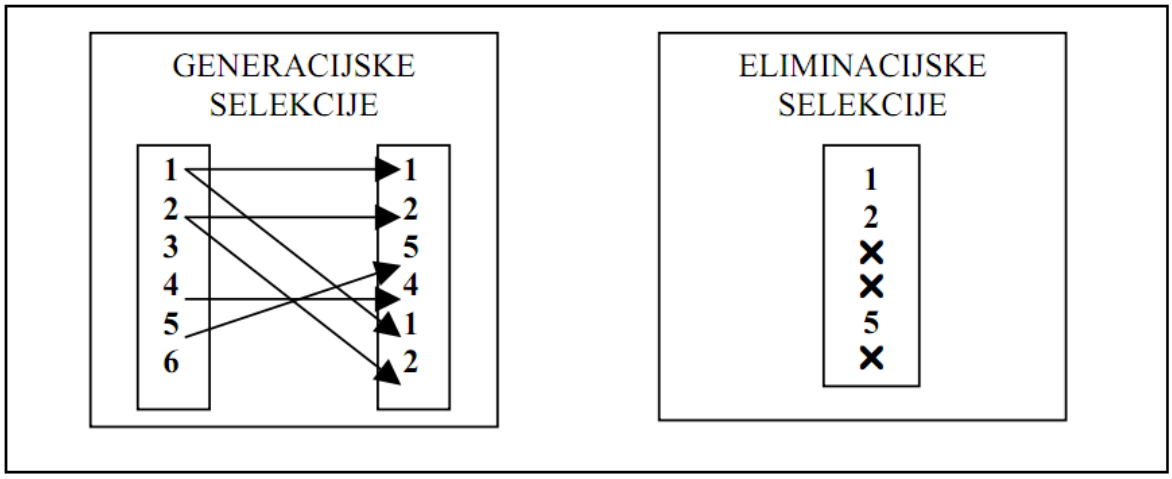
2.4.2 Selekcija

Uloga selekcije je čuvanje i prenošenje dobrih značajki jedinki u sljedeću generaciju te odbacivanje loših značajki. Naivni postupak selekcije bi mogao jednostavno sačuvati najbolje jedinke iz populacije, a odbaciti najlošije. Međutim, takav postupak dovodi do vrlo brze konvergencije genetskog algoritma u lokalnom optimumu, nakon čega nema više poboljšanja [1]. To se događa zbog toga što se navedenim postupkom gubi dobar genetski materijal kojeg mogu sadržavati inače loše jedinke. Zbog toga je potrebno selekciju učiniti stohastičkom, te omogućiti lošim jedinkama neku malu mogućnost preživljavanja nauštrb dobrih jedinki. Ovakav postupak donosi rizik od gubitka dobrog genetskog materijala, ali čuva genetsku raznolikost populacije.

Vrste operatora selekcije dijelimo na generacijske i eliminacijske: [5]

1) Selekcije koje spadaju u generacijske odabiru iz trenutne generacije najbolje jedinke i njihovim kopiranjem stvaraju novu generaciju. Nedostatak ovakve vrste selekcije je što u sljedećoj generaciji može postojati više potpuno jednakih jedinki, čime se smanjuje vrijedna genetska raznolikost.

2) Eliminacijske selekcije su usredotočene na biranje jedinki koje će biti odstranjene iz populacije. Lošije jedinke imaju manju vjerojatnost preživljavanja. Nakon njihovog uklanjanja, populacija se nadopuni novim jedinkama koje se dobiju križanjem boljih kromosoma.



Slika 2.5 Podjela postupka selekcije prema načinu prenošenja genetskog materijala boljih jedinki u novu generaciju

Postoje različiti konkretni mehanizmi selekcije:

1) Jednostavna selekcija (engl. *roulette wheel parent selection*) spada skupinu generacijskih vrsta selekcije. Kod jednostavne selekcije, jedinke koje će sudjelovati u reprodukciji odabiru se s vjerojatnošću koja je proporcionalna njihovoj kvaliteti.

2) Turnirska selekcija (engl. *tournament selection*) može biti generacijska ili eliminacijska. Kod ove vrste selekcije, slučajnim odabirom se bira *k* jedinki iz populacije, koje sudjeluju u turniru. U slučaju generacijske selekcije, među izabranim jedinkama se traži najbolja koja se zatim kopira u sljedeću generaciju, a postupak je potrebno ponoviti toliko puta kolika je veličina populacije. Ako se radi o eliminacijskoj selekciji, najlošija jedinka od odabranih se uklanja, a zamjenjuje je nova jedinka nastala križanjem dvije slučajne

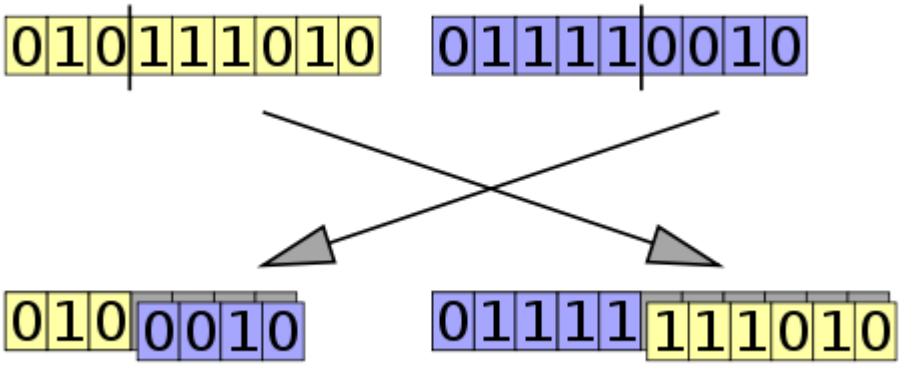
jedinke iz trenutnog turnira. Ova vrsta selekcija, a posebno njena eliminacijska inačica, je pogodna za probleme koji imaju složene kromosome i kod kojih je izračun funkcije dobrote vremenski zahtjevan.

3) Eliminacijska selekcija (engl. *steady state selection*) je eliminacijska varijanta jednostavne selekcije. Za razliku od jednostavne selekcije, ne biraju se kromosomi koji će preživjeti već oni koji će biti eliminirani. Stoga se definira funkcija kazne, koja ima veću vrijednost za lošije jedinke i predstavlja vjerojatnost uklanjanja jedinke iz populacije.

Elitizam je postupak kojim se može unaprijediti neka od navedenih vrsta selekcije, a predstavlja mehanizam zaštite najbolje jedinke iz populacije od nestajanja. Kako su postupci selekcije stohastičke prirode, ne postoji garancija da će jedinka sa najboljim genetskim materijalom preživjeti u sljedeću generaciju. Uvođenjem elitizma se osigurava da se ta jedinka ne izgubi. Genetski algoritam s ugrađenim elitizmom asimptotski napreduje prema globalnom optimumu. Nedostatak elitizma je usporenje rada genetskog algoritma zbog postupka traženja najbolje jedinke prije početka selekcije.

2.4.3 Križanje

Genetski operator križanja (engl. *crossover*) kombinira genetski materijal dvije jedinke (koje se nazivaju roditelji), a kao rezultat nastaju jedna ili dvije jedinke koje se nazivaju djeca. Smisao operatora križanja je nasljeđivanje karakteristika – nastala djeca će imati vrlo slične značajke kao roditelji na koje je primijenjen operator križanja. Pošto se križanje u genetskom algoritmu primjenjuje na jedinke koje su prošle proces selekcije, velika je vjerojatnost da će te jedinke biti kvalitetne te će sukladno tome njihova djeca biti jednako kvalitetna, ako ne i kvalitetnija od roditelja. Kako u njemu sudjeluju dvije jedinke, križanje spada u binarne operatore.



Slika 2.6 Križanje kromosoma s jednom točkom prekida. U ovom slučaju točka križanja nije ista na oba roditeljska kromosoma pa rezultirajuća djeca nisu jednake duljine

Detalji mehanizma operatora križanja ovise o podatkovnoj strukturi koja se koristi za prikaz kromosoma. Razmotrit ćemo nekoliko vrsta križanja na binarnom prikazu kromosoma.

Križanje s m *točaka prekida* se obavlja na način da se kromosomi razrežu na *m* pozicija u binarnom prikaz. Roditelji zatim međusobno razmijene razrezane dijelove, te se ponovo spoje u dva nova potpuna kromosoma. Slika 2.6 prikazuje ovakvu vrstu križanja.

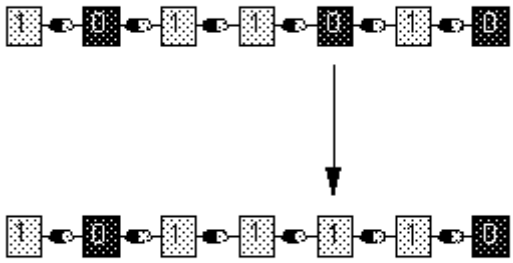
Drugi način križanja kromosoma je tzv. *uniformno križanje*. Pri toj vrsti križanja, za svaki pojedini gen (tj. bit) djeteta se slučajnim odabirom bira hoće li se naslijediti od prvog ili drugog roditelja.

Operator križanja je jedinstvena karakteristika genetskih algoritama, koju ne nalazimo u ostalim područjima evolucijskog računarstva.

2.4.4 Mutacija

Drugi operator koji nalazimo u genetskim algoritmima je *operator mutacije*. Mutacija je unarni operator koji djeluje nad jednom jedinkom i na slučajan način je mijenja. Nepredvidljiva priroda mutacija omogućuje genetskom algoritmu da iznenada počne pretraživati neistraženo područje rješenja u kojem se možda nalazi globalni optimum, čak i u slučajevima kad se većina populacije okupila oko nekog lokalnog optimuma. U slučaju pozitivnog pomaka uslijed mutacije, populacija će se brzo preseliti na novo

neistraženo područje. Ukoliko mutacija daje loše rezultate, mutirana jedinka brzo odumire bez velikih posljedica na ostatak populacije.



Slika 2.7 Prikaz mutacije jednog bita kromosoma

Primjer operatora mutacije je *jednostavna mutacija.* Kod binarnog prikaza kromosoma, jednostavna mutacija mijenja svaki gen (tj. bit) kromosoma s vjerojatnošću p_m, koja je ulazni parametar genetskog algoritma. Kada je vjerojatnost p_m bliska nuli, populacija će često zaglaviti u lokalnom optimumu. Ukoliko je vjerojatnost p_m bliska jedinici, algoritam se pretvara u slučajnu pretragu prostora rješenja. Slika 2.7 prikazuje mutaciju 5. bita kromosoma iz nule u jedinicu.

Osim izbacivanja populacije iz lokalnog optimuma, mutacija također omogućuje povratak izgubljenog genetskog materijala kojeg nije moguće vratiti samo križanjem jedinki [1].

3. Paralelni evolucijski algoritmi

Evolucijski algoritmi su pronašli svoju nišu u rješavanju jako teških optimizacijskih problema, često onih u *NP* klasi složenosti. Iako je njihov razvoj omogućio rješavanje problema koji se egzaktnim metodama ne bi mogli ni „načeti“, brzina izvršavanja evolucijskih algoritama i dalje predstavlja jedan od najvećih izazova u njihovoj praktičnoj primjeni.

Postoji nekoliko načina za povećanje djelotvornosti evolucijskih algoritama. Učinkovitiji algoritam se može postići povećanjem vjerojatnosti postizanja dobrih rješenja, povećanjem kvalitete dobivenih rješenja i skraćenjem trajanja izvođenja programa. Nadalje, skraćenje trajanja izvođenja programa se može postići povećanjem brzine konvergencije rješenja i smanjenjem broja iteracija evolucijskog algoritma, ubrzavanjem izvođenja jedne iteracije te paralelizacijom izvođenja algoritma [5].

Od navedenih metoda, povećanje vjerojatnosti postizanja dobrih rješenja, povećanje kvalitete rješenja i smanjenje broja iteracija se postižu finim podešavanjem parametara evolucijskog algoritma. Podešavanje parametara je dugotrajan posao jer se može obaviti isključivo na temelju eksperimentalnih mjerenja. Iz navedenog proizlazi da je jedna od poželjnih karakteristika evolucijskog algoritma što manji broj podesivih parametara.

Trajanje izvođenja evolucijskog algoritma se može skratiti i optimiziranjem izvornog koda programa. Time su ujedno iscrpljene sve mogućnosti skraćenja trajanja slijednog evolucijskog algoritma, tj. onog koji se izvodi na jednom procesoru. Daljnje mogućnosti poboljšanja brzine izvođena pruža *paralelizacija*.

3.1 Paralelizacija genetskih algoritama

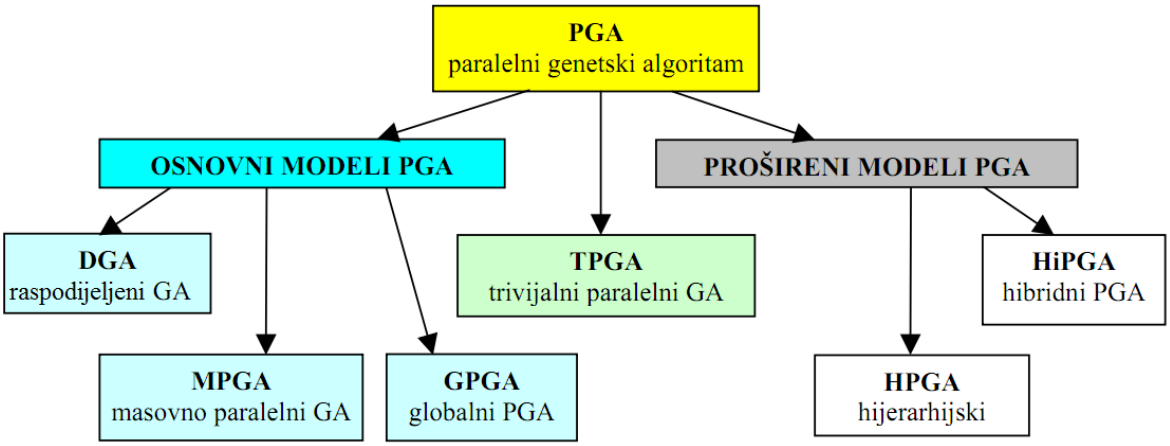
Cilj paralelizacije evolucijskih algoritama općenito, pa tako i genetskih algoritama, je skraćenje trajanja njihovog izvršavanja bez narušavanja poželjnih svojstava slijednog algoritma. Osnovni pristup paralelizacije jest raščlanjivanje slijednog programa na međusobno nezavisne podzadatke sa ciljem njihovog paralelnog izvršavanja. Kod

genetskog algoritma postoji nekoliko kandidata za paralelno izvršive podzadatke. Pri njihovom biranju se mogu izdvojiti dva pristupa:

1) U standardni pristupu se paraleliziraju genetski operatori i izračun vrijednosti funkcije cilja
2) U dekompozicijskom pristupu se populacija dijeli na manje dijelove – subpopulacije (engl. *subpopulation, deme*), te se zatim cijeli genetski algoritam paralelno obavlja na više subpopulacija.

U prvom pristupu se najčešće paralelizira samo postupak evaluacije, tj. izračun vrijednosti funkcije cilja. Evaluacija je često vremenski najzahtjevnija operacija u radu genetskog algoritma, pa se njenom paralelizacijom dobiju najveća ubrzanja. Čitav algoritam evoluira samo jednu populaciju, pa se ovakav model naziva *jednopopulacijskim*. Ovim pristupom se u izvršavanju genetskog algoritma mogu izdvojiti dvije uloge, *gospodar* i *sluga* ili *radnik*. Gospodar izvršava genetske operatore selekcije, križanja i mutacije nad jednom ukupnom populacijom jedinki, dok radnici paralelno računaju vrijednost funkcije cilja jedinki nakon što gospodar obavi svoj slijedni dio posla.

Drugi pristup se, zbog dijeljenja populacije na više subpopulacija, naziva višepopulacijski model. Kako subpopulacije sadrže manje jedinki od čitave populacije u jednopopulacijskom modelu, a njihova evolucija se odvija paralelno, može se očekivati ubrzanje izvođenja u odnosu na slijedni model. Prema načinu formiranja subpopulacija, mogu se razlikovati *krupnozrnati* (engl. *coarse-grained*) i *sitnozrnati* (engl. *fine-grained*) pristup. Krupnozrnati pristup dijeli populaciju na manji broj subpopulacija i raspoređuje ih na dostupnim procesorima za paralelno izvršavanje. Sitnozrnati pristup je omogućen napretkom tehnologije i razvojem tzv. masovno paralelnih računala sa velikim brojem procesora (ili jezgri). Populacija se dijeli na subpopulacije jedne jedine jedinke, a svaki procesor izvršava genetske operatore nad njemu dodijeljenom jedinkom i jedinkama njemu susjednih procesora.



Slika 3.1 Podjela paralelnih genetskih algoritama

Uzimajući u obzir moguće pristupe paralelizaciji genetskih algoritama, možemo identificirati trenutno najpopularnije modele paralelnih genetskih algoritama: *raspodijeljeni genetski algoritam* (engl. *distributed genetic algorithm*), *masovno paralelni genetski algoritam* (engl. *massively parallel genetic algorithm*) i *globalni paralelni genetski algoritam* (engl. *master-slave genetic algorithm*).

Navedena tri modela se mogu međusobno kombinirati ili nadograditi nekom drugom metodom optimiranja, pa sukladno tome dobijemo još dva proširena modela genetskih algoritama: *hijerarhijski paralelni genetski algoritam* (engl. *hierarchical parallel genetic algorithm*) i *hibridni paralelni genetski algoritam* (engl. *hybrid parallel genetic algorithm*).

Posljednji i najjednostavniji model paralelnih genetskih algoritama je *trivijalni paralelni genetski algoritam (engl. embarrassingly parallel genetic algorithm*) u kojem se više genetskih algoritama paralelno obavlja na nekoliko potpuno nezavisnih računala. Svrha takvog izvođenja algoritma je statistička obrada dobivenih rezultata ili određivanje optimalnog skupa parametara. Kako je genetski algoritam u osnovi statistički proces, opetovano izvršavanje istog algoritma može dati korisne podatke za statističku analizu.

3.2 Raspodijeljeni genetski algoritam

Raspodijeljeni genetski algoritam je vrsta paralelnog genetskog algoritma koji populaciju dijeli na više subpopulacija, koje zatim iskorištava za formiranje

paraleliziranih podzadataka, čime spada u krupnozrnate paralelne genetske algoritme. Važna karakteristika raspodijeljenog genetskog algoritma je mogućnost iskorištavanja umreženih računala kao čvorova za izvršavanje paralelnih zadataka, iako čvorovi mogu biti i pojedini procesori na jednom višeprocesorskom računalu koji komuniciraju preko zajedničkog spremnika. Konkretni genetski algoritmi u pojedinim čvorovima se mogu međusobno razlikovati, ali obavezno dijele funkciju cilja [5]. Broj subpopulacija je obično jednak broju procesora na kojima se algoritam izvršava.

Tablica 3.1 Pseudokod raspodijeljenog genetskog algoritma

```
Raspodijeljeni_genetski_algoritam(){
   inicijaliziraj_P_populacija();
   dok (nije_zadovoljen_uvjet_završetka_evolucijskog_procesa){
      za svaku subpopulaciju obavljaj paralelno{
         evaluiraj();
         ako( (broj_iteracija % period_izmjene) == 0 ){
            migracija();   // izmjeni jedinke
         }
         selektiraj();
         križaj();
         mutiraj();
      }
   }
}
```

Više slijednih genetskih algoritama se paralelno izvršavaju nad subpopulacijama, koje su uglavnom izolirane kako bi im se omogućilo da pretražuju različite dijelove prostora rješenja. Ipak, razmjena genetskog materijala među subpopulacijama je omogućena razmjenom jedinki ili *migranata*, što omogućuje raspodijeljenom genetskom algoritmu postizanje boljih rezultata od skupine genetskih algoritama koji se nezavisno izvršavaju (tj. trivijalnog paralelnog genetskog algoritma). Migraciju jedinki se obavlja na način da bolje jedinke neke populacije zamjene loše jedinke druge populacije i nju obavlja *genetski operator migracije.*

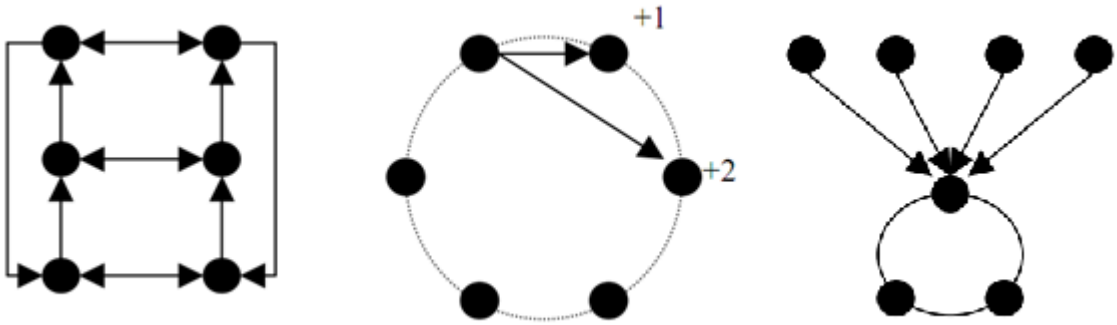
Veličina subpopulacije u pojedinim čvorovima je najčešće manja od populacije koja bi se koristila u slijednom algoritmu, što omogućuje brže izvođenje raspodijeljenog genetskog algoritma. Usporavajući faktor je razmjena jedinki među subpopulacijama, pa ubrzanje nije onoliko koliko je manja subpopulacija od populacije slijednog algoritma.

Zbog svoje pogodnosti za ugradnju na umreženim računalnima raspodijeljeni genetski algoritam je najčešće korištena inačica paralelnog genetskog algoritma u praksi. Također, ideja o razdvojenim subpopulacijama koje paralelno evoluiraju je dobro potkrijepljena primjerima u prirodi, gdje često nailazimo na slučaj vrste razdvojene, primjerice, kontinentima paralelno evoluiraju.

3.2.1 Migracija

Migracija i mehanizam njenog izvođenja ima snažan utjecaj na učinkovitost raspodijeljenog genetskog algoritma. Postupak migracije određen je sa pet parametara:

1) Migracijski interval M_i određuje broj iteracija između dvije migracije. Ovim parametrom se podešava učestalost razmjene jedinki između subpopulacija. Izoliranost subpopulacija je važna zbog izbjegavanja prerane konvergencije svih subpopulacija prema nekom lokalnom optimumu. Ipak, subpopulacije ne smiju biti potpuno izolirane kako bi se dobra rješenja mogla proširiti kroz čitavu populaciju.
2) Migracijska stopa M_s određuje broj jedinki koje se razmjenjuju u postupku migracije. Ovaj parametar utječe na raznolikost populacije na sličan način kao i migracijski interval, te je stoga važan za izbjegavanje prerane konvergencije subpopulacija u lokalni optimum. Često je broj jedinki koje se razmjenjuju u migraciji jednak jedinici.
3) Strategija odabira jedinki za migraciju i eliminaciju određuje koje će jedinke biti odabrane za prijenos u drugu subpopulaciju, te koje će jedinke biti zamijenjene novopridošlim jedinkama. Odabrana strategija značajno utječe na selekcijski pritisak algoritma, a time i na brzinu konvergencije ka rješenju. U praksi se najčešće koristi strategija u kojoj se najbolje jedinke biraju za migraciju, a njima se nadomještaju slučajno odabrane ili najlošije jedinke.
4) Topologija razmjene jedinki je plan po kojem čvorovi genetskog algoritma razmjenjuju jedinke. Pokazalo se da topologija razmjene jedinki ima znatan utjecaj na vjerojatnost zaglavljivanja genetskog algoritma u lokalnom optimumu. Slika 3.2 prikazuje neke od uobičajenih topologija razmjene jedinki.



Slika 3.2 Primjeri topologija migracije jedinki – 1) ljestvičasta topologija, 2) kružna +1+2 topologija, 3) dvoslojna otočna topologija

3.3 Masovno paralelni genetski algoritam

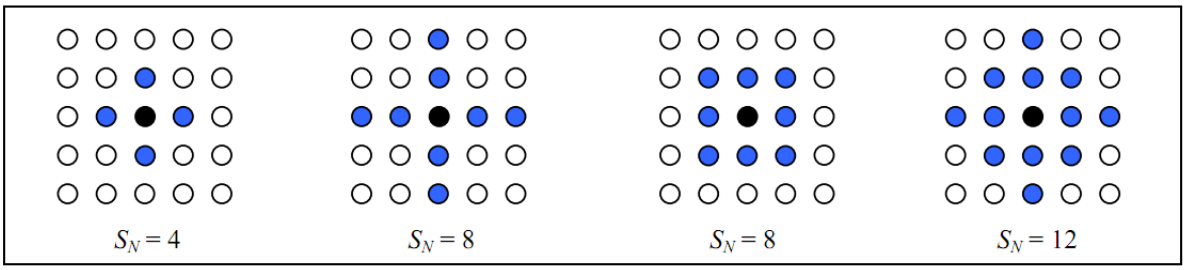
Masovno paralelni genetski algoritam je vrsta paralelnog genetskog algoritma koji je vrlo sličan raspodijeljenom genetskom algoritmu, ali koristi sitnozrnatu podjelu populacije na subpopulacije. Zbog toga je takav algoritam pogodan za izvršavanje samo na višeprocesorskim računalima sa velikim brojem procesora. Veličina populacije je često upravo onolika koliki je dostupan broj procesora, iako se ponekad koristi inačica sa više jedinki po procesoru.

Masovno paralelni genetski algoritam radi na način da je u internoj memoriji procesora pohranjena jedna jedinka. Svi procesori paralelno obavljaju sve genetske operatore, kao i evaluaciju jedinki. Unarni operatori (evaluacija i mutacija) se obavljaju nad jedinkom koja pripada procesoru koji je obavlja, dok se binarni operator križanja obavlja među jedinkama susjednih procesora, koji su određeni topologijom susjedstva.

Tablica 3.2 Pseudokod masovno paralelnog genetskog algoritma

```
Masovno_paralelni_genetski_algoritam(){
   generiraj_paralelno_populaciju_slučajnih_jedinki();
   dok (nije_zadovoljen_uvjet_završetka_evolucijskog_procesa){
      evaluiraj();     // evaluiraj paralelno svaku jedinku
      selektiraj();    // selektiraj paralelno jedinku za
                       // reprodukciju među susjedima
      reproduciraj(); // obavi paralelno reprodukciju među odabranom
                       // jedinkom iz prethodnog koraka i vlastitom
                       // jedinkom
      nadomjesti();    // nadomjesti paralelno vlastitu jedinku s
                       // novodobivenom jedinkom
   }
}
```

Topologija susjedstva je važan parametar kod izvoženja masovno paralelnih genetskih algoritama. Ona direktno utječe na brzinu i način propagacije boljih jedinki kroz populaciju. Analogno karakteristikama migracije kod raspodijeljenih genetskih algoritama, prevelik broj susjednih procesora S_N čini subpopulacije nedovoljno izoliranima, što će rezultirati prebrzom konvergencijom rješenja ka lokalnom optimumu. Manji broj susjednih procesora omogućuje veću izoliranost subpopulacija i sukladno tome pretraživanje većeg prostora rješenja.



Slika 3.3 Ilustracija nekoliko različitih topologija susjedstva masovno paralelnog genetskog algoritma

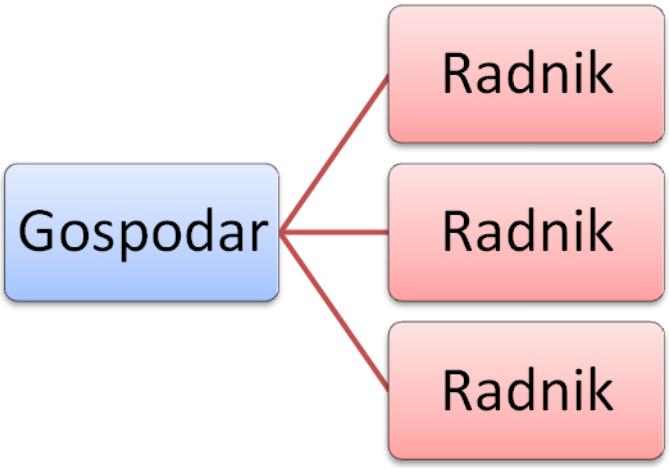
3.4 Globalni paralelni genetski algoritam

Globalni paralelni genetski algoritam je jednopopulacijski algoritam koji pri izvršavanju koristi jednu dretvu gospodara i više dretvi radnika. Gospodar je zadužen za raspoređivanje poslova radnicima, sinkronizaciju razmjene jedinki i upravljanje tokom genetskog algoritma.

Tablica 3.3 Pseudokod tradicionalnog globalnog paralelnog genetskog algoritma

```
Tradicionalni_globalni_paralelni_genetski_algoritam(){
   generiraj_početnu_populaciju_jedinki();
   dok (nije_zadovoljen_uvjet_završetka_evolucijskog_procesa){
      // SLUGE:
      evaluiraj_paralelno(); // evaluiraj paralelno jedinke

      // GOSPODAR:
      selektiraj();    // selektiraj jedinku za reprodukciju
      krizaj();        // obavi reprodukciju nad odabranim jedinkama
      mutiraj();
   }
}
```

Tradicionalni globalni paralelni genetski algoritam raspodjeljuje zadaću evaluacije jedinki na radnike, dok gospodar obavlja sve ostale genetske operatore. Očito je da se u tom slučaju radi o jednopopulacijskom modelu pošto gospodar ima sve jedinke dostupne i pohranjene u vlastitoj radnoj memoriji. Komunikacija između gospodara i radnika se sastoji od razašiljanja jedinki radnicima od strane gospodara, kako bi se obavila njihova evaluacija, te vraćanja izračunatih vrijednosti funkcije cilja gospodaru. Za vrijeme obavljanja posla, radnici ne komuniciraju s gospodarom niti međusobno. Postoji mogućnost da radnici izvrše i ostale genetske operatore osim evaluacije, međutim tada se ne radi o tradicionalnom globalnom paralelnom genetskom algoritmu.



Slika 3.4 Topologija gospodara i radnika u globalnom paralelnom genetskom algoritmu

Računanje vrijednost funkcije cilja je često najzahtjevnija operacija u genetskom algoritmu. To je razlog zbog kojeg se kod tradicionalnog oblika globalnog paralelnog

genetskog algoritma samo taj operator paralelizira. Što je izračun funkcije cilja vremenski zahtjevniji to se veće ubrzanje postiže njegovom paralelizacijom.

Faktor koji usporava rad globalnog paralelnog genetskog algoritma je vrijeme potrošeno na komunikaciju između gospodara i radnika T_C. U idealnom slučaju, kada je trajanje komunikacije zanemarivo, ubrzanje je jednako broju procesora na kojima se algoritam izvršava. Međutim, u praksi trajanje komunikacije nije zanemarivo pa je ubrzanje nešto manje. Ukupno vrijeme potrošeno na komunikaciju sa svakim od S radnika iznosi $(S+1)\cdot T_c$. Ako je vrijeme potrebno za evaluaciju jedne jedinke T_f, onda je kod slijednog algoritma ukupno trajanje evaluacije $N \cdot T_f$, a kod paralelnog modela $\frac{N \cdot T_f}{S+1}$ (pošto S radnika i gospodar paralelno obavljaju evaluaciju). Ukupno trajanje jedne iteracije tada iznosi:

$$T_{uk} = (S+1)\cdot T_c \frac{N \cdot T_f}{S+1}$$

Optimalan broj radnika S^* dobijemo derivacijom izraza za ukupno vrijeme trajanja iteracije i izjednačavanjem s nulom:

$$\frac{\partial T_{uk}}{\partial S} = T_c - \frac{N \cdot T_f}{(S^*+1)^2} = 0 \Rightarrow S^* = \sqrt{\frac{N \cdot T_f}{T_c}} - 1$$

Globalni paralelni genetski algoritam može biti sinkroni i asinkroni. Kod sinkronog modela, gospodar čeka dok radnici obave čitav posao prije prelaska u sljedeću iteraciju genetskog algoritma. Kod asinkrone verzije algoritma, gospodar ne čeka da radnici završe s evaluacijom razaslanih jedinki, već odmah nakon dodjele posla počinje s izvođenjem sljedeće iteracije. Zbog toga se može dogoditi da prilikom postupka selekcije u idućoj iteraciji neke jedinke imaju pogrešnu vrijednost funkcije cilja, tj. onu zaostalu iz prošle iteracije.

Važna osobina sinkrone verzije globalnog paralelnog genetskog algoritma jest da je to jedini model paralelnog genetskog algoritma koji ima iste karakteristike kao slijedni

genetski algoritam, pa čitava teorija vezana uz slijedne genetske algoritme ostaje primjenjiva.

3.4.1 Utjecaj heterogenog sklopovlja radnika na osobine algoritma

Kako se globalni paralelni genetski algoritam često izvršavana na odvojenim računalima međusobno povezanima u mrežu, procesorska snaga pojedinih radnika se može znatno razlikovati. Ta razlika može negativno utjecati na ubrzanje ostvareno uporabom paralelizacije.

Za analizu utjecaja različitog kapaciteta radnika na performanse algoritma, potrebno je detaljnije razmotriti mehanizam raspodjele zadataka. Važan parametar u tom mehanizmu predstavlja parametar J_S, koji određuje obim posla koji se odjednom šalje radniku na izvršavanje (tj. broj jedinki koje se odjednom šalju radniku na evaluaciju). Gospodar iz populacije uzima J_S jedinki kojima je potrebna evaluacija i traži prvog slobodnog radnika kojem potom šalje odabrane jedinke na evaluaciju. Ukoliko su svi radnici zaposleni gospodar čeka dok neki od njih ne završi s poslom.

Kod sinkrone inačice algoritma, nakon što su sve jedinke odaslane na evaluaciju, gospodar čeka da svi radnici završe, nakon čega nastavlja s izvođenjem algoritma. Eventualne neučinkovitosti koje nastaju zbog nejednake procesorske snage pojedinih radnika znatno ovise o parametru J_S. U graničnom slučaju, kada je veličina posla $J_s = \frac{N}{S+1}$, svaki od radnika odjednom dobije ukupni posao će obaviti u trenutnoj iteraciji, čime je brzina evaluacije ograničena najsporijim računalom među radnicima. Ukoliko je veličina posla J_S zanemariva u odnosu na veličinu populacije, u jednoj iteraciji se svakom računalu više puta šalje skupina jedinki za evaluaciju, pa je utjecaj različitog kapaciteta radnika malen.

Asinkrona inačica algoritma radi vrlo slično sinkronoj sve do zadnjeg koraka, kada su sve jedinke razaslane radnicima. Nakon toga se odmah kreće u iduću iteraciju algoritma, bez čekanja da radnici završe posao. Kako nema čekanja, različita snaga radnika ne utječe direktno na brzinu izvršavanja algoritma. Međutim, slaba snaga nekog od radnika može utjecati na kvalitetu rezultata i brzinu konvergencije pošto u najgorem slučaju sve jedinke poslane na evaluaciju u zadnjem koraku mogu imati

zastarjelu vrijednost funkcije cilja. U najgorem slučaju je to $(S+1)\cdot J_s$ broj jedinki.

Kao i kod sinkronog algoritma, ako je J_S zanemariv u odnosu na veličinu populacije, nema velikog utjecaja snage pojedinih radnika na izvršavanje algoritma.

3.4.1.1 Ostvarenje ujednačenja opterećenja među radnicima unutar okruženja ECF

Evolutionary Computation Framework (u daljnjem tekstu *ECF*) je okruženje za razvoj rješenja temeljenih na evolucijskom računarstvu [6]. Okruženje se razvija na Fakultetu elektrotehnike i računarstva na Zagrebačkom sveučilištu. Kao i aplikacije koje ga koriste, pisano je u jeziku C++. Poboljšanja evolucijskih algoritama, kao i rješenje jednog problema strojnog učenja za potrebe ovog rada, ostvareni su u okviru okruženja ECF.

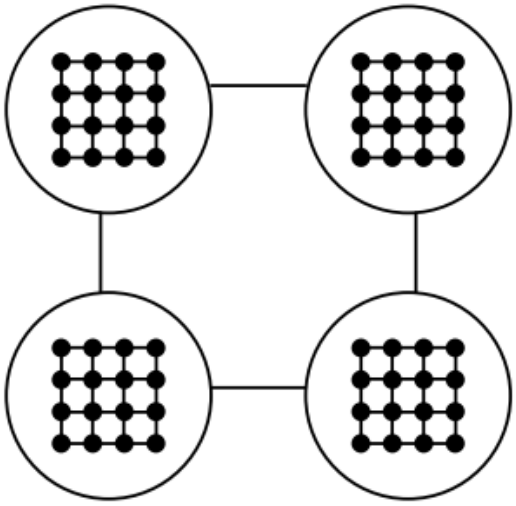
Prilagodba ostvarena za potrebe ovog rada je ujednačavanje opterećenja računala koja imaju ulogu radnika u globalnom paralelnom genetskom algoritmu. Okruženje ECF podržava sinkronu i asinkronu verziju algoritma, a spomenuto ujednačavanje opterećenja ostvareno je za obje varijante.

Mehanizam ujednačavanja opterećenja je izveden pomoću dinamičkog mijenjanja veličine posla J_S za pojedina računala - radnike. U početku algoritam radi poput inačice bez ujednačavanja opterećenja – svim radnicima se dodjeljuje posao neke inicijalne veličine J_S. Za vrijeme rada algoritma mjeri se trajanje izvršavanja posla za svakog pojedinog radnika. U skladu sa izmjerenim trajanjima, veličina posla za svakog radnika se prilagođava za onaj postotak za koji je prosječno vrijeme evaluacije jedne jedinke odgovarajućeg radnika različito od prosječnog vremena evaluacije svih radnika.

3.5 Hijerarhijski paralelni genetski algoritam

Hijerarhijski paralelni genetski algoritam koristi ulančavanje više razina izvršavanja, pri čemu se na svakoj razini neki izvršava neki model paralelnog genetskog algoritma (ili čak slijednog).

Mogućnost ostvarenja hijerarhijskog modela najviše ovisi o dostupnom sklopovlju pogodnom za izvršavanje različitih razina algoritma. Dobar primjer takvog sklopovlja je skupina masovno paralelnih računala međusobno spojenih u lokalnu mrežu. Tada se na višem nivou izvršava raspodijeljeni genetski algoritam koji subpopulacije raspoređuje na pojedina masovno paralelna računala, dok na nižem nivou svako masovno paralelno računalo koristi masovno paralelni genetski algoritam za evoluciju njemu dodijeljene subpopulacije. Slika 3.5 prikazuje opisanu hijerarhiju.



Slika 3.5 Ostvarenje hijerarhijskog paralelnog genetskog algoritma sa raspodijeljenim genetskim algoritmom na višem nivou i masovno paralelnim genetskim algoritmom na nižem

Hijerarhijski modeli znatno ubrzavaju izvođenje algoritma, pošto im je ubrzanje u odnosu na slijedni algoritam jednako umnošku ubrzanja pojedinih razina od kojih se hijerarhijski model sastoji. Nedostatak im je velik broj podesivih parametara (sadrže sve parametre algoritama na pojedinim razinama) i veliki zahtjevi nad dostupnim sklopovljem.

3.6 Hibridni paralelni genetski algoritam

Hibridni modeli paralelnih genetskih algoritama kombiniraju deterministične metode pretrage prostora rješenja sa nekim od uobičajenih modela paralelnih genetskih algoritama. Na kraju iteracije genetskog algoritma, trenutno najbolja rješenja u populaciji se koriste kao početne točke za dodatnu optimizaciju nekom od

determinističkih metoda za lokalno pretraživanje (gradijentne metode, metoda najbržeg spusta, simpleks postupak...).

Tablica 3.4 Pseudokod primjera hibridnog paralelnog genetskog algoritma

```
Hibridni_paralelni_genetski_algoritam(){
   generiraj_paralelno_populaciju_slučajnih_jedinki();
   dok (nije_zadovoljen_uvjet_završetka_evolucijskog_procesa){
      evaluiraj();          // evaluiraj paralelno svaku jedinku
      optimiraj_lokalno(); // za svaku jedinku paralelno obavi
                           // lokalno pretraživanje uporabom neke
                           // gradijentne metode
      selektiraj();         // selektiraj paralelno jedinku za
                           // reprodukciju među susjedima
      reproduciraj();       // obavi paralelno reprodukciju među
                           // odabranom jedinkom iz prethodnog
                           // koraka i vlastitom jedinkom
      optimiraj_lokalno(); // dijete paralelno optimiraj lokalno

      ako (je dijete bolje od roditelja){
         nadomjesti();     // nadomjesti paralelno vlastitu jedinku
                           // s boljom novodobivenom jedinkom
      }
   }
}
```

Slaba točka genetskih algoritama je fino podešavanje pronađenih rješenja. Uporabom hibridnog modela se taj problem eliminira i ubrzava se konvergencija genetskog algoritma ka rješenju.

3.7 Karakteristike paralelnih genetskih algoritama

Razlog paralelizacije genetskih algoritama je ubrzanje njihovog izvođenja. Međutim, stupanj ubrzanja nije jednak kod svih metoda paralelizacije niti za sve probleme koji se rješavaju. Ubrzanje koje dobijemo izvršavajući algoritam na više procesora je u idealnom slučaju jednako broju procesora, dok je u praksi taj broj uvijek manji.

Postoji još nekoliko zahtjeva koje bi idealni paralelni genetski algoritam trebao zadovoljavati [5]. Neki od njih su:

1) Broj parametara paralelnog genetskog algoritma treba biti što manji, kako bi se olakšalo podešavanje algoritma. U idealnom slučaju, paralelni algoritam nema dodatnih parametara, tj. ima samo one parametre koje ima odgovarajući slijedni algoritam.

2) Rješenje koje se dobije uporabom paralelnog genetskog algoritma treba biti iste kvalitete kao i rješenje dobiveno slijednim algoritmom. U idealnom slučaju, paralelni genetski algoritam ima potpuno ista svojstva kao odgovarajući slijedni algoritam.

Pošto su paralelni genetski algoritmi namijenjeni izvršavanju na dostupnim računalima i mrežama, u praksi se javlja još jedan zahtjev:

1) Model paralelnog genetskog algoritma treba biti pogodan za izvođenje na višeprocesorskom računalu sa zajedničkim radnim spremnikom ili na više umreženih računala.

Trenutno dostupni modeli paralelnih genetskih algoritama zadovoljavaju navedene zahtjeve u različitoj mjeri, ali niti jedan ih ne zadovoljava u potpunosti.

4. Primjena evolucijskih algoritama na strojno učenje

Strojno učenje je grana računarstva koja se bavi razvojem i usavršavanjem algoritama koji omogućuju računalima da unaprijede i poboljšaju vlastito ponašanje [7]. Poboljšanje ponašanja u željenom smjeru se postiže korištenjem empirijski dobivenih podataka, kao što su očitavanja s različitih senzora ili unaprijed pripremljena baza podataka.

Za računalni program *R* kažemo da uči iz iskustva *E* s obzirom na neku klasu zadaća *T* i mjeru učinkovitosti *P*, ako se njegovi rezultati u obavljanju zadaće *T* (mjereno s obzirom na kriterij *P*), poboljšavaju s iskustvom *E* [8].

4.1 Tehnike strojnog učenja

Kako su područja primjene strojnog učenja vrlo raznolika, postoji velik broj tehnika i algoritama kojima se postiže traženo učenje. Neke od najčešće korištenih tehnika su:

1) Učenje stablima odluke (engl. *decision tree learning*) – u ovoj tehnici učenja se kroz stablastu strukturu podataka zapisuju zapažanja o uzorcima u ulaznim podacima, kako bi se na temelju izgrađenog stabla mogao donijeti zaključak o ispravnom ponašanju pri susretu s nepoznatim ulazim podacima.
2) Učenje pravilima povezivanja (engl. *association rule learning*) – metoda koja otkriva skrivene povezanosti među varijablama iz velikog broja podataka.
3) Umjetne neuronske mreže (engl. *artificial neural networks*) – predstavljaju matematički model koji pokušava simulirati strukturu i ponašanje bioloških neuronskih mreža. Obično se koriste za modeliranje složenih odnosa između ulaznih i izlaznih podataka.
4) Genetsko programiranje (engl. *genetic programming*) – tehnika koja koristi genetske algoritme kako bi evoluirala *programe* sa ciljem pronalaženja programa koji će dobro obavljati željenu zadaću.
5) *Support Vector Machines* – tehnika u kojoj se zadani skup uzoraka za učenje označava se tako da svaki uzorak pripada jednoj od dviju kategorija. Zadatak naučenog sustava je da pokaže kojoj kategoriji pripada novi uzorak.

6) Grupiranje (engl. *clustering*) – tehnika koja svrstava prikupljene skupine podataka u grupe (engl. *clusters*) na način da su podaci u istoj grupi slični po nekom željenom kriteriju.
7) Bayesove mreže (engl. *Bayesian network*) – tehnika koja koristi usmjerene grafove bez ciklusa (engl. *directed acyclic graph*) kako bi modelirala skup slučajnih varijabli i uvjetne zavisnosti među njima.

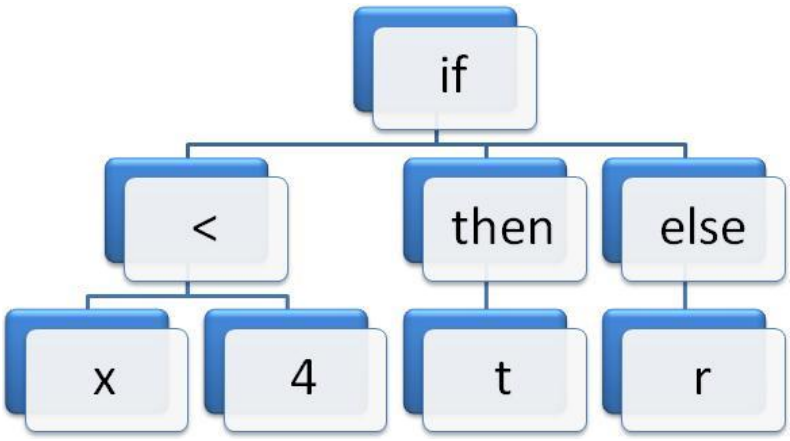
4.2 Genetsko programiranje kao tehnika strojnog učenja

Genetsko programiranje je specijalizacija genetskih algoritama u kojoj se kao jedinke populacije evoluiraju računalni programi koji izvršavaju neku željenu zadaću [9]. Genetski algoritam stvara populaciju računalnih programa i kroz određeni broj generacija poboljšava jedinke populacije korištenjem uobičajenih genetskih operatora – selekcije, križanja i mutacije. Kvaliteta pojedinog programa se vrednuje prema učinkovitosti u izvršavanju željene zadaće i u konačnici se najbolje ocijenjeni program predstavlja kao rezultat. Jednu od prvih primjena genetskog programiranja ostvario je John R. Koza tijekom 1980-tih godina. Međutim, zbog velike računalne zahtjevnosti genetskog programiranja, tek posljednjih godina ova tehnika nalazi široku primjenu.

Jedan od često korištenih pristupa strojnom učenju je razvijanje stabala odluke koja će sustav čije ponašanje želimo oblikovati upotrijebiti za odluku o odgovoru na neke ulazne podatke. Ova stabla odluke se vrlo prirodno mogu promatrati kao računalni programi koji za određene ulazne podatke računaju izlazne. Stoga je genetsko programiranje vrlo pogodno za rješavanje problema čije rješenje možemo prikazati stablima odluke.

4.2.1 Zapis kromosoma

Jedinka populacije u genetskom programiranju je funkcija predstavljena stablastom podatkovnom strukturom, nazvanom *stablo izraza* (engl. *expression tree*). Takav prikaz se jako dobro može iskoristiti za modeliranja stabala odluke.



Slika 4.1 Prikaz kromosoma koji predstavlja stablo odluke. Ulazni parametar na temelju kojeg se odluka provodi je x, dok t i r predstavljaju moguće izlaze stabla odluke

Općenito kažemo da je stablo izraza izgrađeno od *završnih* i *nezavršnih* simbola. Završni simboli se u stablu izraza javljaju kao unutarnji čvorovi, dok se nezavršni simboli javljaju kao listovi stabla. Nezavršni simboli su matematički operatori i funkcije koje imaju točno određen broj argumenata i jednu vrijednost kao rezultat. Izlazni rezultat svakog čvora predstavlja ulazni parametar njegovog roditeljskog čvora, dok vrijednost rezultata čitavog stabla izraza predstavlja izlaz programa ili „presudu“ stabla odluke kojeg kromosom predstavlja.

Tablica 4.1 Funkcija koja opisuje stablo odluke

```
function decision(x)
      if (x < 4)
      then return t
      else return r
end
```

Slika 4.1 prikazuje primjer kromosoma koji opisuje jedno stablo odluke. Varijabla *x* je ulazna varijabla na temelju koje se donosi odluka, dok su *t* i *r* konstante i moguće vrijednosti odluke. Tablica 4.1 sadrži funkciju koja opisuje prikazano stablo odluke.

4.2.2 Genetski operatori prilagođeni genetskom programiranju

Osnovni operatori u genetskom programiranju su uobičajeni operatori genetskih algoritama – križanje i mutacija. Koriste se inačice tih operatora prilagođene izvršavanu nad stablastim strukturama podataka.

Operator križanja se primjenjuje na jedinku tako što se jedan čvor stabla koje pripada jedinki zamijeni čvorom iz neke druge jedinke. Zamjena čvora u ovom slučaju znači zamjenu čitavog podstabla koje proizlazi iz tog čvora. Ovakvim pristupom se povećava učinkovitost operatora križanja, a nastala djeca su vrlo različita od njihovih roditelja.

Operator mutacije djeluje nad jednim stablom – programom. Mutacija može mijenjati samo podatke u čvoru stabla, ili može zamijeniti čitavo podstablo koje proizlazi iz mutiranog čvora. U oba slučaja je potrebno voditi računa o tome da dobiveni kromosom predstavlja ispravan program. Primjerice, pri mutaciji se mora paziti na broj parametara koji pojedine funkcije primaju, ili se funkcije moraju učiniti otpornima na neispravno korištenje.

4.2.3 Proširenje genetskog programiranja genskim izrazima

Genetsko programiranje ima karakteristiku značajnog usporenja pri povećanju dimenzija problema (tj. veličine kromosoma). Uzrok ovakvom usporenju leži u činjenici da stablasta struktura koja se koristi za prikaz kromosoma može imati zapis koji ne odražava ispravni program ili stablo odluke. Primjerice, slučajnim generiranjem jedinke ili primjenom nekog od genetskih operatora se dobije slučaj da čvor koji sadrži unarnu funkciju (npr. *sinus*) ima dva djeteta čvora. Osim što se uzaludno troši vrijeme na stvaranje ovakvih jedinki, potrebno je potrošiti još vremena na provjeru ispravnosti svake stvorene jedinke i po potrebi zamjenu.

Metoda optimiranja koja rješava navedene probleme zove se programiranje genskih izraza (engl. *gene expression programming*) [10]. Osnovna ideja je uvođenje tzv. genskih izraza kao jedinki u genetski algoritam, umjesto stabala koja predstavljaju funkcije kao genetskom programiranju. Genski izrazi su linearne strukture iz kojih se deterministčkim postupkom može dobiti program kojeg predstavljaju. Ovime se postiglo slično razdvajanje genotipa i fenotipa na kakvo nailazimo u prirodi. Važna karakteristika genskih izraza je da svaki genski izraz koji se može pojaviti pri radu genetskog algoritma, nakon izračuna njegovog fenotipa, predstavlja ispravan program kojeg se može evaluirati i ocijeniti.

5. Kartaška igra Ajnc

Ajnc (engl. *Blackjack*) je popularna kartaška igra na sreću. Jedna je od najpopularnijih igara u kockarnicama diljem svijeta. Ajnc je igra usporedbe karata koja se igra sa jednim do osam špilova, bez uporabe džokera. Poznata je kao jedna od rijetkih igara na sreću protiv kuće u kojoj vješt igrač može steći statističku prednost pred kockarnicom.

Slika 5.1 Izvučene karte u igri Ajnc s ukupnom vrijednošću od 21. boda

Cilj igre je izvlačenjem karata postići ukupnu vrijednost izvučenih karata veću od vrijednosti djelitelja, a da se pri tome ne prijeđe 21 bod.

5.1 Tijek igre i pravila

Ajnc je igra brojanja karata u kojoj je cilj postići veći broj bodova od djelitelja. Karte se broje na sljedeći način:

1) Karte s brojevima vrijede onoliko bodova koliki je broj na karti
2) Dečko, dama i kralj vrijede 10 bodova

3) As vrijedi 11 bodova ili 1 bod, na način da igrač ima najpovoljniji mogući broj bodova. Ako igrač ima asa od 11 bodova, tada mu je zbroj *mekan* (engl. *soft*), a ako nema asa ili ima asa od jednog boda, tada mu je zbroj *tvrd* (engl. *hard*)

Prije početka igre svaki igrač ulaže neki iznos novca. Djelitelj tada dijeli svakom igraču dvije karte licem gore, a sebi jednu kartu licem gore i jednu licem dolje. Igrači u tom trenutku imaju nekoliko opcija:

1) Opcija *hit* – odabirom ove opcije igrač dobiva još jednu kartu
2) Opcija *stand* – odabirom ove opcije igrač ne dobiva dodatne karte i njegov red završava. Svi igrači moraju odabrati *stand* ili prijeći 21 bod da bi djelitelj mogao početi sa svojom igrom.
3) Opcija *double down* – odabirom ove opcije igrač dobiva još jednu kartu i njegov red završava. Pri tome je dužan dodatno uložiti iznos jednak svom početnom ulogu. Ovaj izbor je dostupan samo igračima sa mekanim zbrojevima i tvrdim zbrojevima iznosa 10 i 11.
4) Opcija *split* – ova opcija je igraču dostupna ako ima dvije iste karte. Odabirom ove opcije ih može razdvojiti, pri čemu mora dodatno uložiti iznos jednak početnom ulogu. Nakon toga igra svaku od razdvojenih karata kao zaseban igrač.

Ukoliko igrač na dijeljenju nakon bilo koje akcije dobije zbroj 21, njegov red se automatski završava. U tom slučaju može imati remi samo ako djelitelj ima 21, inače pobjeđuje. Međutim, ako je dobio 21 isprve (mora imati asa i kartu koja vrijedi 10 bodova), igrač ima *ajnc*, te dobiva dobitak isplaćen u omjeru 3:2 naspram uloženog novca. Igrač ne dobiva ništa samo ako djelitelj također ima ajnc. Tada se dogodi remi.

Ako djelitelj ima više bodova nego igrač, igrač gubi. Ako je obrnuto, igrač dobiva i dobitak mu se isplaćuje u omjeru 1:1. Ako igrač i djelitelj imaju jednak broj bodova, igrač dobiva svoj ulog natrag, ali bez dobitka, osim kada jedan od njih ima ajnc, a drugi normalnih 21. Tada onaj s ajncom dobiva.

Djeliteljeva je igra strogo propisana i u njoj ne postoji element vještine jer djelitelj ne može donositi nikakve odluke o igri. Kad svi igrači završe svoj red, djelitelj okreće svoju kartu licem nagore. Zatim uzima nove karte sve dok ima 16 ili manje.

Ako dobije 17 - 21, djelitelj stane i drugi igrači uspoređuju svoj broj bodova s njegovim. Ako ima 22 ili više, djelitelj je automatski izgubio i svi preostali igrači dobivaju.

5.2 Igranje Ajnca kao problem strojnog učenja

Promatrajući Ajnc kao igru jednog igrača protiv kuće postavlja se zanimljivo pitanje mogućnosti ostvarenja računalnog igrača koji bi postizao dobre rezultate. Matematička analiza vjerojatnosti dobitka u pojedinim situacijama i strategija igre koja se temelji na takvoj analizi je moguća, iako nije ni blizu jednostavna kako bi se to na prvi pogled moglo činiti.

Drugi pristup je pokušati razviti strategiju igre uporabom genetskog programiranja. Jedinke populacije će tada predstavljati različite strategije igre, te će se ocjenjivati po uspješnosti igre protiv simulirane kockarnice.

5.2.1 Prikaz strategije igranja u kromosomu

Strategija igranja Ajnca se prirodno može prikazati koristeći stabla odluke. U trenutku kada igrač mora birati između ponuđenih opcija, poznate su mu karte koje ima te jedna karta djelitelja. Ulazne varijable u stablo odluke će stoga biti:

1) Karta koju ima djelitelj K_d
2) Zbroj bodova koje ima igrač Z
3) Informacija o tome je li igračev zbroj tvrd ili mekan (binarna varijabla) M

Strategija igre će biti predstavljena korištenjem dva različita stabla odluke od kojih oba imaju gore navedene ulazne varijable, a kao rezultat binarnu vrijednost:

1) Prvo i glavno stablo odluke daje odgovor na pitanje hoće li igrač odabrati opciju *hit* ili *stand*. Ukoliko, nakon izvlačenja karte, igrač ponovo ima takvu opciju stablo odluke se ponovo evaluira sa osvježeni ulaznim varijablama.
2) Drugo stablo odluke se koristi kada igrač ima mogućnost odabrati opciju *double down.* Ukoliko se ponuđena opcija ne odabere, igrač dalje bira između opcija *hit* i *stand*, za što se koristi prvo stablo odluke.

Opcija *split* će se u ostvarenju unutar ovog rada ignorirati.

5.2.2 Evaluacija jedinki – testiranje računalnog igrača

Funkcija cilja pri evoluiranju strategije igranja Ajnca mora vjerno odražavati vještinu igranja računalnog igrača. Najjednostavniji način njene procjene je simulacija određenog broja partija, sa slučajno dodijeljenim kartama i praćenje uspješnosti igrača.

Ovakav pristup pripada klasi tzv. *Monte Karlo metoda* [12]. Toj klasi metoda pripadaju sve tehnike u kojima se iz velikog skupa kombinacija ulaznih varijabli slučajno odabere neki manji skup, obavi deterministička simulacija koristeći odabrani ulazni skup podataka, te konačni rezultat oformi koristeći rezultate pojedinačnih simulacija.

U slučaju igre Ajnc, moguće kombinacije ulaznih varijabli su sve permutacije igraćeg špila (koji se može sastojati od jednog do osam običnih špilova), a simulacije se obavljaju na određenom broju slučajno odabranih permutacija. Povećavajući broj simuliranih partija možemo s većom pouzdanošću procijeniti vještinu računalnog igrača. Međutim, to dovodi do usporenja rada genetskog algoritma zbog dužeg vremena računanja funkcije cilja. Podešavanjem broja simuliranih partija možemo, uz malo žrtvovanje pouzdanosti rezultata, precizno podešavati zahtjevnost operatora evaluacije u odnosu na ostale dijelove genetskog algoritma.

Za potrebe evaluacije se može pretpostaviti da igrač ima neograničeno mnogo novaca za ulaganje te da u svakoj partiji uloži isti iznos. U tom slučaju, ocjenu uspješnosti računalnog igrača predstavlja prosječni učinak tj. zarada po jednoj partiji.

5.2.3 Ostvarenje unutar okruženja ECF

Razvoj računalnog igrača Ajnca korištenjem genetskog programiranja ostvaren je unutar okruženja ECF. Evaluacijska funkcija računalnog igrača ostvarena je korištenjem programskog jezika C++. Pri pokretanju genetskog algoritma, okruženje ECF pruža gotova ostvarenja različitih vrsta kromosoma i pripadnih genetskih operatora, te korištenjem zadane evaluacijske funkcije dolazi do rješenja problema.

Za potrebe razvoja računalnog igrača Ajnca korištena su dva stablasta prikaza kromosoma, jedan od direktno podržanih vrsta prikaza u okruženju ECF. Nezavršni

simboli za izgradnju kromosoma su logičke funkcije *nand* (negirana konjunkcija) i *nor* (negirana disjunkcija). Završnih simbola ima mnogo i uključuju skup simbola *dealerScoreLessOrEqualThanN*, gdje N poprima vrijednosti od 0 do 21, a pripadni završni simbol ima istinitu vrijednost ako i samo ako je djeliteljev zbroj karata manji ili jednak broju N. Analogno tome, završne simbole čine još i skupovi simbola:

- *dealerScoreGreaterOrEqualThanN*
- *playerScoreLessOrEqualThanN*
- *playerScoreGreaterOrEqualThanN*

Preostali završni simbol je *playerSoftCards*, koji govori je li igračev zbroj karata tvrd ili mekan.

Kombiniranjem opisanih završnih simbola koji poprimaju binarne vrijednosti i logičkih funkcija *nand* i *nor*, evaluacijom oba kromosoma dolazi se do binarnih vrijednosti koje određuju ponašanje računalnog igrača za vrijeme igre. Slika 5.2 prikazuje primjer opisanih kromosoma.

```
Stablo odluke za opcije hit i stand
  nand
    playerScoreGreaterOrEqualThan15
    nand
        playerSoftCards
        playerSoftCards

Stablo odluke za opciju double down
   nor
      playerScoreGreaterOrEqual14
      nor
         dealerScoreGreaterOrEqualThan7
         playerSoftCards
```

Slika 5.2 Primjer genotipa za donošenje odluka u igri Ajnc, sastavljen od dva stabla

6. Mjerenje značajki paralelnih genetskih algoritama i analiza rezultata

U ovom poglavlju prikazani su rezultati mjerenja učinkovitosti paralelizacije genetskih algoritama. Mjerenja su provedena korištenjem sinkrone i asinkrone inačice globalnog paralelnog genetskog algoritma na problemu razvoja računalnog igrača Ajnca. Prikazani su rezultati za različite kombinacije broja procesora i vremenske zahtjevnosti evaluacijske funkcije.

6.1 Pokretanje genetskog algoritma

6.1.1 Zadavanje parametara genetskog algoritma u okruženju ECF

Okruženje ECF omogućuje jednostavno podešavanje parametara genetskog algoritma korištenjem konfiguracijske datoteke u formatu XML. Kroz konfiguracijsku datoteku moguće je podesiti vrstu i detalje prikaza kromosoma, tehniku paralelizacije genetskog algoritma i različite parametre vezane uz odabranu tehniku, operator i parametre selekcije jedinki, veličinu populacije, uvjet zaustavljanja genetskog algoritma i još mnoge druge parametre. Slika 6.1 prikazuje neke od opisanih parametara.

```
<ECF>
      <Algorithm>
            <SteadyStateTournament>
                  <Entry key="tsize">3</Entry>
            </SteadyStateTournament>
      </Algorithm>

      <Genotype>
            <Tree>
                  <Entry key="maxdepth">5</Entry>
                  <Entry key="mindepth">3</Entry>
                  <Entry key="functionset">nand nor</Entry>
                  <Entry key="terminalset">term1 term2</Entry>
            </Tree>
      </Genotype>

      <Registry>
            <Entry key="randomizer.seed">0</Entry>
            <Entry key="population.size">100</Entry>
            <Entry key="population.demes">1</Entry>
            <Entry key="mutation.indprob">0.3</Entry>
            <Entry key="term.stagnation">200</Entry>
            <Entry key="log.level">3</Entry>
            <Entry key="log.filename">log.txt</Entry>
            <Entry key="milestone.filename">out.txt</Entry>
            <Entry key="milestone.interval">0</Entry>

            <Entry key="blackjack.gamecount">5000</Entry>
            <Entry key="blackjack.cardsseed">0</Entry>
      </Registry>
</ECF>
```

Slika 6.1 Primjer konfiguracijske datoteke okruženja ECF

Okruženje ECF također omogućuje korištenje korisničkih parametara koji mogu poslužiti za prilagodbu načina rada evaluacijske funkcije. Pri ostvarenju evaluacijske funkcije za procjenu kvalitete igrača Ajnca, upotrjebljen je parametar koji određuje broj partija Ajnca koje će računalni igrač odigrati prije procjene njegove sposobnosti. Mijenjanje ovog parametra znatno utječe na vremensku zahtjevnost evaluacijske funkcije.

6.1.2 Korištenje MPICH2 okruženja za paralelizaciju

Paralelna verzija okruženja ECF omogućuje pokretanje neke od varijanti paralelnih genetskih algoritama bez potrebe za mijenjanjem programskog ostvarenja evaluacijske funkcije za zadani problem. Odabir paralelnog genetskog algoritma se

ostvaruje kroz konfiguracijsku datoteku. Za pokretanje genetskog algoritma na više računala koristi se okruženje MPICH2 [13]. MPICH2 je ostvarenje sučelja za razmjenu poruka (engl. *message passing interface*, u daljnjem tekstu *MPI*). Okruženje ECF koristi MPI za komunikaciju i sinkronizaciju među procesima pokrenutima na više računala. Pokretanje programa unutar okruženja MPICH2 se postiže jednostavnim zadavanjem imena izvršne datoteke koju želimo pokrenuti (i njenim mogućim parametrima) i adresama računala na kojima želimo pokrenuti zadani program. Pri tome okruženje MPICH2 i sama izvršna datoteka moraju biti prisutni na svim zadanim računalima.

6.1.3 Korištenje Amazon EC2 platforme

Amazon Elastic Compute Cloud (u daljnjem tekstu *EC2*) je usluga koja omogućuje zakup velikog broja virtualnih računala na zahtjev, kao i njihovo otpuštanje kada više nisu potrebna [14]. Korištenjem jednostavnog grafičkog sučelja moguće je pokrenuti računala različitih performansi, dobiti pregled nad trenutno pokrenutim računalima i ugasiti mogući višak.

Dostupno je nekoliko modela virtualnih računala, s različitim omjerom procesorske snage, memorije i diskovnog prostora. Podignuta računala naplaćuju se po satu korištenja, s različitom jediničnom cijenom za pojedine modele. Jednostavnost ovakvog modela iskorištena je za lako mjerenje karakteristika paralelizacije genetskih algoritama na većem broju procesora i u heterogenom procesorskom okruženju.

6.2 Testirane inačice genetskog algoritma

Mjerenja različitih karakteristika paralelizacije genetskih algoritama provedena su na dvije inačice globalnog paralelnog genetskog algoritma ostvarene u okruženju ECF. Prva od njih je sinkroni algoritam sa generacijskom selekcijom. Kao i kod svih modela globalnih paralelnih genetskih algoritama, uloge računala na kojima se izvršava dijele se na jednog gospodara i više radnika. Na računalima radnicima se paralelno obavlja evaluacija jedinki unutar populacije, dok se ostali genetski operatori izvršavaju na računalu gospodaru. Algoritam spada u sinkrone jer gospodar čeka da radnici obave čitav posao prije prelaska u sljedeću iteraciju genetskog algoritma. Slika 6.2 prikazuje

pseudokod sinkronog globalnog paralelnog genetskog algoritma s generacijskom selekcijom.

```
Sinkroni_globalni_paralelni_genetski_algoritam(){
   generiraj_i_evaluiraj_početnu_populaciju_jedinki();

   dok nije_zadovoljen_uvjet_završetka_evolucijskog_procesa {
      odaberi_najbolje_jedinke_iz_populacije()
      stvori_novu_populaciju_križanjem_odabranih_jedinki()
      mutiraj_populaciju()

     dok ima_jedinki_za_evaluaciju {
        pronađi_slobodnog_radnika()
        pošalji_skup_jedinki_na_evaluaciju()
     }

     čekaj_dok_svi_radnici_ne_završe()
   }
}
```

Slika 6.2 Pseudokod sinkronog globalnog paralelnog genetskog algoritma s generacijskom selekcijom

Druga inačica globalnog paralelnog genetskog algoritma je asinkroni algoritam s eliminacijskom selekcijom. Kod ove verzije algoritma, gospodar ne čeka da radnici završe s evaluacijom razaslanih jedinki, već odmah nakon dodjele posla počinje s izvođenjem sljedeće iteracije. Nedostatak ovog pristupa je mogućnost da prilikom postupka selekcije u idućoj iteraciji neke jedinke imaju pogrešnu vrijednost funkcije cilja, tj. onu zaostalu iz prošle iteracije. Slika 6.3 prikazuje pseudokod asinkronog globalnog genetskog algoritma s turnirskom eliminacijskom selekcijom.

```
Asinkroni_globalni_paralelni_genetski_algoritam(){
   generiraj_i_evaluiraj_početnu_populaciju_jedinki();

   dok nije_zadovoljen_uvjet_završetka_evolucijskog_procesa {
     dok nije_stvoreno_novih_jedinki_kolika_je_veličina_populacije {
        zamijeni_skup_jedinki_turnirskim_odabirom()
        pošalji_skup_jedinki_na_evaluaciju()
     }
     //kada smo obavili zadani broj turnira nema čekanja
     //na radnike da završe posao već algoritam odmah
     //prelazi na iduću iteraciju
   }
}
```

Slika 6.3 Pseudokod asinkronog globalnog paralelnog genetskog algoritma s turnirskom eliminacijskom selekcijom

6.3 Utjecaj složenosti evaluacijske funkcije na paralelizaciju

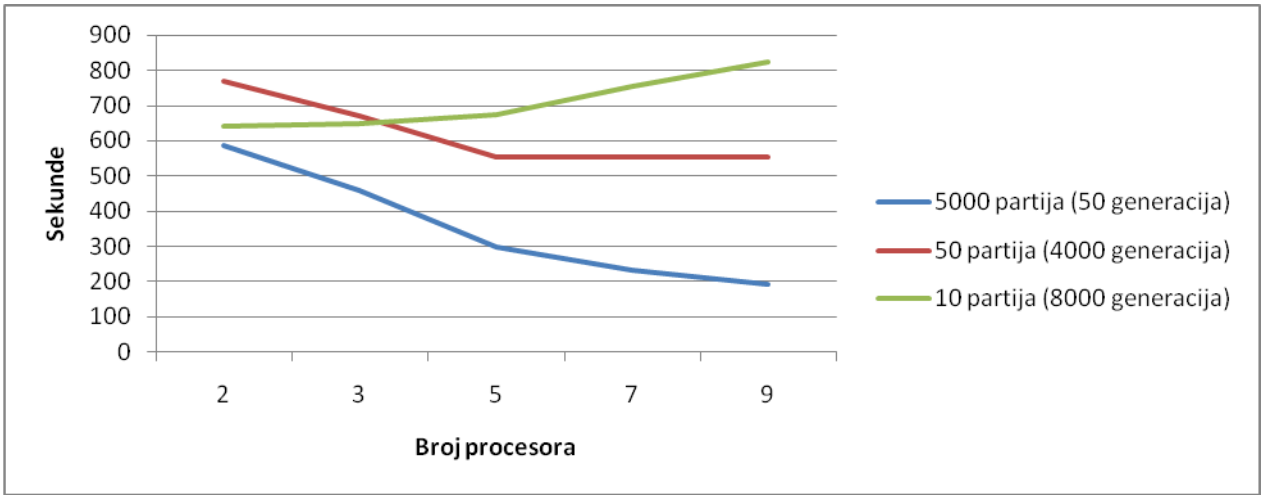
Analizom brzine izvršavanja genetskog algoritma za različite složenosti evaluacijske funkcije i utjecaja povećanja broja procesora na brzinu izvršavanja možemo odrediti za kakve evaluacijske funkcije povećavanje broja procesora pruža željeno ubrzanje.

Genetski algoritam je pokrenut na promjenjivom broju dvoprocesorskih računala. Na računalu na kojem se nalazi proces koji ima ulogu gospodara je bio pokrenut samo jedan proces, dok su na ostalim računalima bili pokrenuti jedan ili dva procesa. Uvjet završetka rada algoritma je zadani broj iteracija, različitih za svaki stupanj složenosti evaluacijske funkcije, kako bi trajanje za različite stupnjeve bilo međusobno usporedivo. Svako pojedinačno mjerenje je obavljeno tri puta, nakon čega je uzeto njihovo prosječno trajanje. Korištena veličina populacije je 600 jedinki, a broj jedinki koje se odjednom šalju radniku na obradu je 4.

Mjerenja su provedena korištenjem 3 različita stupnja složenosti evaluacijske funkcije u kojima se koristi 5000, 50 i 10 partija Ajnca za procjenu učinkovitosti računalnog igrača. 5000 partija predstavlja vremenski vrlo zahtjevnu evaluacijsku funkciju, dok vrijeme simuliranja 10 partija nije veliko u odnosu na trajanje ostalih dijelova genetskog algoritma.

6.3.1 Sinkroni algoritam

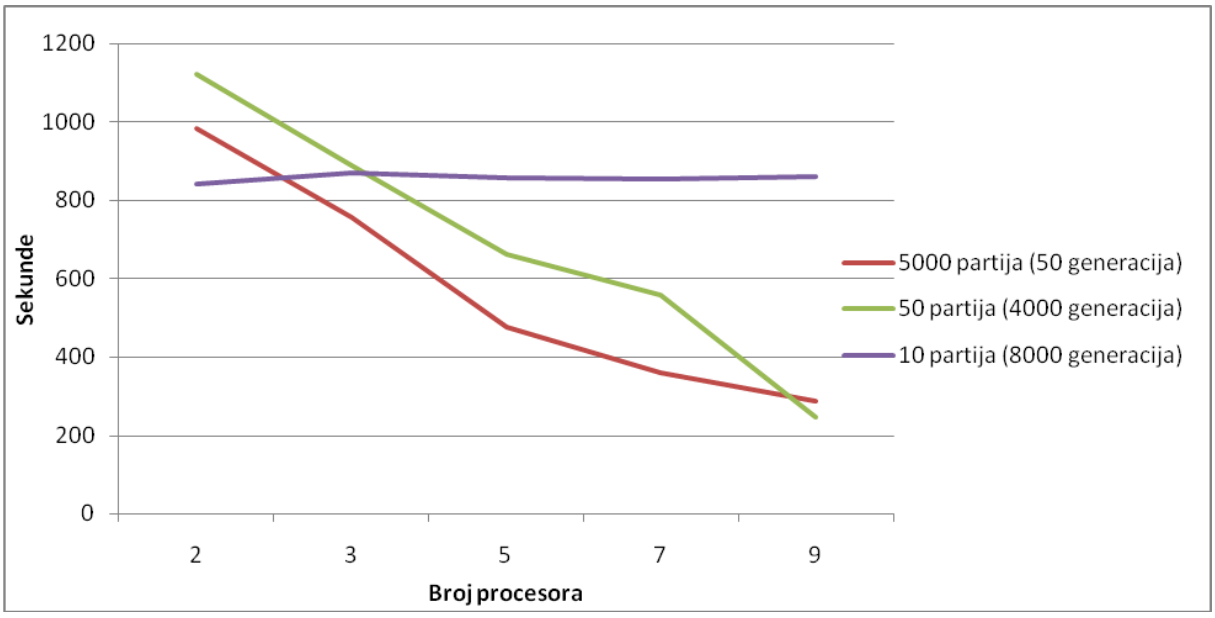
Slika 6.4 prikazuje vrijeme trajanja sinkronog globalnog paralelnog genetskog algoritma u ovisnosti o broju procesora za različite stupnjeve složenosti evaluacijske funkcije. Za vrlo zahtjevnu evaluacijsku funkciju, dodavanje procesora pozitivno utječe na vrijeme izvršavanja. Za srednje zahtjevnu funkciju, dodavanje procesora pomaže do određenog broja, nakon čega vrijeme izvršavanja stagnira ili se čak pogoršava. U slučaju zanemarivo zahtjevne evaluacijske funkcije, vrijeme izvršavanja se ne može poboljšati paralelizacijom i vrlo lako se pogorša.



Slika 6.4 Utjecaj složenosti evaluacijske funkcije na paralelizaciju sinkronog globalnog paralelnog genetskog algoritma

6.3.2 Asinkroni algoritam

Slika 6.5 prikazuje vrijeme trajanja asinkronog globalnog paralelnog genetskog algoritma u ovisnosti o broju procesora za različite stupnjeve složenosti evaluacijske funkcije. Utjecaj dodavanja broja procesora je sličan kao i kod sinkrone inačice algoritma. Razlika je vidljiva u tome da nema pogoršanja brzine izvođenja uslijed dodavanja procesora, već se u najgorem slučaju dobije stagnacija.



Slika 6.5 Utjecaj složenosti evaluacijske funkcije na paralelizaciju asinkronog globalnog paralelnog genetskog algoritma

6.4 Utjecaj ujednačenja opterećenja među radnicima na paralelizaciju

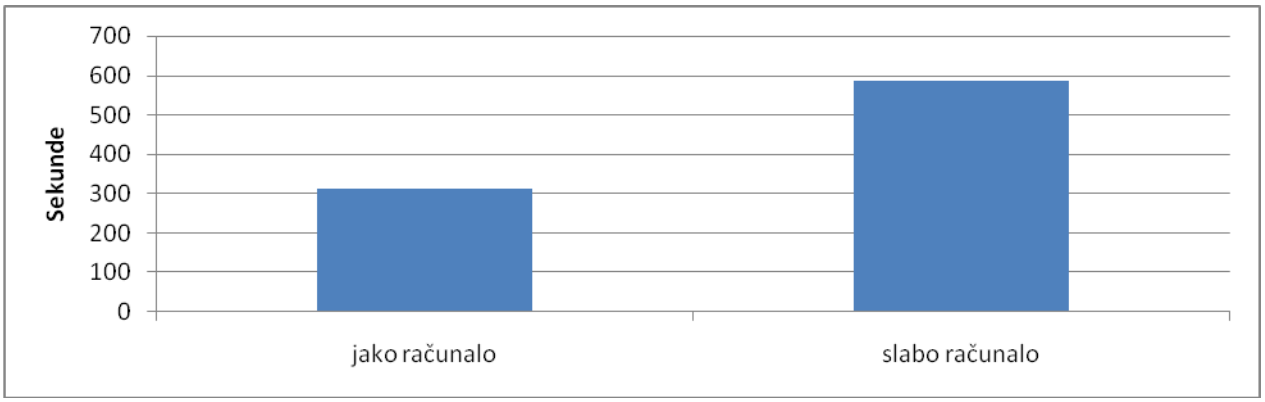
Vrlo je čest slučaj da je za izvršavanje paralelnog genetskog algoritma dostupno nekoliko računala međusobno različitih performansi. Kako u uobičajenom ostvarenju globalnog paralelnog genetskog algoritma gospodar dodjeljuje svim radnicima jednak broj jedinki za evaluaciju, postoji mogućnost da kapacitet jačih računala ne bude u potpunosti iskorišten.

Ujednačenjem opterećenja među radnicima može se izbjeći opisani problem. Za vrijeme izvršavanja genetskog algoritma, gospodar prati prosječno vrijeme evaluacije za svakog radnika i na temelju dobivenih podataka prilagođava broj jedinki koje će dotičnom radniku poslati na evaluaciju. Slika 6.6 prikazuje pseudokod opisanog mehanizma.

```
Globalni_paralelni_genetski_algoritam_sa_ujednačenjem_opterećenja(){
   generiraj_početnu_populaciju_jedinki();
   dok nije_zadovoljen_uvjet_završetka_evolucijskog_procesa {

      //provjeravaju se svi slobodni radnici
      dok ima_jedinki_za_evaluaciju {
         za svakog_radnika {
            ako radnik_slobodan i ima_poslane_jedinke {
               zapiši_trenutno_vrijeme()
               izračunaj_vrijeme_evaluacije_jedinke()
            }
         }

         radnik = odaberi_slobodnog_radnika()
         broj_jedinki_za_evaluaciju =
            veličina_posla *
            (prosječno_vrijeme_evaluacije_jedinke(radnik) /
             prosječno_vrijeme_evaluacije_jedinke_svih_radnika())

         pošalji_na_evaluaciju(radnik, broj_jedinki_za_evaluaciju)
         zapiši_vrijeme_slanja_i_broj_jedinki(
                  radnik,
                  broj_jedinki_za_evaluaciju)
      }

      selektiraj();
      krizaj();
      mutiraj();
   }
}
```

Slika 6.6 Pseudokod algoritma za ujednačenje opterećenja među radnicima

Mjerenje učinkovitosti ujednačenja opterećenja provedeno je koristeći dvije različite jednoprocesorske konfiguracije iz okruženja Amazon EC2. Genetski algoritam je pokrenut na jednom slabom računalu s ulogom gospodara, te po dva slaba i dva jaka računala s ulogama radnika. Mjereno je vrijeme trajanja genetskog algoritma kroz 400 generacija, pri čemu je korištena vrlo zahtjevna evaluacijska funkcija. Također, mjerenja su obavljena sa različitim omjerom broja jedinki koje se odjednom šalju radniku na obradu naspram ukupne veličine populacije. Svako pojedinačno mjerenje je obavljeno tri puta, nakon čega je uzeto njihovo prosječno trajanje.



Slika 6.7 Prikaz omjera snaga pojedinih računala u heterogenom procesorskom okruženju

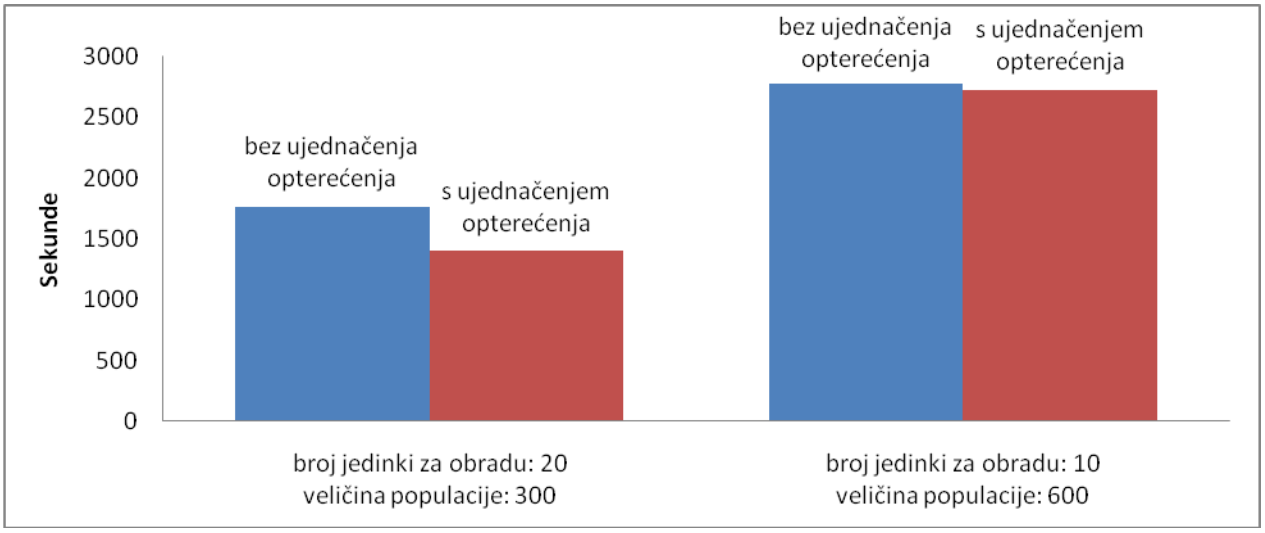
Slika 6.7 prikazuje relativni odnos snaga odabranih konfiguracija, dobiven mjerenjem vremena potrebnog za evaluaciju 100 jedinki kroz 40 generacija na obje vrste računala. Tablica 6.1 prikazuje iste podatke u tabličnom obliku.

Tablica 6.1 Prikaz omjera snaga pojedinih računala u heterogenom procesorskom okruženju

	Trajanje (sekunde)	**Postotno ubrzanje (sekunde)**
Jako računalo	312	46.85%
Slabo računalo	587	0%

6.4.1 Sinkroni algoritam

Slika 6.8 prikazuje utjecaj ujednačenja opterećenja na trajanje sinkronog genetskog algoritma za različite omjere broja jedinki koje se odjednom šalju radniku na evaluaciju naspram veličine populacije. Vidljivo je da postoji poboljšanje vremena izvršavanja uslijed ujednačenje opterećenja u slučaju kada je veličina posla za pojedinog radnika dovoljno velika u odnosu na veličinu populacije. Kada je veličina posla jako mala u odnosu na veličinu populacije, ubrzanje je zanemarivo. Tablica 6.2 daje podatke o utjecaju ujednačenja opterećenja na trajanje sinkronog genetskog algoritma u tabličnom obliku.



Slika 6.8 Utjecaj ujednačenja opterećenja među radnicima na brzinu izvršavanja sinkronog globalnog paralelnog genetskog algoritma

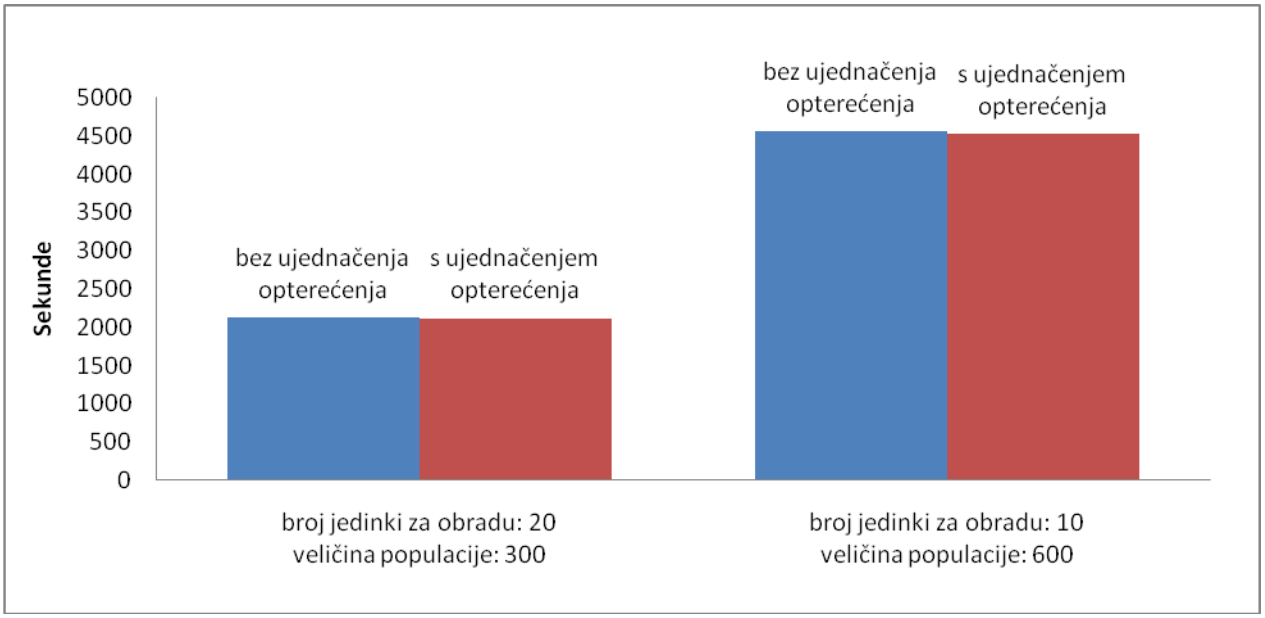
Tablica 6.2 Utjecaj ujednačenja opterećenja među radnicima na brzinu izvršavanja sinkronog globalnog paralelnog genetskog algoritma

	Broj jedinki za obradu: 20 Veličina populacije: 300		**Broj jedinki za obradu: 10 Veličina populacije: 600**	
	Trajanje (sekunde)	**Postotno ubrzanje**	**Trajanje (sekunde)**	**Postotno ubrzanje**
Bez ujednačenja opterećenja	1768	0%	2780	0%
S ujednačenjem opterećenja	1402	20.7%	2727	1.9%

6.4.2 Asinkroni algoritam

Slika 6.9 prikazuje utjecaj ujednačenja opterećenja na trajanje asinkronog genetskog algoritma za različite omjere broja jedinki koje se odjednom šalju radniku na evaluaciju naspram veličine populacije. Za razliku od sinkronog algoritma, nema poboljšanja trajanja uslijed ujednačenja opterećenja. Takvo ponašanje je i očekivano, s obzirom da asinkroni algoritam ne čeka sve radnike da završe s obradom prije nego krene na iduću generaciju. Zbog toga ne može doći do situaciju u kojoj jači

radnici besposleno miruju, dok gospodar čeka slabije radnike da završe. Tablica 6.3 daje podatke o utjecaju ujednačenja opterećenja na trajanje asinkronog genetskog algoritma u tabličnom obliku.



Slika 6.9 Utjecaj ujednačenja opterećenja među radnicima na brzinu izvršavanja asinkronog globalnog paralelnog genetskog algoritma

Tablica 6.3 Utjecaj ujednačenja opterećenja među radnicima na brzinu izvršavanja asinkronog globalnog paralelnog genetskog algoritma

	Broj jedinki za obradu: 20 Veličina populacije: 300		**Broj jedinki za obradu: 10 Veličina populacije: 600**	
	Trajanje (sekunde)	**Postotno ubrzanje**	**Trajanje (sekunde)**	**Postotno ubrzanje**
Bez ujednačenja opterećenja	2129	0%	4560	0%
S ujednačenjem opterećenja	2122	0.33%	4524	0.79%

6.5 Mjerenje učinkovitosti različitih inačica genetskog algoritma

6.5.1 Metodologija mjerenja

Mjerenje brzine izvršavanja algoritma na različitim konfiguracijama za neki zadani broj iteracija pruža zanimljive informacije o učinkovitosti paralelizacije genetskih algoritama. Međutim, najbolja mjera učinkovitosti bilo kojeg genetskog algoritma je ponašanje vrijednosti funkcije cilja kroz vrijeme, te mjerenje brzine dostizanja neke željene vrijednosti funkcije cilja.

Kao mjera uspješnosti paralelizacije genetskih algoritama koristi se *relativna učinkovitost paralelnog algoritma* i *relativno ubrzanje paralelnog algoritma.* Relativna učinkovitost algoritma definira se kao:

$$E_{rel} = \frac{T_1}{P * T_P}$$

gdje je T_1 trajanje algoritma na jednom procesoru, a T_P trajanje na P procesora. Relativno ubrzanje algoritma definirano je kao:

$$S_{rel} = E_{rel} * P$$

Veličine su označene kao relativne jer različiti genetski algoritmi, ovisno o parametrima algoritma i zahtjevnosti problema koji se rješava, mogu imati različito trajanje na jednom procesoru.

Kako je ponašanje genetskog algoritma podložno slučajnosti, brzina postizanja zadovoljavajućeg rezultata može jako varirati. Zbog toga se za analizu ponašanja genetskog algoritma koristi prosječna vrijednost funkcije cilja za nekoliko izvršavanja algoritma, promatrana kroz vrijeme. Tablica 6.4 prikazuje primjer prosječne vrijednosti funkcije cilja nakon 900 sekundi izvršavanja, dobivenu iz 4 pokretanja algoritma.

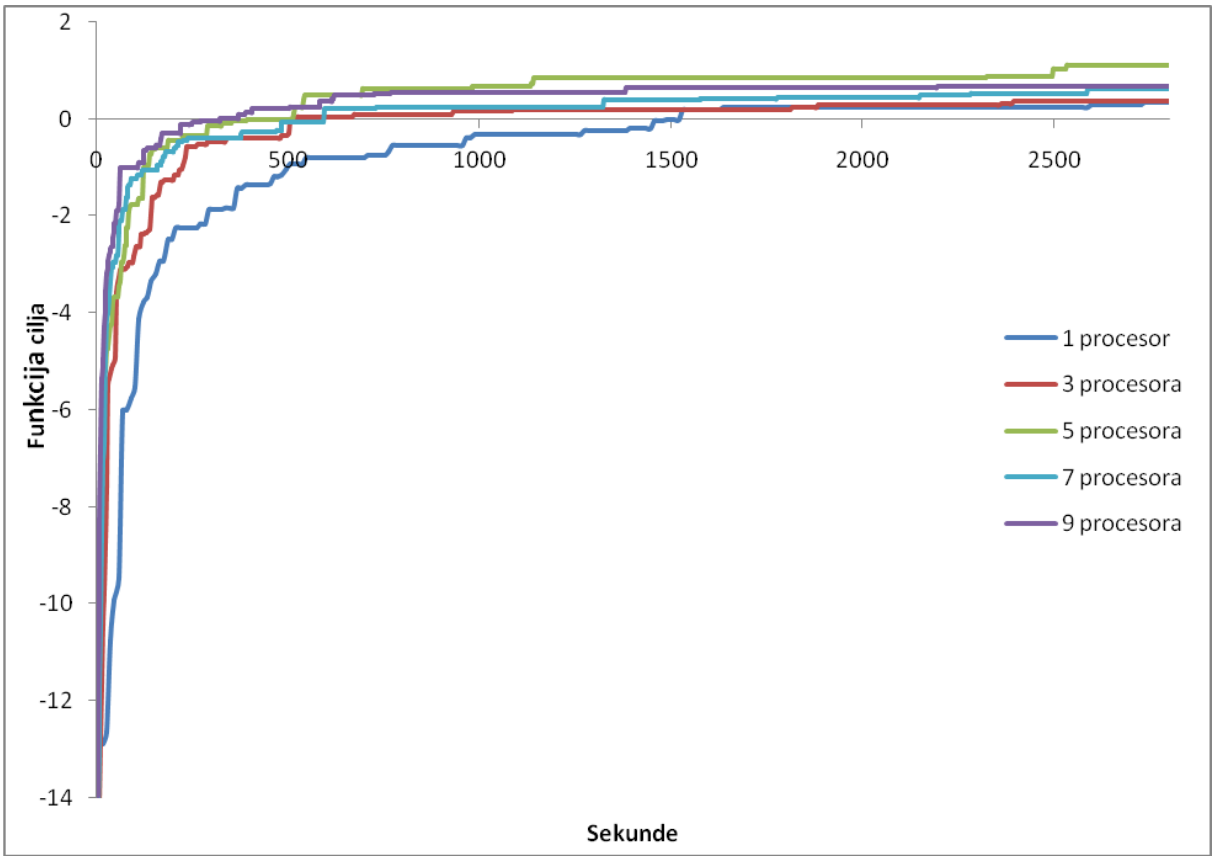
Tablica 6.4 Prikaz prosječne vrijednosti funkcije cilja za 4 mjerenja nakon 900 sekundi izvršavanja algoritma

	Mjerenje 1	Mjerenje 2	Mjerenje 3	Mjerenje 4	Prosjek
300 sekundi	1.2	1.4	0.5	1.1	1.05
600 sekundi	1.5	1.8	0.9	1.4	1,4
900 sekundi	1.6	1.8	1.0	1.5	1,475
1200 sekundi	1.6	1.9	1.3	1.7	1,625

Mjerenje učinkovitosti genetskih algoritama provedeno je koristeći 5 dvoprocesorskih računala iz okruženja Amazon EC2. Na računalu na kojem se nalazi proces koji ima ulogu gospodara je bio pokrenut jedan proces, dok su na ostalim računalima bili pokrenuti jedan ili dva procesa. Mjerenje je provedeno za različitim kombinacijama od ukupno 1, 3, 5, 7 odnosno 9 procesora. Uvjet završetka rada algoritma je ukupno trajanje od 2800 sekundi. Svako pojedinačno mjerenje je ponovljeno pet puta, iz čega je potom izračunata prosječna vrijednost funkcije cilja u svakom trenutku. Korištena veličina populacije je 300 jedinki, a broj jedinki koje se odjednom šalju radniku na obradu je 4.

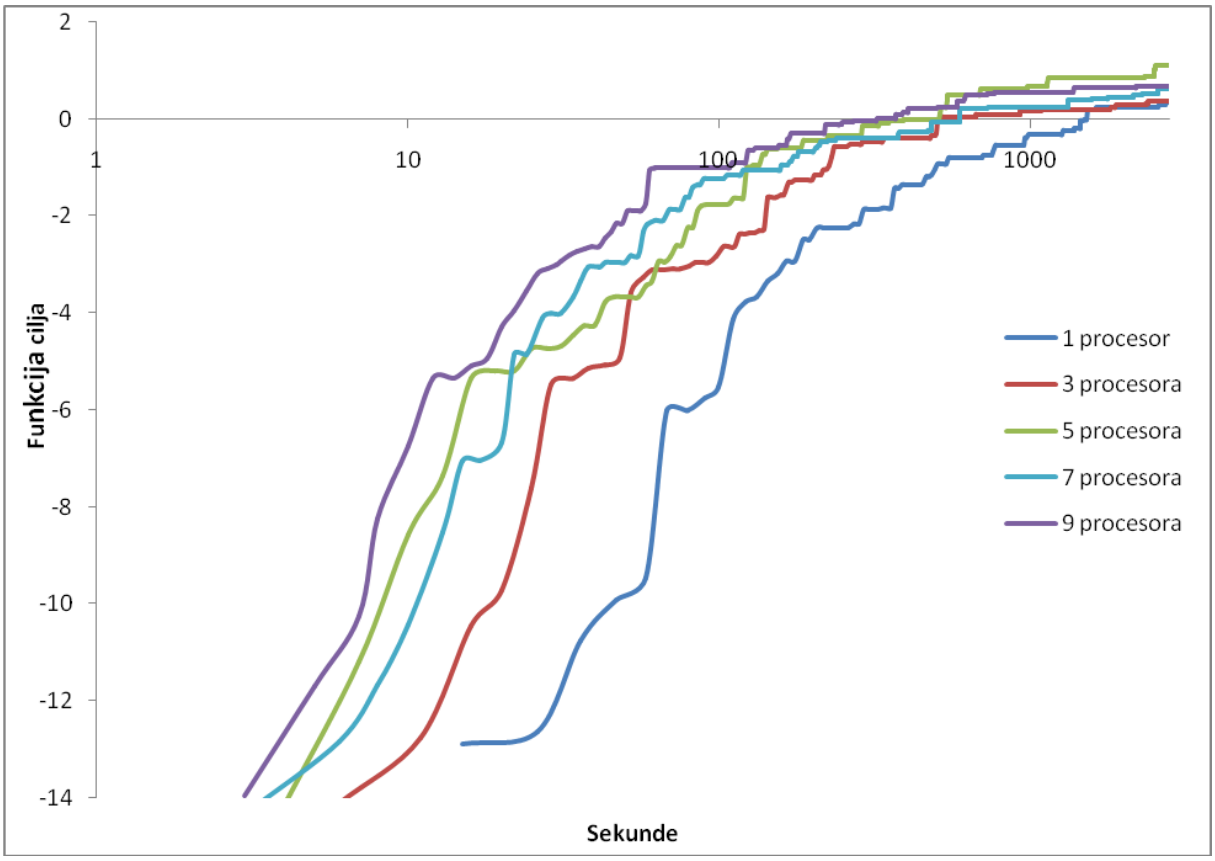
6.5.2 Sinkroni algoritam

Slika 6.10 prikazuje prosječnu dosegnutu vrijednost funkcije cilja sinkronog globalnog paralelnog genetskog algoritma u ovisnosti o vremenu za konfiguracije s različitim brojem procesora. Vidljivo je da za veći broj procesora funkcija cilja u početku znatno brže raste, no približavanjem maksimalnoj dosegnutoj vrijednosti ponašanje na svim konfiguracijama postaje sve sličnije. Sve većim poboljšavanjem vrijednosti funkcije cilja, daljnja poboljšanja sve manje ovise o brzini pretrage prostora rješenja, a više o slučajnim mutacijama i uspješnim križanjima. Zbog toga se može dogoditi da algoritam koji se izvršava na manjem broju procesora dođe do zadovoljavajućeg rješenja u kraćem vremenu i kroz manji broj iteracija.



Slika 6.10 Prikaz kretanja prosječne vrijednosti funkcije cilja kroz vrijeme za sinkroni algoritam

Slika 6.11 prikazuje kretanje rješenja s osi vremena prikazanoj u logaritamskoj skali. Takav prikaz omogućuje bolji uvid u ponašanje algoritma u ranom stadiju, dok funkcija cilja brzo raste.



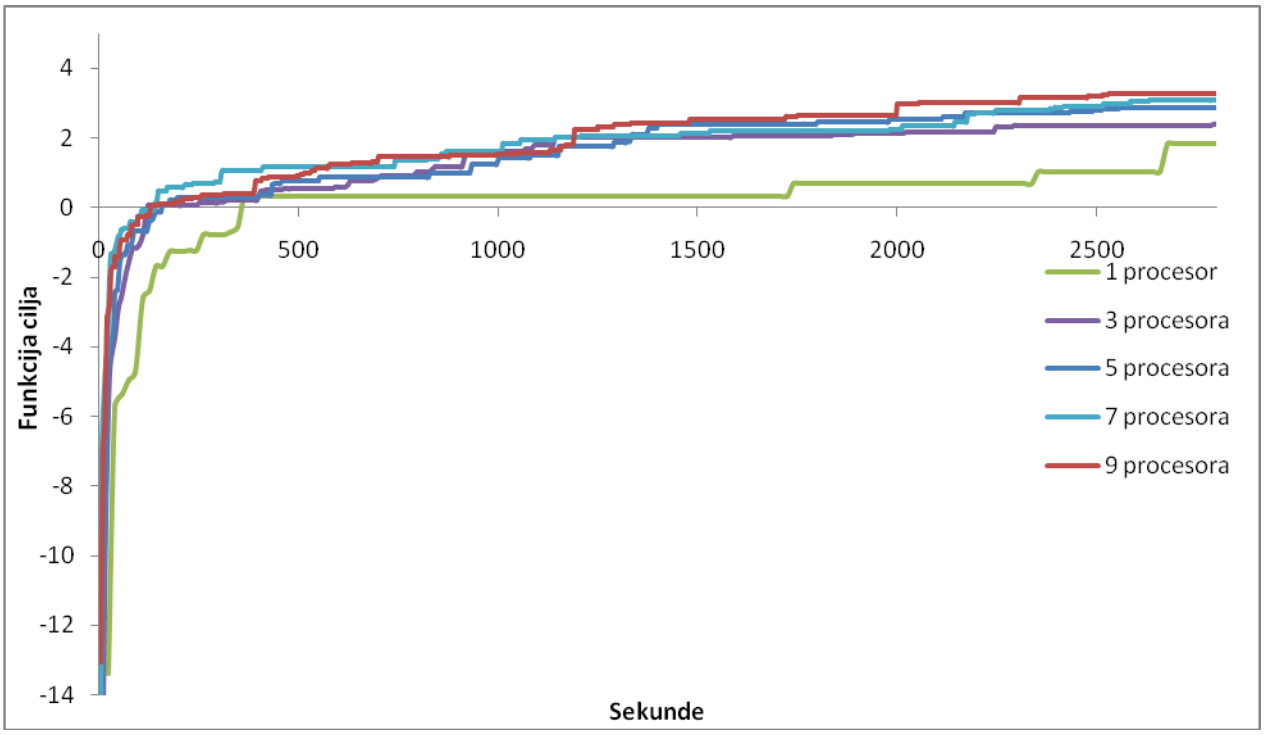
Slika 6.11 Prikaz kretanja prosječne vrijednosti funkcije cilja kroz vrijeme u logaritamskoj skali za sinkroni algoritam

6.5.3 Asinkroni algoritam

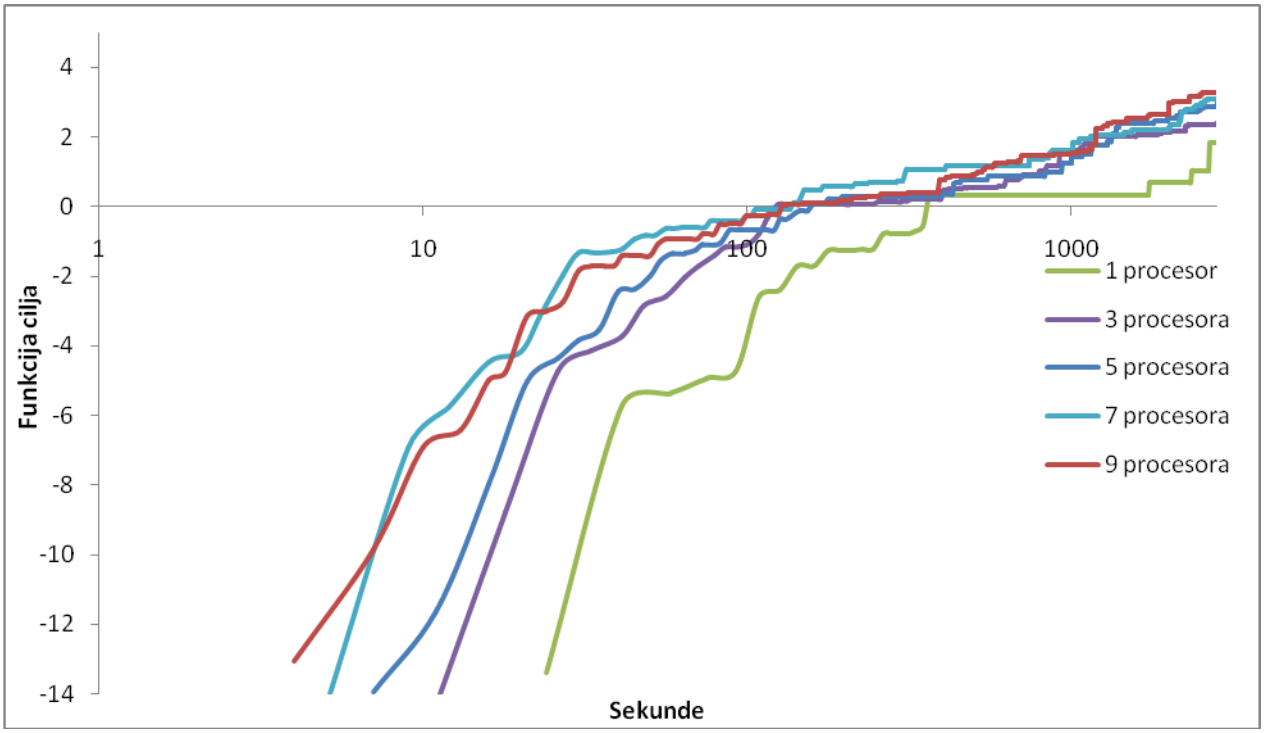
Slika 6.12 prikazuje prosječnu dosegnutu vrijednost funkcije cilja asinkronog globalnog paralelnog genetskog algoritma u ovisnosti o vremenu za konfiguracije s različitim brojem procesora. Kao i kod sinkronog algoritma, za veći broj procesora funkcija cilja u početku znatno brže raste. Kada se vrijednost funkcije cilja približi maksimalnoj dosegnutoj, razlika u ponašanju algoritma na različitim konfiguracijama se smanjuje.

Asinkroni algoritam je brže pronalazio dobra rješenja i u svim mjerenjima je pronašao znatno bolje konačno rješenje nego sinkroni algoritam. Ovo je vjerojatno posljedica eliminacijske selekcije koja se koristi u asinkronoj inačici i njene bolje prilagođenosti problemima sa velikim kromosomima i zahtjevnom evaluacijskom funkcijom od generacijske selekcije.

Slika 6.13 prikazuje kretanje rješenja s osi vremena prikazanoj u logaritamskoj skali, radi boljeg prikaza ponašanja algoritma u ranom stadiju, dok funkcija cilja brzo raste.



Slika 6.12 Prikaz kretanja prosječne vrijednosti funkcije cilja kroz vrijeme za asinkroni algoritam



Slika 6.13 Prikaz kretanja prosječne vrijednosti funkcije cilja kroz vrijeme u logaritamskoj skali za asinkroni algoritam

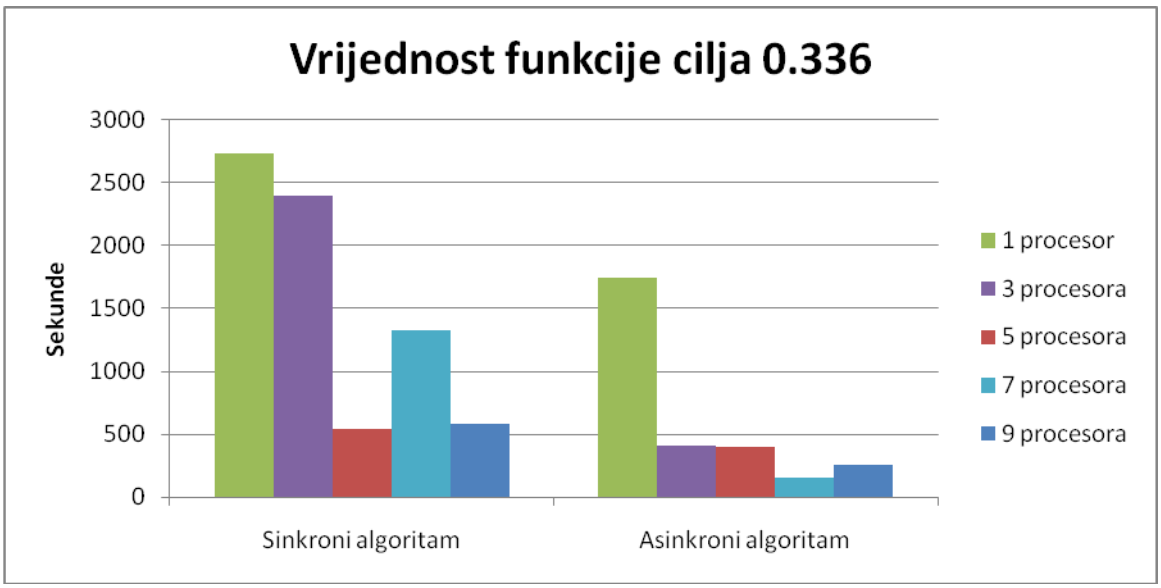
6.5.4 Brzina dostizanja zadane vrijednosti funkcije cilja

Analiza učinkovitosti paralelizacije obavljena je za dvije vrijednosti funkcije cilja: 0.336 i 1.83. Prva vrijednost je odabrana kao najbolja prosječna vrijednost funkcije cilja koju su za vrijeme izvođenja ostvarili i sinkroni i asinkroni algoritam na svim konfiguracijama. Druga vrijednost funkcije cilja za koju je provedena analiza je općenito najbolja vrijednost postignuta na svim konfiguracijama i nju je dosegnuo samo asinkroni algoritam.

Slika 6.14 prikazuje vrijeme dostizanja prosječne vrijednosti funkcije cilja u iznosu od 0.336 za različite konfiguracije i inačice genetskog algoritma. Ta vrijednost znači da računalni igrač Ajnca nastao kao rezultat genetskog algoritma ima prednost nad kućom od 0.336%. Bilo kakva prednost nad kućom se u igri Ajnc smatra vrlo uspješnim rezultatom.

Sinkrona i asinkrona verzija algoritma su na većem broju procesora postigle generalno bolje rezultate. Ipak, u nekim slučajevima je algoritam na manjem broju procesora brže došao do rješenja od algoritma koji se vrtio na većem broju

procesora. Iz toga je vidljivo da velika razlika u procesorskoj snazi nije uvijek dovoljna da poništi slučajnu narav traženja rješenja.



Slika 6.14 Brzina dostizanja zadane vrijednosti funkcije cilja

Tablica 6.5 prikazuje relativnu učinkovitost i relativno ubrzanje paralelizacije genetskog algoritma izmjerenih pri analizi vrijednosti funkcije cilja u iznosu od 0.336 za sinkroni algoritam, a Tablica 6.6 za asinkroni.

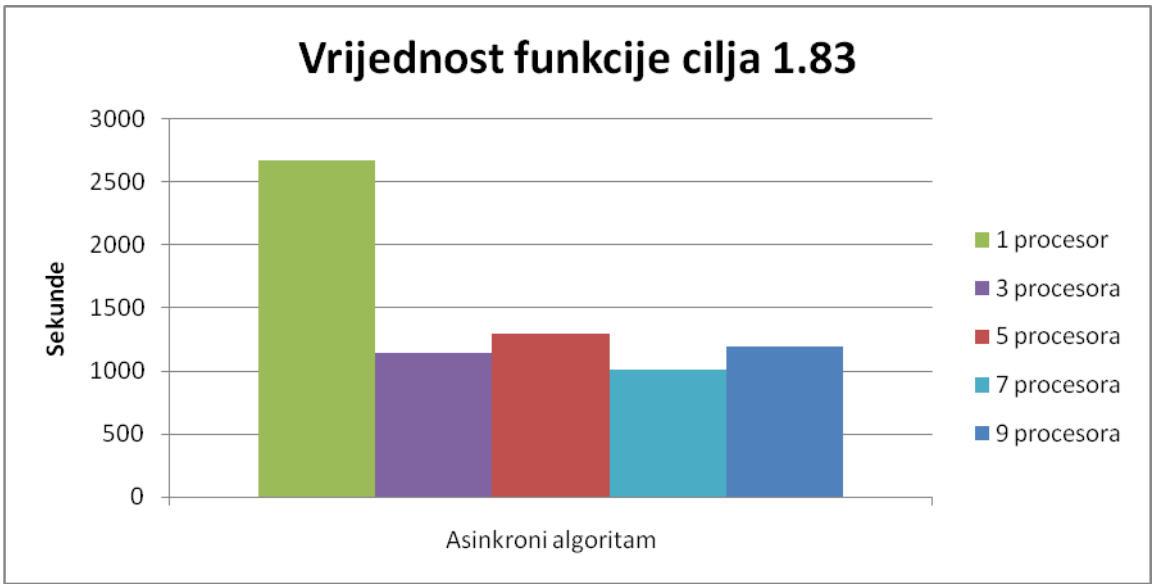
Tablica 6.5 Relativna učinkovitost i relativno ubrzanje paralelizacije sinkronog genetskog algoritma za vrijednost funkcije cilja 0.336

Broj procesora	**1**	**3**	**5**	**7**	**9**
Relativna učinkovitost	1	0.38	1.011	0.3	0.52
Relativno ubrzanje	1	1.14	5.06	2.07	4.69

Tablica 6.6 Relativna učinkovitost i relativno ubrzanje paralelizacije asinkronog genetskog algoritma za vrijednost funkcije cilja 0.336

Broj procesora	**1**	**3**	**5**	**7**	**9**
Relativna učinkovitost	1	1.44	0.87	1.66	0.75
Relativno ubrzanje	1	4.3	4.33	11.6	6.77

Za vrijeme izvršavanja algoritma u vremenu od 2800 sekundi, asinkroni algoritam je dosegao znatno bolju maksimalnu vrijednost funkcije cilja od sinkronog, u iznosu od 1.83. Brzinu dosega te vrijednost funkcije cilja prikazuje Slika 6.15, a Tablica 6.7 relativnu učinkovitost i ubrzanje algoritma.



Slika 6.15 Brzina dostizanja visoke vrijednosti funkcije cilja za asinkroni algoritam

Tablica 6.7 Relativna učinkovitost i relativno ubrzanje paralelizacije asinkronog genetskog algoritma za vrijednost funkcije cilja 1.83

Broj procesora	**1**	**3**	**5**	**7**	**9**
Relativna učinkovitost	1	0.78	0.41	0.38	0.23
Relativno ubrzanje	1	2.34	2.07	2.65	2.25

7. Zaključak

Paralelizacija genetskih algoritama pokazala se kao dobra tehnika za poboljšanje njihove učinkovitosti i skraćenja vremena potrebnog za dolazak do zadovoljavajućeg rješenja. Odabir prikladnog načina paralelizacije uvelike ovisi o karakteristikama problema koji se rješava: vremenskoj zahtjevnosti evaluacije jedinke i veličini kromosoma za prikaz rješenja.

Globalni paralelni genetski algoritam najprikladniji je za rješavanje problema sa vremenski zahtjevnom evaluacijom jedinki. Vrlo dobro povećanje broja obavljenih iteracija u vremenu postiže se jednostavnim dodavanjem procesorskih jezgri, bez potrebnih izmjena u programskom ostvarenju. Ipak, takvo poboljšanje nije jednako uspješno prenosivo na ubrzanje dostizanja zadane vrijednosti funkcije cilja. Tehnika ujednačenja opterećenja među radnicima omogućuje postizanje dobrih rezultata sinkronoj inačici algoritma i na heterogenom procesorskom okruženju.

Problemi male složenosti evaluacijske funkcije i velikog kromosoma za prikaz rješenja bolje su prilagođeni nekoj od tehnika paralelizacije s mogućnošću paralelnog izvršavanja ostalih genetskih operatora (selekcije, križanja i mutacije). Dobar smjer za daljnja istraživanja je proučavanje karakteristika paralelizacije korištenjem raspodijeljenih genetskih algoritama i masovno paralelnih genetskih algoritama.

8. Literatura

[1] Golub, M: Genetski algoritam – prvi dio, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf , 2004.

[2] Holland, John H *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[3] Wikipedia.org: On the Origin of Species, s Interneta, http://en.wikipedia.org/wiki/On_the_Origin_of_Species

[4] Obitko, M: Introduction Genetic Algorithms, s Interneta, http://www.obitko.com/tutorials/genetic-algorithms/

[5] Golub, M: Genetski algoritam – drugi dio, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf, 2004.

[6] Jakobović, D: ECF - Evolutionary Computation Framework, s Interneta, http://gp.zemris.fer.hr/ecf/

[7] Wikipedia.org: Machine learning, s Interneta, http://en.wikipedia.org/wiki/Machine_learning

[8] Tom M. Mitchell: Machine Learning p.2, 1997.

[9] Wikipedia.org: Genetic programming, s Interneta, http://en.wikipedia.org/wiki/Genetic_programming

[10] Ferreira, C: „Gene Expression Programming: A New Adaptive Algorithm For Solving Problems“, Complex Systems, vol. 13, issue 2, pp. 87-129, SAD, s Interneta, http://www.gene-expression-programming.com/webpapers/gep.pdf, 2001.

[11] Wikipedia.org: Blackjack, s Interneta, http://en.wikipedia.org/wiki/Blackjack

[12] Wikipedia.org: Monte Carlo method, s Interneta, http://en.wikipedia.org/wiki/Monte_Carlo_method

[13] MPICH2 : High-performance and Widely Portable MPI, s Interneta, http://www.mcs.anl.gov/research/projects/mpich2/

[14] Amazon Elastic Compute Cloud (Amazon EC2), s Interneta, http://aws.amazon.com/ec2/