Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

Marko Kompara

VPELJAVA VERJETNOSTI IZBIRE GENOV NA OSNOVI STAROSTI PRI GENETSKIH ALGORITMIH

Magistrsko delo

Maribor, september 2015

VPELJAVA VERJETNOSTI IZBIRE GENOV NA OSNOVI STAROSTI PRI GENETSKIH ALGORITMIH

Magistrsko delo

Študent:	Marko Kompara
Študijski program:	Študijski program 2. stopnje
	Informatika in tehnologije komuniciranja
Mentor:	red. prof. dr. Vili Podgorelec

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: E5012752
Datum in kraj: 26. 02. 2015, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 46/2012)
izdajam

SKLEP O MAGISTRSKEM DELU

1. **Marku Kompari**, študentu študijskega programa 2. stopnje INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA, se dovoljuje izdelati magistrsko delo.

2. Tema magistrskega dela je pretežno s področja Inštituta za informatiko.

MENTOR: red. prof. dr. Vili Podgorelec

3. Naslov magistrskega dela:
VPELJAVA VERJETNOSTI IZBIRE GENOV NA OSNOVI STAROSTI PRI GENETSKIH ALGORITMIH

4. Naslov magistrskega dela v angleškem jeziku:
THE INTRODUCTION OF AGE BASED GENE SELECTION PROBABILITY IN GENETIC ALGORITHMS

5. Magistrsko delo je potrebno izdelati skladno z »Navodili za izdelavo magistrskega dela«. Skladno s 6. členom *Pravilnika o postopku priprave in zagovora magistrskega dela na študijskih programih 2. stopnje UM* je bilo odobreno podaljšanje roka za oddajo magistrskega dela do 14. 05. 2016. Magistrsko delo študent-ka odda v treh izvodih (dva vezana izvoda in en v spiralo vezan izvod) v pristojni referat ter elektronski izvod v Digitalno knjižnico Univerze v Mariboru.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 15 dni.

Obvestiti:
1. kandidata
2. mentorja
3. odložiti v arhiv

Dekan:
red. prof. dr. Borut Žalik

UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO

W: feri.um.si | E: feri@um.si | T: 02 220 7000 | F: 02 220 7272 | TRR: 01100 6090106039 | ID: SI71674705

ZAHVALA

Zahvaljujem se mentorju prof. dr. Viliju Podgorelcu za vodenje, pomoč in napotke pri izdelavi magistrskega dela.

Posebej bi se rad zahvalil staršem, ki so mi omogočili in me podpirali pri študiju.

Prav tako bi se rad zahvalil Urošu in Tanji za potrpežljivost in pomoč.

Vpeljava verjetnosti izbire genov na osnovi starosti pri genetskih algoritmih

Ključne besede: genetski algoritem, Dawkins Weasel, funkcija Michalewicz, problem nahrbtnika, starostno odvisna izbira genov

UDK: 004.421.4(043.2)

Povzetek

Magistrska naloga preučuje posledice vpeljave spremenjenega načina križanja in mutacije v genetske algoritme. Naključna izbira genov v obeh operacijah je nadomeščena s postopkom izbire, pri kateri je verjetnost izbora posameznega gena odvisna od njegove starosti. Za ta namen je zgrajena aplikacija, v kateri so poleg tradicionalnega genetskega algoritma implementirani še dodatni operatorji mutacije in križanja. Izbira genov za sodelovanje v novonastalih načinih reprodukcije je na različne načine odvisna od starosti genov. Vsi načini delovanja so preizkušeni na treh različnih problemih. Rezultati starostno odvisnih načinov delovanja so z namenom ugotavljanja kredibilnosti takšnega delovanja primerjani z rezultati in delovanjem konvencionalnega genetskega algoritma.

The introduction of age based gene selection probability in genetic algorithms

Key words: genetic algorithm, Dawkins Weasel, Michalewicz function, knapsack problem, age dependant gene selection

UDK: 004.421.4(043.2)

Abstract

The master's thesis studies the effect of introducing a new type of crossover and mutation into the genetic algorithms. The random selection in both operations is replaced by a new procedure, where the probability for each gene to be selected is dependant on its age. For this porpose an application is built, that in addition to the traditional genetic algorithm also implements extra mutation and corssover operators. Gene selection in the newly created reproduction methods is in different ways dependant on the genes age. All the various modes of operation are tried on three different problems. The results of age-dependent modes of operation are compared with the results of conventional genetic algorithm in order to establish the new operations credibility.

KAZALO

KAZALO SLIK

KAZALO TABEL

UPORABLJENE KRATICE

CSV	Comma-separated values
DNA	Deoxyribonucleic acid
DNK	Deoksiribonukleinska kislina
EA	Evolucijski algoritem
FWHM	Full width at half maximum
GA	Genetski algoritem
GNF	Generacija najboljšega fitnesa
NF	Najboljši fitnes
PF	Povprečen fitnes skozi izvajanje
PNF	Povprečen najboljši fitnes skozi izvajanje
SPSS	org. Statistical Package for the Social Sciences

1 UVOD

Digitalni računalnik je najverjetneje najpomembnejši izum v zgodovini znanosti in tehnologije. Z njegovim nadaljnjim razvojem se hitro povečuje naša sposobnost predvidevanja in kontroliranja narave, po kateri so se pri svojem delu zgledovali tudi pionirji računalništva. Biološki navdih je zatem nekoliko zamrl, dokler ni postal ponovno zelo zanimiv za raziskovalce v zgodnjih 1980-ih letih. Eden izmed rezultatov tega zanimanja so tudi genetski algoritmi.

Genetski algoritem je programska metoda za reševanje optimizacijskih problemov, ki v svojem delovanju oponaša proces naravnega izbora. Genetski algoritmi so tudi glavna tema te naloge. Specifično se bomo ukvarjali z izborom genov v operacijah mutacije in križanja. Standardno delovanje genetskih algoritmov opravi ta izbor na povsem naključen način. V tej nalogi bomo v tri različne genetske algoritme uvedli izbiro genov, ki bo delovala na podlagi starosti genov in raziskali posledice, ki jih ima takšna sprememba na rezultate oziroma delovanje algoritmov. To bomo storili na šest različnih načinov, kjer bo ena starostna skupina vedno imela prednost pred drugimi. Tako bomo imeli tudi možnost opazovanja vpliva različnih starostnih skupin. Cilj te naloge je odgovoriti na vprašanje, ali starostno odvisen izbor genov v operacijah mutacije in križanja (pozitivno ali negativno) vpliva na učinkovitost ter uspešnost delovanja genetskega algoritma. Pri tem učinkovitost kvalificiramo kot kvaliteto rešitev genetskih algoritmov, medtem ko je učinkovitost predstavljena kot hitrost, s katero algoritmi pridobijo najboljšo rešitev.

Uporaba starosti v genetskih algoritmih ni nova ideja. Uporablja se na primer v selekciji, kjer je lahko izbor osebkov za nadaljnje razmnoževanje odvisen od starosti kromosomov oziroma starosti samega osebka. Kar se zdi kot nov koncept je uporaba starosti posameznih genov, katere uporabe nismo zasledili v nobeni drugi raziskavi.

Vsebina naloge se začne s poglavjem, namenjenem opredelitvi raziskovalne metode, v skladu s katero je bila izvedena raziskava in na podlagi katere je osnovan ta dokument. Naslednje poglavje je precej obsežno in zavzema teorijo področja genetskih algoritmov. V tem poglavju se seznanimo z izvorom, zgodovino, vsemi sestavnimi deli in delovanjem genetskih algoritmov. Naslednji dve poglavji sta namenjeni kratki predstavitvi treh problemov, za katere smo v nadaljevanju implementirali genetske algoritme, in opisu

različnih načinov vpeljave starostne odvisnosti pri izbiri genov, ki smo jih razvili ter v nadaljevanju vnesli v algoritme poleg konvencionalnega delovanja. Zatem se posvetimo aplikaciji, ki smo jo razvili z namenom zbiranja podatkov, potrebnih za raziskavo. Aplikacija vsebuje tri omenjene genetske algoritme, katerih implementacija in delovanje sta podrobno obravnavana. Tu se lahko vidi tudi implementacija različnih načinov starostne odvisnosti pri izbiri genov. Temu sledi kratek opis samega izvajanja aplikacije oziroma pridobivanja podatkov za raziskavo. Analiza pridobljenih podatkov je ponovno ločeno obravnavana za vsak reševani problem. Rezultati vsakega so pregledani in uporabljeni v testih, na podlagi katerih lahko sodimo o učinkih, ki jih je spremenjeno delovanje imelo na genetske algoritme.

Nazadnje bi poudarili še hipoteze, ki jih bomo skozi to nalogo raziskovali. Ker pričakujemo, da bodo posamezni genetski algoritmi drugače reagirali na različne tipe vpeljave starostne odvisnosti pri izbiri genov, bomo hipoteze ločeno preverjali za vse pare genetskih algoritmov in tipov izvajanja.

H_{0A}: Starostno odvisen izbor genov nima statistično pomembnega vpliva na uspešnost genetskega algoritma.
H_{1A}: Starostno odvisen izbor genov pomembno spremeni uspešnost genetskega algoritma.

H_{0B}: Starostno odvisen izbor genov nima statistično pomembnega vpliva na učinkovitost genetskega algoritma.
H_{1B}: Starostno odvisen izbor genov pomembno spremeni uspešnost učinkovitost algoritma.

2 RAZISKOVALNA METODA

To poglavje je namenjeno krajši predstavitvi eksperimentalne metode, ki je uporabljena v tej nalogi. V poglavju se ne bomo spuščali v podrobnosti teorije empiričnega raziskovanja z eksperimentalno metodo, njegov namen je predvsem predstaviti pomembnejše aspekte takšne raziskave, tako da bo sledenje izvedeni raziskavi lažje, ter da bo namen korakov, ki smo jih med raziskavo izvedli, razumljiv. Dodatno bo to poglavje omogočilo tudi povprečnemu bralcu soditi o veljavnosti raziskave.

2.1 Raziskava

Čeprav natančna definicija raziskave (angl. *research*) ni enotno določena, saj obstajajo različne interpretacije procesa pomembnih avtorjev na tem področju, lahko rečemo, da je raziskava proces, ki pripelje do zanesljive rešitve za dotičen problem skozi načrtovano in sistematično zbiranje, analizo ter interpretacijo podatkov. Njen namen je pridobiti odgovor na določeno zastavljeno vprašanje, kjer ta odgovor še ni del človeškega znanja. Raziskovanje je zelo pomemben proces za napredovanje znanja, spodbujanje napredka in omogočanje učinkovitejše povezave med človekom in njegovim okoljem, tako da lažje dosega svoj namen ter rešuje konflikte. [1]

Pri zajemanju podatkov oziroma merjenju rezultatov raziskave moramo biti pozorni na veljavnost in zanesljivost meritev. Zanesljivost (angl. *reliability*) je mera konsistentnosti meritev. To pomeni, da večkratno merjenje istih spremenljivk ob enakih pogojih vrne bolj ali manj iste rezultate ob vsaki ponovitvi. Za ocenjevanje zanesljivosti obstaja precejšnje število pristopov. Tu bomo omenili samo testiranje in ponovno testiranje zanesljivosti (angl. *test-retest reliability*), zato ker bomo to metodo tudi sami uporabili v raziskavi. Pristop je relativno enostaven in zahteva, da iste meritve pod enakimi pogoji izvedemo dvakrat ob različnih časih. Če razlike med meritvami niso velike, potem so opravljene meritve zanesljive. Zanesljivost (angl. *validity*) se nanaša na sposobnost meritve, da ustrezno predstavi merjen konstrukt. Meritev je lahko zanesljiva in istočasno neveljavna (npr. merjenje napačne spremenljivke). Obratno je lahko meritev veljavna, vendar nezanesljiva, če so meritve nekonsistentne. [1-3]

Veljavnost se deli na številne (pod) vrste, kjer sta eni najosnovnejših in najpogosteje omenjenih notranja in zunanja veljavnost. Obe lastnosti sta zaželeni, vendar je načrtovanje raziskave, ki zagotavlja obe, tipično precej zahtevno. K zahtevnosti prispevajo tudi raziskovalne metode, ki skoraj praviloma zagotavljajo visok nivo veljavnosti za en tip in nizek nivo za drugi. Posledično nekateri raziskovalci trdijo, da sta ti dve lastnosti obratno sorazmerni, vendar to ne drži vedno, saj lahko z različnimi pristopi izboljšamo posamezen tip veljavnosti. [1, 2, 4]

Notranja veljavnost (angl. *internal validity*) izhaja iz nadzora nad spremenljivkami raziskave in govori o tem, ali je sprememba odvisne spremenljivke resnično posledica neodvisne spremenljivke in ne vplivov zunanjih faktorjev (več o tipih spremenljivk v naslednjem podpoglavju). Sposobnost zagotavljanja notranje veljavnosti je v veliki meri odvisna od raziskovalne metode. Na primer eksperimenti imajo praviloma dobro notranjo veljavnost, medtem ko so ankete pri zagotavljanju te veliko slabše. [1, 2]

Zunanja veljavnost (angl. *external validity*) je pogoj za generalizacijo pridobljenih rezultatov na celotno populacijo. Iz širšega nabora, kot zajamemo raziskovalni vzorec, večja je možnost generalizacije rezultatov in posledično večja zunanja veljavnost. Podobno kot pri notranji veljavnosti je tudi tukaj možnost zagotavljanja zunanje veljavnosti odvisna od raziskovalne metode. V tem primeru so ankete veliko učinkovitejše, saj lahko zajemajo veliko večji nabor iz širše populacije, medtem ko je eksperiment zaradi težje preslikave rezultatov v realno okolje, kjer ni nadzora nad spremenljivkami in posegi vanje, manj uspešen pri zagotavljanju zunanje veljavnosti. [1, 2]

Osnovna sestava raziskave se nekoliko razlikuje, glede na okoliščine oziroma uporabnike. Zato bomo tu predstavili eno bolj konservativnih konstrukcij, kjer je raziskovanje sestavljeno iz šestih korakov [1]:

1. izbira problema,
2. oblikovanje hipotez,
3. načrtovanje raziskave,
4. zbiranje podatkov,
5. analiza podatkov in
6. oblikovanje zaključkov.

Ti koraki bodo izvedeni tudi tekom te naloge. Izbira problema je prvi in najpomembnejši korak v raziskavi. Problem raziskave smo že predstavili v uvodnem poglavju. Problem

raziskave lahko rešimo učinkovito samo, če raziskovalci dobro razumejo okoliščine in faktorje, ki vplivajo na problem. Zato je zelo pomembno, da raziskovalci odlično poznajo predhodno teorijo in raziskave na raziskovalnem področju. Posledično je ena zgodnejših faz raziskovalnih projektov (čeprav ni vključena v raziskovalne korake) tipično seznanitev z raziskovalnim področjem, oziroma pregled literature (angl. *literature review*). V nalogi se predstavitev raziskovalnega področja nahaja v poglavju 3 (Genetski algoritmi). Prav tako je lahko dodaten korak v raziskavi izbira raziskovalnih vprašanj [2]. Raziskovalno vprašanje je podobno hipotezi, vendar ni tako natančno zastavljeno in je vedno v vprašalni obliki in ne v trdilni. Hipoteze so središče vsakega raziskovalnega procesa, saj so vse aktivnosti v tem procesu usmerjene v potrditev ali zavrnitev teh predpostavljenih trditev. Hipoteza je predlagana domnevna trditev, ki jo želimo tekom raziskave dokazati. Tako kot raziskovalni problem smo raziskovalno vprašanje in hipoteze za raziskavo definirali že v uvodnem poglavju. Načrtovanje raziskave v osnovi vključuje načrtovanje naslednjih aspektov: raziskovalna metoda, vzorčenje, izbira ali izdelava raziskovalnih orodij in izbira statističnih pristopov. Sama raziskovalna metoda je tema tega poglavja, vzorčenje in raziskovalno orodje sta opisana v 7. (Priprava in izvedba eksperimenta) in 6. (Izdelava aplikacije in implementacija genetskih algoritmov) poglavju, medtem ko je izbira statističnih tehnik opisana v poglavju 8.1.1 (Mann-Whitney U test). Zbiranje podatkov lahko vključuje širok nabor metod pridobivanja podatkov. V našem primeru so ti podatki rezultat delovanja genetskega algoritma. Postopek zbiranja podatkov in njihova struktura je opisana v poglavju 7 (Priprava in izvedba eksperimenta). Analiza zbranih podatkov je predstavljena v 8. poglavju (Analiza rezultatov), končni zaključki, ki jih lahko oblikujemo na podlagi analize, pa so predstavljeni v poglavju 9 (Sklep). [1]

2.2 Eksperimentalna metoda

Eksperimentalna metoda (angl. *experimental method*) je znanstvena metoda, pogosto opisana kot metoda testiranja hipotez. Podobno kot velja za raziskovanje ima tudi eksperiment številne, sicer podobne, vendar še vedno različne definicije. Kot združek vseh bi lahko rekli, da je eksperiment proženje in opazovanje dogodka pod poznanimi in nadzorovanimi pogoji, ob čim manjših zunanjih vplivih. Eksperimentiranje je najbolj primerno za razlagalno oziroma pojasnjevalno raziskavo (angl. *explanatory research*), čeprav bo v tem primeru metoda uporabljena za poizvedovalno raziskavo (angl. *exploratory research*). Namen eksperimenta je pridobiti verificirano funkcionalno razmerje med pojavi pod nadzorovanimi pogoji. [1, 2]

Osnovna predpostavka za principom eksperimenta se pogosto imenuje zakon ene spremenljivke (angl. *law of single variable*), ki obravnava dve v vsakem pogledu enaki situaciji. Zakon trdi, da če v eni izmed situacij spremenimo eno spremenljivko ter posledično med obema situacijama nastanejo razlike, so vse nastale razlike med obema situacijama posledica spremembe v tisti eni spremenljivki. [1]

Glavni cilj znanstvenih študij je analiza funkcijskih odvisnosti (angl. *functional relationship*) med spremenljivkami, kjer se funkcijska odvisnost nanaša na vzročno-posledično (angl. *cause and effect*) razmerje med spremenljivkami. Takšno razmerje med dvema spremenljivkama pomeni, da ena spremenljivka (vzrok) povzroči neke spremembe na drugi spremenljivki (posledica). Vzrok ima lahko tudi več posledic. Za to, da med dvema spremenljivkama lahko vzpostavimo vzročno-posledično razmerje, morajo biti izpolnjeni trije kriteriji. Prvi pogoj zahteva, da se vzrok vedno zgodi pred posledico. Drugi pogoj zahteva, da se ob vzročnem dogodku vedno dogodi tudi posledica. Zadnji pogoj, ki mora biti izpolnjen, da lahko govorimo o vzročno-posledičnem razmerju, je neobstoj drugega faktorja, ki lahko pojasni odvisnost med vzrokom in posledico. Zagotavljanje tega kriterija je nekoliko težje od prejšnjih dveh in tu vstopi na pomoč zakon ene spremenljivke. Z njegovo pomočjo namreč zagotovimo, da so vse posledice, ki se dogodijo v našem nadzorovanem sistemu, posledica ene spremenljivke (vzroka). [1, 5]

Če je fokus eksperimentalne metode opazovanje vzročno-posledičnih razmerij, se sam proces eksperimentiranja vrti okrog spremenljivk, med katerimi tudi opazujemo ta razmerja. Spremenljivka je vsaka lastnost dogodka, funkcije ali procesa, ki vpliva na neki drug dogodek, funkcijo ali proces, ki ga opazujemo. [1]

V eksperimentu se spremenljivke ločijo v štiri različne skupine [1]:

- Neodvisna spremenljivka (angl. *experimental variable* ali *independent variable*): je spremenljivka, katere učinke preučujemo v eksperimentu. Neodvisna spremenljivka je vzrok v vzročno-posledičnem razmerju med spremenljivkami.
- Nadzorovana spremenljivka (angl. *controlled variable*): je spremenljivka, ki se med eksperimentiranjem ne spreminja, je vedno konstantna.
- Odvisna spremenljivka (angl. *criterion variable* ali *dependent variable*): je osnova, na podlagi katere je opredeljena učinkovitost neodvisne spremenljivke. Odvisna spremenljivka je spremenljivka, ki jo merimo, da razberemo razlike med dvema situacijama, ki sta, z izjemo neodvisne spremenljivke, identični. V vzročno-posledičnem razmerju odvisna spremenljivka predstavlja posledico.

- Posredovalna spremenljivka (angl. *intervening variable*): je abstraktna spremenljivka, ki moti oziroma posega v efekt, ki ga ima neodvisna spremenljivka na odvisno spremenljivko. Dva primera takšne spremenljivke sta motivacija in odnos sodelujočih v eksperimentu.

Sedaj, ko ločimo med odvisnimi in neodvisnimi spremenljivkami, lahko s praktičnega vidika gledano rečemo, da je eksperiment spreminjanje neodvisne spremenljivke, tako da lahko preučujemo učinek, ki ga ima takšno spreminjanje na odvisno spremenljivko. [1]

Dva osnovna koncepta eksperimentalne metode sta poseg (angl. *treatment*) in nadzorna ali kontrolna skupina (angl. *control group*). V eksperimentih se na delu primerkov aplicira eno ali več eksperimentalnih stimulacij, katere imenujemo posegi. Skupino, v kateri so posegi izvedeni, bomo imenovali skupina s posegom (angl. *Treatment group*), medtem ko se skupina osebkov, ki niso deležni posega, imenuje nadzorna skupina. Skupina s posegom se upošteva kot uspešnejša od kontrolne, če njeni osebki pokažejo boljše rezultate v odvisni spremenljivki. V enem eksperimentu imamo lahko več različnih posegov in posledično več različnih skupin s posegom. [2]

Eksperimentalne raziskave se v osnovi delijo v dve skupini: pravi eksperimentalni modeli in kvazieksperimentalni (angl. *quasi-experimental*) modeli. Obe skupini vsebujeta manipulacijo posegov, vendar samo pravi eksperimenti zahtevajo naključno porazdelitev v skupine. Naključna selekcija in naključno dodeljevanje zagotavljata, da ima vsak osebek v populaciji enako možnost vključitve v raziskavo in podobnost med skupinami osebkov v eksperimentu. Naključna selekcija je vključevanje naključnih osebkov iz raziskovane populacije v vzorec eksperimenta. Naključno dodeljevanje je naključno razporejanje osebkov iz vzorca v skupine s posegi in nadzorne skupine. Naključna selekcija vpliva na zunanjo veljavnost, po drugi strani pa naključno dodeljevanje vpliva predvsem na notranjo veljavnost. Pravi eksperimenti lahko vključujejo tako naključno selekcijo kot dodeljevanje, medtem ko kvazieksperimenti ne vključujejo nobene od teh opcij. [2]

Najbolj pogosta oblika pravega eksperimenta je eksperiment z dvema skupinama (skupina s posegom in kontrolna skupina), ki je odličen za testiranje posledic posega v eno neodvisno spremenljivko. Dva osnovna primera takšne oblike sta eksperiment s predtestom, potestom in kontrolno skupino (angl. *pretest-posttest control group design*) ter eksperiment samo s potestom in kontrolno skupino (angl. *posttest-only control group design*). Vsaka o teh oblik lahko vključuje še kovarianco (angl. *covariance*). Ko na odvisno

spremenljivko vplivajo zunanji vplivi, ki v sklopu same raziskave niso pomembni, vendar morajo biti vseeno nadzorovani, se za odstranitev nezaželenih vplivov uporabi kovariantna oblika. Naprednejši eksperimenti so lahko osnovani tudi na faktorski obliki (angl*. factorial designs*), kjer je omogočeno spremljanje več neodvisnih spremenljivk (lahko vnesemo več različnih posegov) ali hibridni obliki eksperimentov (angl*. hybrid experimental designs*), kjer se združujejo različne uveljavljene oblike eksperimentov. Za predstavitev različnih oblik raziskave se najpogosteje uporabi standardizirana notacija (Slika 2.1), v kateri R predstavlja naključno razdelitev osebkov v skupine, znak O predstavlja merjenje (testiranje) odvisne spremenljivke in X pomeni, da nad skupino opravimo poseg. [2]

Eksperiment s potestom in kontrolno skupino (Slika 2.1) je ena najosnovnejših oblik eksperimenta in istočasno tudi oblika, ki jo bomo v nadaljnji raziskavi uporabili. Osebke najprej naključno razdelimo med dve skupini. Na skupini, ki ni kontrolna, izvedemo poseg. Nazadnje obe skupini testiramo oziroma izmerimo njihove rezultate. Učinek posega je predstavljen kot razlika rezultatov testa med kontrolno in skupino s posegom. Prednost eksperimenta brez predtesta je preprostost in boljša notranja veljavnost. [2]

R	X	O_1	(Skupina s posegom)
R		O_2	(Kontrolna skupina)

Slika 2.1: Standarden zapis eksperimenta s potestom.

Ostalih oblik eksperimentov, ki jih v nalogi ne bomo izvedli, ne bomo podrobno opisovali. Eksperiment s predtestom, potestom in kontrolno skupino je nadgradnja že opisane oblike, tako, da se za naključno razdelitev osebkov v skupine doda še eno testiranje (za obe skupini). Oblike kvazieksperimentov so zelo podobne oblikam pravih eksperimentov, vendar ne vključujejo naključne porazdelitve v skupine. Osebki se v skupine določijo glede na neki parameter (npr. zaposleni enega podjetja so kontrolna skupina, medtem ko so zaposleni drugega podjetja skupina, v katero bo vnesen poseg). To omogoča vključitev pristranskosti v razdelitev osebkov. Posledično so kvazieksperimenti slabši pri zagotavljanju notranje veljavnosti. Številne oblike pravih eksperimentov so zmožne transformacije v kvazieksperimentalni model s preprosto izpustitvijo naključne razporeditve. [2]

Koraki eksperimentalne metode so praktično enaki korakom v splošni znanstveni metodi [1]:

1. Izbira in omejevanje problema: poimenovanje koraka že samo po sebi razloži njegovo vsebino. Problemi, ki so raziskovani z eksperimenti, so tipično lahko pretvorjeni v hipoteze (praviloma bi tudi morali biti), tako da jih lahko zavrnemo ali sprejmemo na podlagi podatkov pridobljenih tekom eksperimenta.
2. Pregled literature: namenjen spoznavanju področja in pregledu podobnih že obstoječih raziskav.
3. Priprava načrta za eksperiment: zajema od osnovnih podatkov, kot sta na primer lokacija in trajanje eksperimenta, do bolj zahtevnih in pomembnih delov načrtovanja, kot so kontrola, naključnost in ponovljivost eksperimenta. Glede na kompleksnost eksperimenta je priporočljiva izvedba poskusne študije, za zagotovitev ustreznosti načrta.
4. Določanje populacije: potrebno zato, da vemo, na kakšno populacijo lahko apliciramo zaključke raziskave.
5. Izvedba eksperimenta: korak, kjer fizično izvedemo eksperiment. Tu je pomembno, da natančno sledimo pripravljenemu načrtu, saj se nanj nanašajo dejavniki kontrole, naključja in ponovljivosti. Čas izvajanja eksperimenta mora biti zadosten, da imajo opazovane spremenljivke možnost spreminjanja, ki ga lahko merimo.
6. Merjenje rezultatov: mora biti izvedeno tako, da so kriteriji merjenja pošteni do vseh vpletenih, in ob različnih izvajanjih.
7. Analiza in razlaga rezultatov: raziskovalec mora biti pozoren na možnost, da so rezultati raziskave posledica nenadzorovanih vplivov zunanjih dejavnikov. Raziskovalec mora z določeno verjetnostjo tudi zavrniti možnost, da so rezultati zgolj posledica naključja. Uporaba statistike je v eksperimentu, v primerjavi z drugimi raziskovalnimi metodami, zelo pomembna za pravilno interpretacijo podatkov. Pri tem pa se je potrebno zavedati, da statistika ne more odpraviti pomanjkljivosti v načrtovanju ali nepravilnosti v podatkih.
8. Izpeljava zaključkov: omejena mora biti na populacijo, ki je bila vključena v eksperiment, tako da rezultatov ne posplošimo prekomerno (angl. *generalization*). Zaključki so omejeni tudi na pogoje, pri kateri so bili pridobljeni. Poleg teh omejitev končnih zaključkov se mora raziskovalec tudi zavedati, da so zaključki izpeljani na podlagi statistične verjetnosti.
9. Poročanje rezultatov: je zadnja točka v procesu izpeljave eksperimenta. Poročanje mora biti dovolj natančno in mora vsebovati vse potrebne podatke, da si lahko bralci sami ustvarijo mnenje o ustreznosti in uspešnosti eksperimenta.

3 GENETSKI ALGORITMI

Genetski algoritem (angl. *genetic algorithm*), pogosto okrajšan na GA, je programska optimizacijska in iskalna tehnika. GA je hevrističen, kar pomeni, da išče dovolj dobro rešitev na določen problem in ni nujno osredotočen na najboljšo možno rešitev. Ideja genetskih algoritmov je zasnovana na osnovi genetike in naravnega izbora, včasih imenovanega tudi naravna selekcija (angl. *natural selection*). [6]

Genetski algoritmi so skupaj z evolucijskimi strategijami (angl. *evolution strategy*) in evolucijskim programiranjem (angl. *evolutionary programming*) tri glavne oblike evolucijskih algoritmov (okrajšano EA; angl. *evolutionary algorithm*). Začetek evolucijskega programiranja sega v sredino 1950-ih let. Eden prvih opisov uporabe evolucijskega principa delovanja v računalniškem programu je objavil R. M. Friedberg leta 1958 [7] in nato 1959 [8] v nadaljevalnem članku. Nekateri pionirji na tem področju izpostavljajo članek A. S. Fraserja (1957) [9] kot pomemben vpliv na njihovo začetno delo. Področje se je zatem razvijalo in raslo, dokler niso v drugi polovici 60ih let natančneje opredelili treh prej omenjenih oblik evolucijskih algoritmov. Za genetske algoritme je te začetke postavil J. H. Holland leta 1967 [10], ideje delovanja genetskih algoritmov pa je objavil že na začetku istega desetletja. V njegovi knjigi z naslovom "*Adaptation in natural and artificial systems*" [11], ki je postala eno temeljnih del področja, je leta 1975 kot prvi predstavil koncept adaptivnega sistema, ki uporablja operatorje selekcije, križanja in mutacije [12]. [13]

3.1 Izvor genetskih algoritmov

Kot je že bilo rečeno, ideja za GA izhaja iz genetike in naravnega izbora, od koder se je prevzel tudi velik del terminologije. Vsi živi organizmi so sestavljeni iz celic, ki vsebujejo zbirko enega ali več kromosomov (angl. *Chromosome*). Kromosomi vsebujejo deoksiribonukleinsko kislino – DNK (angl. *Deoxyribonucleic acid – DNA*), ki nosi osnovo oziroma načrt za dani organizem. Vsak kromosom se lahko konceptualno naprej deli v gene. V naravi vsak gen vsebuje specifičen protein, ki je del DNK. Na gene lahko gledamo kot nosilce posameznih značilnosti osebka (npr. barva oči). Vsaka različna možnost, ki jo gen lahko zasede se imenujejo alel (angl. *Allele*), lokus (angl. *Locus*) pa je točno določeno mesto, na katerem se nahaja posamezen gen v kromosomu. [14]

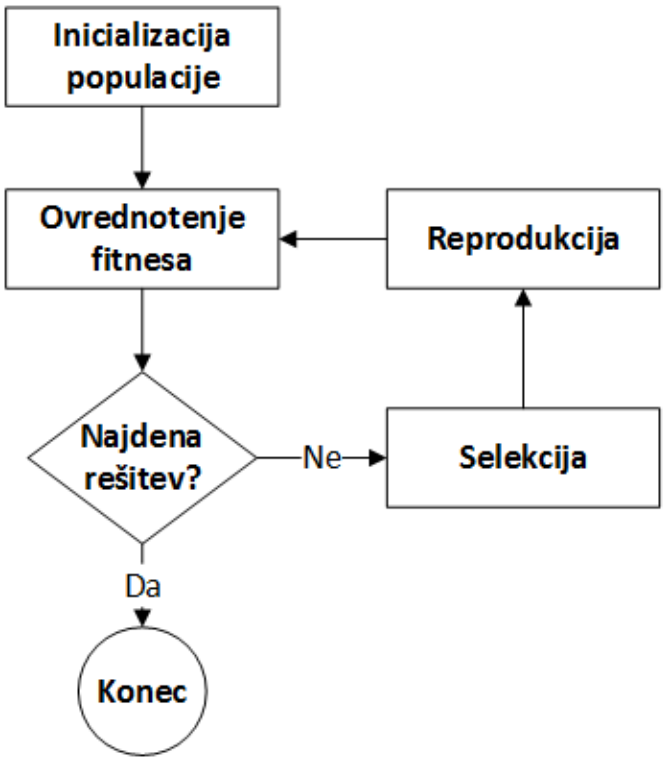
Številni organizmi imajo v vsaki celici več kot en kromosom (npr. človek jih ima 23 parov). Celoten genetski material (skupek vseh kromosomov) imenujemo genom (angl. *genome*), medtem ko je genotip (angl. *genotype*) specifičen nabor genov v genomu. Dva osebka z identičnima genomoma imata isti genotip. Fenotip (angl. *phenotype*) je skupek osebkovih fizičnih in mentalnih karakteristik (npr. barva oči, višina, inteligenca itd.), ki nastane skozi razvoj zaradi njegovega genotipa in vpliva okolja. [14]

Organizme, katerih kromosomi so urejeni v pare, imenujemo diploidi (angl. *diploid*), tiste, pri katerih niso, pa haploidi (angl. *haploid*). V naravi je večina vrst, ki se spolno razmnožujejo, vključno s človekom, diploidov. Med spolno reprodukcijo se v vsakem staršu geni v paru kromosomov izmenjajo v procesu, imenovanem križanje (angl. *crossover*), rezultat katerega je gameta (angl. *gamete*) – samostojni kromosom. Zatem se gameti obeh staršev združita v par diploidnih kromosomov. Pri haploidni spolni reprodukciji se geni izmenjajo neposredno med kromosomi staršev. Potomci so lahko predmet mutacije (angl. *mutation*), kjer se posamezni nukleotidi (angl. *nucleotide*) spremenijo med staršem in otrokom, najpogosteje kot posledica napake pri kopiranju. Cenilna funkcija, funkcija uspešnosti ali s tujko kar fitnes funkcija (angl. *fitness function*) se za organizme tipično ocenjuje kot verjetnost preživetja do reprodukcije ali kot število potomcev, ki jih ta organizem ima. [14]

V GA je kromosom kandidat za rešitev problema. Preprosta oblika kromosoma je bitni niz, kjer je gen lahko predstavljen kot posamezen bit ali kot blok bitov. Kromosom je lahko predstavljen tudi kot na primer tabela kompleksnejših podatkov (gen je en element v tabeli) ali kot izvedljivo drevo kode (gen je vozlišče v drevesu) [12], vendar se bomo v tem poglavju omejili na osnovno bitno predstavitev zgolj za enostavnejše razumevanje preslikave med naravnim procesom in GA. Kjer je gen izražen kot posamezen bit, je alel ali znak 0 ali 1, v primeru bloka bitov pa so to vse možne kombinacije obeh znakov. Križanje v GA tipično zavzema izmenjavo genov med haploidnimi (kromosomi niso parni) starši. Mutacija v primeru bitnega zapisa predstavlja obrnitev bita na naključnem lokusu. Velika večina GA uporablja haploidne osebke, še posebej, ko gre za osebke z enim samim kromosomom. Genotip vsakega osebka je zaporedje vseh bitov v njegovem genomu. V GA fenotip tipično ne obstaja. [14]

3.2 Koncept in osnove genetskih algoritmov

V prejšnjem poglavju smo govorili o tem, kako so GA povezani z biološkim procesom in kako se je od tam terminologija prenesla na področje evolucijskih algoritmov, nismo pa še opisali samega delovanja. GA začne z začetnim naborom rešitev oziroma kandidatov. Celoten nabor imenujemo populacija. Vsaka posamezna rešitev ali osebek v tem naboru je rešitev na problem, ki ga algoritem rešuje. Tipično je začetna populacija zapolnjena z naključno generiranimi rešitvami, ki so bolj ali manj uspešne. Začetno populacijo imenujemo prva generacija. Da lahko izmerimo koristnost oziroma kvaliteto posamezne rešitve, uporabimo funkcijo uspešnosti (fitnes funkcija). Višja fitnes vrednost osebka pomeni, da predstavlja boljšo rešitev za dani problem. Fitnes populacije se izmeri kot povprečje fitnesa vseh posameznikov v populaciji. V pravilno delujočem GA se bo fitnes populacije višal vzporedno z generacijami. Isto velja tudi za posamezen najboljši in najslabši fitnes v populaciji. Generacija predstavlja število iteracij, ki jih je GA že opravil. Načeloma se rešitve v vsaki iteraciji izboljšujejo, dokler ne dokončno konvergirajo ali pa postanejo izboljšave praktično neopazne. Zato je potrebno uvesti končni kriterij, ob katerem se izvajanje ustavi. Na spodnji sliki (Slika 3.1) je prikazan osnoven potek delovanja GA, kjer je nazorno prikazana iterativnost izvajanja, ki je ključna za uspešnost tehnike. [15]



Slika 3.1: Diagram poteka za genetski algoritem. [16]

Kot smo že omenili, je najpogostejši način inicializacije populacije naključno generiranje rešitev. Včasih pa lahko v začetno populacijo vstavimo specifične kandidate - npr. ko želimo najti izboljšavo za že obstoječo rešitev [12]. Kvaliteta ni pomembna za začetni nabor rešitev, veliko pomembnejša je raznolikost (angl. *diversity*) teh rešitev. Vsak gen prispeva h karakteristikam osebkov. Te karakteristike so dobre ali slabe in ustrezno višajo ali nižajo fitnes vrednost posameznih rešitev. Celotno delovanje GA je namenjeno generiranju različnih kombinacij teh karakteristik. Če torej neka karakteristika (možna vrednost v enem izmed genov), ki zelo dobro vpliva na kvaliteto rešitve, sploh ni zastopana v populaciji, bo delovanje GA trpelo. V najslabšem primeru se ta karakteristika ne bo nikoli odkrila in bo povsem izključena iz končne rešitve, ali pa bo operacija mutacije uvedla to karakteristiko v populacijo, v katerem primeru bo posledica le počasnejša optimizacija. Velika raznolikost osebkov v populaciji je zato primarna zahteva za uspešno delovanje GA in naključno generiranje je preprost način za zagotavljanje te lastnosti v začetni populaciji. [15]

Število osebkov v populaciji je tudi pomembno. Zelo veliko število pomeni veliko več genetskih operacij in drugih izračunov med dvema populacijama. Dobra lastnost velike populacije pa je večja raznolikost osebkov v populaciji. Poleg že omenjene pomembnosti raznolikosti med osebki v začetnih fazah izvajanja GA, lahko to vodi tudi v zgodnjo konverzijo ali zgodnje generiranje zelo dobrih rešitev. Na drugi strani so majhne populacije manj časovno zahtevne za izračun, vendar ne omogočajo velike raznolikosti med osebki. [15]

Preoblikovanje populacije iz ene generacije v naslednjo je doseženo z genetskimi operatorji. Iz slike (Slika 3.1) je razvidno, da funkciji uspešnosti in preverjanju zaustavitvenega pogoja sledi (če pogoj ni izpolnjen) selekcija. Operacija izbere pare osebkov iz trenutne populacije, ki se bodo reproducirali in ustvarili nove osebke za naslednjo populacijo. Selekcija je pomemben korak, zato je nastalo več strategij izbire staršev (opisani v poglavju 3.6 (Selekcija)) vendar so cilji vseh enaki – križanje dobrih osebkov v upanju, da bodo potomci tudi dobre ali še boljše rešitve. Selekcija deluje na osnovi fitnes vrednosti. Osebek z višjo vrednostjo ima večjo verjetnost, da bo izbran (lahko tudi večkrat), medtem ko slabši osebki morda sploh ne bodo. Takšno delovanje je ponovno osnovano na naravi, kjer imajo močnejši predstavniki posameznih živalskih vrst večje možnosti razmnoževanja. Bolj kot se neki osebek razmnožuje, več potomcev bo imel in ti bodo zasedali večji delež osebkov v naslednji populaciji. Ker so novi osebki potomci dobrih staršev, bodo tudi sami najverjetneje močni posamezniki. Vsaka generacija torej vsebuje

relativno velik delež osebkov, ki izhajajo iz manjše množice najboljših posameznikov prejšnje generacije. Na takšen način se v naravi in GA krepi kvaliteta vsake populacije. [15]

Rezultat selekcije so pari staršev, katerih dedni material se v operaciji križanja zmeša in tvori potomce. Križanje je na sliki (Slika 3.1) skupaj z mutacijo združeno v korak reprodukcije. Cilj križanja je združitev lastnosti staršev v nov osebek v upanju, da se bodo prenesle boljše karakteristike obeh staršev, medtem ko bodo slabše izpuščene. Podobno kot pri selekciji so se tudi tu razvile različne metode križanja, ki bodo podrobneje predstavljene v poglavju 3.7 (Križanje). Številne karakteristike pogosto niso prisotne v populaciji, med njimi tudi takšne, ki bi bile zaželene v dobri rešitvi. Vzrok za to je lahko, da niso bile nikoli generirane, ali pa da so tekom izvajanja izumrle. Takšne karakteristike je potrebno zato ponovno uvesti v populacijo. Križanje zmeša karakteristike dveh že obstoječih staršev v novo rešitev. Pri tem se ne ustvarijo nove karakteristike, temveč zgolj nove kombinacije teh. Mutacija je operator, ki v populacijo vključuje naključne karakteristike. V preprostih genetskih algoritmih, kjer so kromosomi predstavljeni kot nizi bitov, je mutacija preprosto obračanje bitov na naključnem mestu v genomu. Nova karakteristika je lahko dobra ali slaba. Če je dobra bo izboljšala fitnes osebka in vseh potomcev, s tem povečala njihovo možnost reprodukcije ter posledično preživela. V nasprotnem primeru bo osebek utrpel zmanjšanje fitnesa in možnosti reprodukcije. Tudi če osebek vseeno uspešno ustvari potomce, bodo tudi ti imeli poslabšano fitnes vrednost, saj bodo vsebovali to karakteristiko in posledično bo ta prej ali slej izumrla. [15]

GA je v osnovi zelo generična metoda, kjer je mogoče številne aspekte implementirati na različne načine (npr. predstavitev kromosomov, strategija selekcije, tip mutacije in križanja itd.), glede na reševan problem [16]. V naslednjih poglavjih bomo podrobneje pogledali gradnike GA in kakšne možnosti implementacij so na razpolago. [15]

3.3 Funkcija uspešnosti

Funkcija uspešnosti meri kvaliteto posameznih rešitev v populaciji in se izrazi kot fitnes vrednost. Tipično bo dobra rešitev imela visok fitnes, slaba rešitev pa nizkega. Uporabi se lahko tudi obratno razmerje. Z uporabo fitnes funkcije lahko razporedimo rešitve v zaporedje od najboljše do najslabše, ki se zatem uporabi v operaciji selekcije. S funkcijo uspešnosti lahko tudi preverjamo obnašanje GA skozi čas ali generacije. Na podlagi fitnesa posameznih rešitev v populaciji nas navadno zanimajo tri vrste fitnesa. Prvi je najboljši

fitnes. Najboljše rešitve imajo najvišji fitnes v celotni populaciji. Najboljši fitnes pokaže uspešnost algoritma in trenutno najboljše rešitve. Ob zaključku izvajanja postane osebek z najvišjim fitnesom končni rezultat. Naslednji tip fitnesa je povprečni fitnes, ki poda informacijo o kvaliteti povprečne rešitve v populaciji. Tretja in zadnja oblika je najnižji fitnes. Ta pripada najslabšemu osebku v celotni populaciji. Praviloma se vse tri oblike fitnesa čez čas povečujejo (rešitve se optimizirajo). Proti koncu (če se izvajanje ne predčasno zaustavi) se doseže optimalna vrednost (najdena najboljša rešitev) in fitnes vrednosti konvergirajo (ker se začne najboljša rešitev številčno reproducirati). Pri ustvarjanju funkcije uspešnosti je potrebno biti pozoren na neveljavne rešitve in poskrbeti, da bodo takšne vedno imele najnižjo fitnes vrednost. [15]

3.4 Zaustavitveni pogoj

Genetski algoritem je iterativna tehnika, ki se izvaja iz generacije v generacijo brez prestanka. Rešitve se med izvajanjem sicer izboljšujejo, vendar bi se brez zaustavitvenega pogoja (angl. *stopping condition*) izvajanje nadaljevalo v nedogled. Zato je potrebno sprejeti odločitev, kdaj naj se delovanje ustavi in to opredeliti v zaustavitvenem pogoju. Osebek z največjo fitnes vrednostjo ob koncu izvajanja postane končni rezultat GA (čeprav lahko pridobimo in uporabimo tudi ostale rešitve, če je prva iz določenega razloga neustrezna). Zaustavitveni pogoj lahko definiramo na različne načine: [15]

- Če so rezultati, ki nastajajo pri izvajanju dovolj dobri in nadaljnja optimizacija ni potrebna ali ni verjetno, da bo imela velik doprinos, lahko izvajanje ustavimo. Zaustavitveni pogoj v tem primeru predstavlja kvaliteto rešitve, za katero želimo, da jo GA doseže. Pogoj je potrebno nastaviti pred začetkom izvajanja in ko bo ta izpolnjen, se bo GA ustavil.
- Če izvajanje preseže določen čas delovanja ali število generacij, lahko izvajanje zaključimo.
- Če je izboljšanje v določenem številu generacij ali času premajhno in je populacija začela konvergirati, lahko izvajanje zaključimo.

3.5 Skaliranje fitnesa

Kot smo že povedali, je naloga selekcije izbira staršev, ki bodo generirali dobre rešitve v naslednji populaciji. Operator se za svoje delovanje opira na funkcijo uspešnosti, tako da pogosteje izbira osebke z višjo fitnes vrednostjo. Problem nastane, ko takšno delovanje

vodi v zelo pogosto izbiro osebkov z najvišjimi fitnes vrednostmi. Posledično bi v naslednji generaciji prevladovali potomci, ki bi si bili med seboj zelo podobni, ker bi izhajali iz majhne skupine staršev in celotna populacija bi zelo hitro konvergirala. Osebki, ki so imeli dobre karakteristike, vendar so bili šele v procesu izboljšave, bi v tem primeru izumrli, vključno s številnimi dobrimi karakteristikami, ki so jih nosili. Da preprečimo takšno delovanje, je potrebna dodatna metoda. Skaliranje fitnesa (angl. *fitness scaling*) je en izmed načinov odprave opisane težave. Deluje tako, da osebke v populaciji skalira glede na njihovo fitnes vrednost. To ustvari t. i. pričakovano število za vsakega posameznika v populaciji. Pričakovano število predstavlja fitnes vrednost v novem merilu oziroma skali in jo zamenja kot osnovo za izbiro staršev v operaciji selekcije. V nadaljevanju si bomo pogledali nekaj pogostih metod, ki skalirajo fitnes glede na potrebe. [15]

3.5.1 Skaliranje z rangiranjem

Pri skaliranju z rangiranjem (angl. *rank scaling*) so osebki v populaciji rangirani z namenom preprečitve prehitre konvergence. Vsak posameznik je v naraščajočem vrstnem redu glede na fitnes razporejen v rang med 1 in N, na podlagi katerega se izračuna pričakovano število za vsako rešitev v populaciji. Enačba za izračun je predstavljena v (3.1). [14] [15]

$$PSt(i,t) = Min + (Max - Min)\frac{rang(i,t) - 1}{N - 1} \tag{3.1}$$

Tu je:
$PSt(i,t)$ – pričakovano število osebka i ob času t,
Min – najmanjše možno pričakovano število,
Max – največje možno pričakovano število,
$rang(i,t)$ – rang osebka i ob času t in
N – število osebkov v populaciji.

Osebek s prvim rangom bo kot pričakovano število prejel Min, osebek z rangom N pa Max [15]. Skaliranje z rangiranjem se znebi absolutnih razlik med fitnes vrednostmi in uvede pričakovano število, kjer je razlika med vrednostmi dveh zaporednih članov (rang i in i+1) konstantna, neglede na to ali je bila absolutna razlika fitnesov med obema osebkoma velika ali majhna. Prednost takšnega delovanja je upočasnjena konvergenca, ker razlike med osebki v populaciji niso več tako izrazite. Rangiranje zniža razlike za verjetnost selekcije pri

populacijah, kjer so razlike med fitnes vrednostmi velike, in jih poveča, ko je med njimi majhna raznolikost. Možna slabost uporabe skaliranja z rangiranjem je počasnejše odkrivanje zelo dobrih rešitev. Na drugi strani pa povečano ohranjanje raznolikosti osebkov v populaciji pogosto vodi v boljše rezultate GA. [14]

3.5.2 Proporcionalno skaliranje

Proporcionalno skaliranje (angl*. proportional scaling*) neposredno spremeni merilo fitnes vrednosti. Pričakovano število je zato proporcionalno z vrednostmi funkcije uspešnosti. Izrazi se ga kot: [15]

$$PSt(i,t) = \alpha\, f(i,t) \tag{3.2}$$

Tu je:
$PSt(i,t)$ – pričakovano število osebka i ob času t,
α – poljubna konstanta in
$f(i,t)$ – fitnes vrednost osebka i ob času t.

Proporcionalno skaliranje ima težave s populacijami, kjer fitnes posameznih osebkov ni v dobrem območju rešitev, in ko ima majhno število osebkov veliko boljšo fitnes vrednost od preostanka populacije. Slednje vodi v veliko porast teh osebkov v naslednjih generacijah in hitro konvergenco, o čemer pa smo že govorili. [15]

3.5.3 Skaliranje vrha

Skaliranje vrha (angl*. top scaling*) izbere določen delež najboljših osebkov iz populacije in jih skalira na isti način, tako da je pričakovano število enako $1/N$. Vsem ostalim osebkom se pričakovano število nastavi na 0. Enačba za takšno delovanje je: [15]

$$PSt(i,t) = \begin{cases} 1/N, & \text{če je } i \text{ znotraj } 100 \times \alpha\%\ N \\ 0, & \text{če } i \text{ ni znotraj omenjenega območja} \end{cases} \tag{3.3}$$

Tu je:
$PSt(i,t)$ – pričakovano število osebka i ob času t,
α – poljubna konstanta med 0 in 100 ter
N – število osebkov v populaciji.

Takšen izračun pričakovanega števila pomeni, da je le določen del najboljših osebkov sposoben sodelovati v selekciji. Preostali osebki kot rezultat skaliranja ne bodo nikoli izbrani. Število osebkov, ki so vključeni v selekcijo, se določi kot delež celotne populacije. Vse izbrane rešitve imajo enako pričakovano število in posledično verjetnost izbire v operaciji selekcije. [15]

3.5.4 Skaliranje s sigmoidno funkcijo

Skaliranje s sigmoidno funkcijo (angl. *sigma scaling*) je namenjeno preprečevanju prezgodnje konvergence v GA. Preoblikovanje fitnes vrednosti v pričakovano število izrazimo v (3.4). [14]

$$PSt(i,t) = \begin{cases} 1 + \dfrac{f(i) - \bar{f}(t)}{2\sigma(t)} & \text{če } \sigma(t) \neq 0 \\ 1.0 & \text{če } \sigma(t) = 0 \end{cases} \tag{3.4}$$

Tu je:
$PSt(i,t)$ – pričakovano število osebka i ob času t,
$f(i)$ – fitnes vrednost osebka i,
$\bar{f}(t)$ – povprečen fitnes v populaciji ob času t in
$\sigma(t)$ – standardni odklon fitnesa v populaciji ob času t.

Če je nastalo pričakovano število negativno, je mogoče to vredno spremeniti v zelo majhno pozitivno vrednost, tako da imajo tudi najslabše rešitve v populaciji možnost reprodukcije, čeprav zelo majhno. Rezultat skaliranja s sigmoidno funkcijo je enak kot pri skaliranju z rangiranjem. Ob začetku izvajanja, ko je standardni odklon fitnes vrednosti v populaciji tipično visok, najboljši osebki ne bodo imeli pretirane prednosti pri selekciji. Kasneje v izvajanju, ko je populacija že nekoliko konvergirala in osebki v populaciji niso več tako raznoliki, bo skaliranje s sigmoidno funkcijo najboljšim rešitvam dalo prednost, ki je sicer ne bi imeli in tako omogočilo nadaljevanje evolucije. [14]

3.6 Selekcija

O osnovah in vlogi selekcije v GA smo že govorili v poglavju 3.2 (Koncept in osnove genetskih algoritmov), kjer smo kot nalogo operatorja izpostavili tvorjenje parov osebkov, nad katerimi se zatem izvede križanje. Dobra selekcijska tehnika poveča verjetnost izbire dobrih rešitev iz populacije. Ti osebki se reproducirajo in ustvarijo potomce z njihovimi karakteristikami, medtem ko slabši osebki izumrejo. Za ta namen so se razvile številne tehnike, katerih nekaj najbolj pogosto uporabljanih bomo predstavili v nadaljevanju. [15]

3.6.1 Selekcija z ruleto

Pri selekciji z ruleto (angl*. roulette wheel selection*) je verjetnost izbire posameznega osebka v populaciji neposredno sorazmerna s pričakovanim številom. Vsak posameznik predstavlja del ruletnega kolesa. Višja kot je osebkova fitnes vrednost, večja je površina na ruletnem kolesu, ki mu pripada. Pri izvajanju selekcije z ruleto se kolo zavrti N-krat, kjer je N število osebkov v populaciji. Pri vsakem vrtenju zmaga osebek, na čigar površini se je ustavila kroglica. Osebki z večjo prisotnostjo na kolesu so zato pogosteje izbrani. Sama implementacija je seveda nekoliko drugačna. Kolo rulete se vzpostavi kot seštevek vseh pričakovanih števil – imenujmo ga T. Zatem selekcija izbere naključno število r, ki je med 0 in T. Sledi postopno seštevanje pričakovanih števil osebkov. Izbrana je rešitev, katere prišteto pričakovano število prvo preseže r. [15]

3.6.2 Stohastično univerzalno vzorčenje

Selekcija z ruleto se lahko v pravih pogojih sooča s težavo. Način delovanja selekcije z ruleto omogoča izbiro slabih osebkov. Verjetnost, da se bo to zgodilo, je zelo majhna, vendar zaporedje slabih izbir lahko pripelje do tega, da v populaciji ostanejo zgolj še slabi osebki. Iz tega razloga je nastalo stohastično univerzalno vzorčenje (angl*. stochastic universal sampling*), ki je osnovano na selekciji z ruleto. Stohastično univerzalno vzorčenje deluje na enak način kot selekcija z ruleto, vendar z eno izjemo. Razlika je, da v primeru stohastičnega univerzalnega vzorčenja izberemo N osebkov ob enem vrtenju kolesa, kjer smo prej izbrali zgolj enega. Namesto da N-krat zavrtimo kolo, zato da izberemo N staršev, s stohastično univerzalnim vzorčenjem zavrtimo kolo enkrat in izberemo N staršev. Pri tem morajo biti kazalci, ki izberejo starše, enakomerno porazdeljeni. V nadaljevanju je prikazan

primer implementacije takšnega delovanja v programskem jeziku Java (Koda 3.1), kjer je i indeks za člane populacije in $PSt(i,t)$ vrne pričakovano število osebka i ob času t [17]. [14, 15]

```
ptr = Math.random(); //Vrne naključno število iz območja [0,1]
      for(sum = i = 0; i < N; i++)
            for (sum += PSt(i, t); sum > ptr; ptr++)
                  Select(i);
```

Koda 3.1: Primer stohastičnega univerzalnega vzorčenja.

Stohastično univerzalno vzorčenje ima, tako kot selekcija z ruleto, še vedno težave z izbiro osebkov, ki je sorazmerna z njihovimi fitnes vrednostmi. To je težava velikega standardnega odklona povprečnega fitnesa v zgodnjih populacijah, ki vodi v hitro množenje najboljših osebkov. Posledica je že nekajkrat omenjen problem zmanjšanja raznolikosti osebkov in prezgodnja konvergenca. [14]

3.6.3 Selekcija z rangiranjem

Selekcija z rangiranjem (angl. *rank selection*) je podobna selekciji z ruleto, le da so posamezne rešitve skalirane v range. Osebki z višjim fitnesom imajo večjo možnost reprodukcije, vendar ta način selekcije odstrani absolutne razlike med njimi. Tako se nekaj zelo dobrih rešitev v delovanju GA ne bo zgodaj razmnožilo čez celotno populacijo. Čeprav so te rešitve veliko boljše od drugih, nimajo tako velike prednosti pri selekciji, kot bi nakazovale njihove fitnes vrednosti. Razlog za takšno delovanje je enakomerna porazdelitev rangov. Selekcija z rangiranjem torej rešuje problem s prezgodnjo konvergenco, ki je pestil prejšnje načine selekcije. Deluje s pomočjo skaliranja z rangiranjem, ki smo ga predstavili v poglavju 3.5 (Skaliranje fitnesa) in kjer je tudi predstavljen način razvrščanja osebkov. [14, 15]

3.6.4 Turnirska selekcija

Vse s fitnesom sorazmerne metode selekcije, ki smo jih opisali v tem podpoglavju, za svoje delovanje potrebujejo dva prehoda čez podatke v vsaki generaciji (prvi za funkcijo uspešnosti in drugi prehod za izračun pričakovanega števila). V selekciji z rangiranjem je potrebno v vsaki generaciji sortirati celotno populacijo, kar je potencialno časovno zahtevno

opravilo. Turnirska selekcija (angl. *tournament selection*) je v končnem rezultatu zelo podobna selekciji z rangiranjem, vendar je učinkovitejša, ko pride do izrabljanja virov in zato primernejša za paralelno implementacijo. Tako kot selekcija z rangiranjem tudi turnirska selekcija nima težav s predčasno konvergenco. [14]

V turnirski selekciji se najprej naključno izbereta dva osebka iz populacije. Zatem se izbere še naključna vrednost r, ki naj bo med 0 in 1. Če je r manjši kot predhodno izbran k (tipično 0,75), potem je kot eden izmed staršev izbran osebek z boljšo fitnes vrednostjo, sicer pa je izbran slabši izmed obeh osebkov. Oba se zatem vrneta v populacijo in sta lahko izbrana ponovno. To delovanje lahko prilagodimo tudi tako, da bosta naenkrat tekmovala več kot dva osebka. Tudi v tem primeru je potrebno poskrbeti, da ima najboljša rešitev najvišjo verjetnost, da bo izbrana. Verjetnosti izbire za ostale kandidate se ustrezno razvrstijo za tem. [14, 15]

3.7 Križanje

Križanje je v GA na splošno sprejeto kot operacija, ki je odgovorna za uvajanje sprememb in inovativnosti v GA. To ne pomeni, da preostanek komponent ni pomemben. Bolj kot katerikoli posamezen genetski operator, je za uspešen GA pomembno pravo ravnovesje med križanjem, mutacijo in selekcijo. Križanje je zgolj temeljni kamen okrog katerega so bili razviti GA. Križanje je namreč značilen operator v GA, ki je le-tega zgodovinsko razlikoval od ostalih evolucijskih algoritmov (modernejši EA uporabljajo tudi križanje). [13, 14]

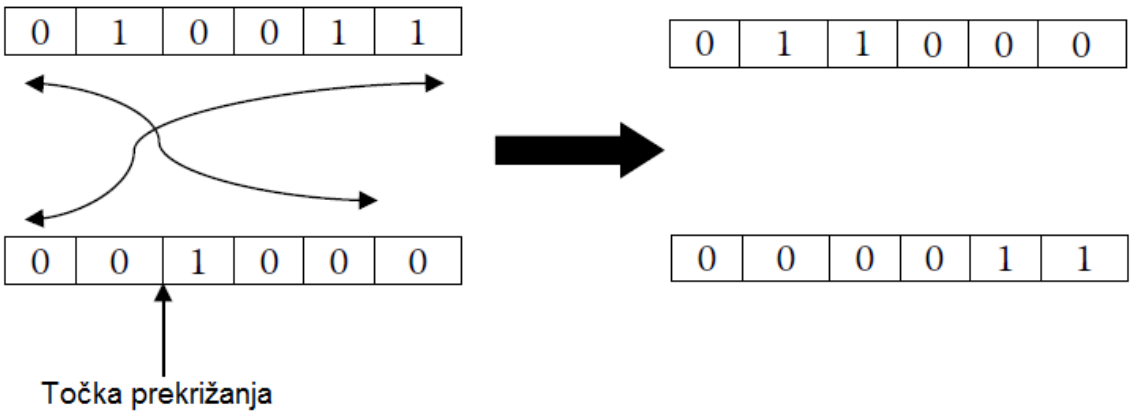
Ideja za križanjem je precej preprosta: vzamemo dva starša in iz njiju, s kombiniranjem genov iz obeh staršev, generiramo nov osebek. Cilj je ustvariti osebek, ki bo združeval najboljše karakteristike obeh staršev. Lastnosti, ki najboljše vplivajo na kvaliteto rešitev, jasno niso znane (sicer bi bilo izvajanje GA brezpredmetno), zato mora operacija križanja združevati karakteristike naključno. Včasih se v osebek združijo slabe lastnosti - v takšnem primeru ti potomci ne bodo preživeli dolgo. Na drugi strani se včasih ustvari rešitev iz najboljših karakteristik obeh staršev in nastane osebek, ki je boljši kot oba starša. [13, 15]

S križanjem je povezana verjetnost križanja - p_k (angl. *probability of crossover*). Verjetnost križanja odloča, kakšen odstotek populacije se bo kot posledica križanja, prenesel v naslednjo generacijo. Ta delež je tipično precej visok, saj je operacija križanja precej predvidljiva in ne vsebuje veliko naključnega delovanja. Verjetnost križanja je visoka, zato

da ohranjamo raznolikost in omogočimo algoritmu, da konvergira ob pravem času. Preostanek osebkov (delež osebkov, ki se ne prenese preko križanja) v populacijo vnesejo mutacije in drugi genetski operatorji. V nadaljevanju so predstavljene nekatere osnovne metode križanja. [15]

3.7.1 Enomestno križanje

Enomestno križanje (angl. *single-point crossover*) je najosnovnejša oblika križanja. Naključno se izbere število, ki je med 0 in n, kjer je n dolžina kromosoma. Izbrano mesto imenujemo točka prekrižanja (angl. *crossover point*). Geni na levi strani točke prekrižanja se v nov osebek prenesejo iz prvega starša, medtem ko se tisti na desni strani točke vzamejo iz drugega starša. Enomestno križanje je preprosto, vendar ima svoje težave. Največje so: nezmožnost ustvarjanja vseh možnih kombinacij kromosomov, težave z dolgimi kromosomi in preferenčna obravnava nekaterih lokusov (začetek in konec genoma). Proces je prikazan na spodnji sliki (Slika 3.2). [14, 15]

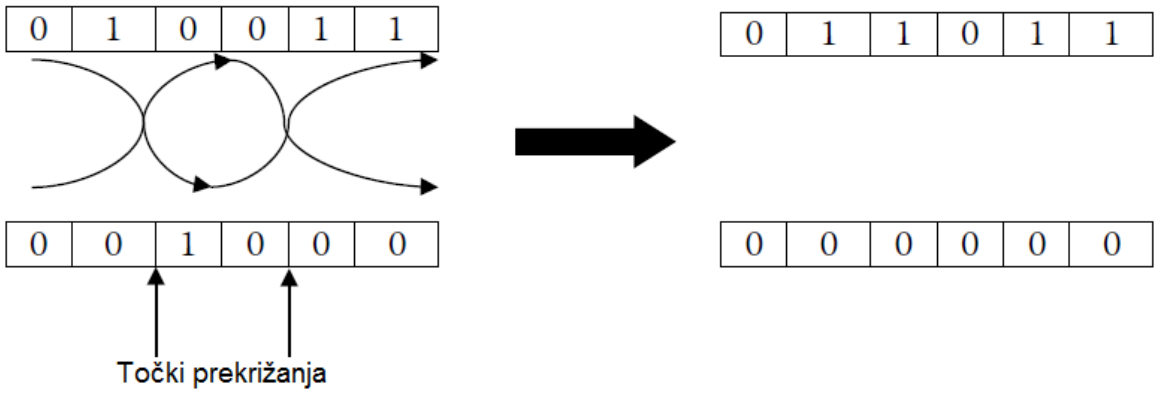


Slika 3.2: Enomestno križanje. [15]

3.7.2 Dvomestno križanje

Dvomestno križanje (angl. *two−point crossover*) je nastalo, kot odgovor na pomanjkljivosti enomestnega križanja. Dvomestno križanje teh težav sicer ne odpravi, vendar jih bistveno zmanjša. Kot nakazuje že ime, je dvomestno križanje zelo podobno enomestnemu, le da si namesto ene točke prekrižanja izberemo dve. Potomci se zatem tvorijo iz dednega

materiala med obema točkama enega starša in preostalimi geni drugega starša, kot prikazuje naslednja slika (Slika 3.3). [15]



Slika 3.3: Dvomestno križanje.

3.7.3 Uniformno križanje

Ena glavnih pomanjkljivosti prejšnjih dveh načinov križanja je odvisnost učinkovitosti križanja od pozicije genov v genomu. Z drugimi besedami: geni, ki so blizu eden drugemu imajo veliko večjo možnost, da bodo podedovani skupaj, kot če bi se nahajali na različnih koncih genoma. To omejuje nabor osebkov, ki so lahko generirani s prejšnjimi načini križanja. En način križanja, kjer lokus posameznega gena nima nobenega vpliva na delovanje, je uniformno križanje (angl*. uniform crossover*), včasih imenovano tudi razpršeno križanje (angl*. scattered crossover*). V osnovi je uniformno križanje nadgraditev dvomestnega križanja, tako da postane število točk prekrižanj naključno. To dosežemo tako, da za vsak gen, ki ga potomec lahko podeduje, vržemo kovanec. V primeru, da je rezultat meta glava, se gen deduje od prvega starša, sicer pa od drugega. Posledica takšnega delovanja je precejšen razkroj (preveliko mešanje genov). Na srečo lahko to enostavno nadzorujemo z uvedbo dodatne vrednosti, ki ureja verjetnosti meta kovanca. Pri vrednosti 0,0 razkroja ni in bo otrok popolna kopija drugega starša (če vrednost ureja verjetnost meta glave). Pri vrednosti 0,5 (ki velja za uniformno križanje) pa je razkroj največji in v tem primeru bo otrok v povprečju vseboval polovico genov od vsakega starša. Način križanja, ki omogoča nastavljanje razkroja, je poimenovan parametrizirano uniformno križanje (angl*. parameterized uniform crossover*). [15, 18]

3.7.4 Vmesno križanje

Vmesno križanje (angl. *intermediate crossover*) je običajno uporabljeno na genih s številsko predstavitvijo. Pri takšnem križanju bodo geni potomcev imeli vrednosti, ki bodo v območju med geni njihovih staršev. Tu je način izračuna gena pri vmesnem križanju: [15]

$$c = p_1 + r\ \alpha\ (p_2 - p_1) \tag{3.5}$$

Tu je:
c – gen otroka, ki bi ga radi pridobili,
p_1 – gen prvega starša,
p_2 – gen drugega starša,
r – naključna vrednost med 0 in 1 ter
α – fiksno razmerje.

Nastali gen bo zagotovo imel vrednost med genoma obeh staršev, če je α med 0 in 1. [15]

3.7.5 Hevristično križanje

Hevristično križanje (angl. *heuristic crossover*) je še en primer križanja, ki je primerno za uporabo s številskimi geni. Posebnost te metode je usmerjanje iskanja z upoštevanjem fitnes vrednosti staršev. Nastali gen bo na območju med vrednostmi obeh staršev. Oddaljenost od vsakega starša je določena z vrednostjo α. Pri izračunu novega gena je potrebno zadovoljiti pogoj, da p_1 ni slabši od p_2. Matematičen zapis hevurističnega križanja je predstavljen v (3.6). [13, 15]

$$c = p_1 + \alpha(p_1 - p_2) \tag{3.6}$$

Tu je:
c – gen otroka, ki bi ga radi pridobili,
p_1 – gen starša, ki ni slabši od p_2,
p_2 – gen starša, ki je enak ali slabši od p_1,
α – vrednost med 0 in 1, ki jo izbere uporabnik glede na želeno delovanje.

3.8 Mutacija

Mutacija je operacija dodajanja novih karakteristik v populacijo. To dosežemo z naključnim spreminjanjem vsebine genov. Če so vnesene spremembe koristne, bodo preživele in nadaljevale svoj obstoj v naslednjih populacijah. Nezaželene karakteristike bodo skozi generacije prej ali slej izumrle. Kot smo že rekli, je križanje najpomembnejši in značilen operator v GA, medtem ko mutacija deluje bolj v ozadju in ni tako pomembna (mutacija je veliko pomembnejši operator v drugih EA). Mutacija je redka operacija in se izvede le nad nekaj osebki v vsaki populaciji. Je naključna in nepredvidljiva operacija, ki v prevelikem številu spremeni delovanje GA v skoraj naključno iskanje. Razlog je preveliko število deformacij v populaciji in posledično uničenje smeri iskanja, ki so jo vnesli križanje in drugi operatorji. Količina mutacij, ki se izvede, je opredeljena v stopnji mutacije (angl*. mutation rate*), ki je predstavljena s številčno vrednostjo med 0 in 1. Kot že rečeno, morajo biti mutacije redke, zato je potrebno stopnjo mutacij nastaviti na nizko vrednost. Operacija mutacije gre skozi vse osebke v populaciji in izbere sodelujoče z verjetnostjo stopnje mutacije. Sledi nekaj najpogostejših strategij mutacije. [14, 15]

3.8.1 Uniformna mutacija

Uniformna mutacija (angl*. uniform mutation*) je najpogostejša oblika operacija mutacije. Če je genotip predstavljen kot bitni niz, se izbrana vrednost v osebku ob mutaciji preprosto obrne. Če je bila vrednost prej 0, je po mutaciji 1 in obratno. Takšno delovanje je predstavljeno na spodnji sliki (Slika 3.4). Pri mutaciji genov, ki so predstavljeni s številskimi vrednostmi je potreben drugačen pristop. Stara vrednost se v tem primeru nadomesti z novim naključnim številom v dovoljenem rangu. [15]

Slika 3.4: Uniformna mutacija nad bitnim nizom. [15]

3.8.2 Gaussova mutacija

Gaussova mutacija (angl. *Gaussian mutation*) je operacija, primerna le za številsko predstavljene gene. Ob mutaciji se vrednosti gena prišteje naključna vrednost. Ta vrednost je pridobljena s pomočjo Gaussove porazdelitve z aritmetično sredino 0. Ta običajno generira zelo majhna pozitivna in negativna števila. Posledično so tudi spremembe na mutiranem genu zelo majhne. V redkih primerih pa se zgodi, da je naključno pridobljena vrednost velika in v tem primeru je tudi sprememba na mutiranem osebku večja. [15]

3.8.3 Spremenljiva stopnja mutacije

Stopnjo mutacije lahko spreminjamo tudi med samim izvajanjem GA. Spremenljiva stopnja mutacije (angl. *variable mutation rate*) se lahko uporabi za upravljanje delovanja GA. Kot smo omenili na začetku tega poglavja, je mutacija operator, ki vnaša v populacijo naključnost. Govorili smo tudi o tem, kako naključnost škoduje delovanju GA, vendar je v začetnih fazah izvajanja, kjer želimo raziskati karseda različne možne rešitve, lahko naključnost zaželena. Proti koncu izvajanja algoritma pa je najboljša posebej nizka stopnja mutacij, kar dovoljuje populaciji, da konvergira. S spremenljivo stopnjo mutacije to dosežemo tako, da tekom izvajanja spreminjamo verjetnost, s katero so osebki izbrani za mutacijo. [15]

3.9 Drugi genetski operatorji

Selekcija, križanje in mutacija so glavni genetski operatorji. V tem poglavju bomo namenili nekaj besed še drugim, manj poznanim operatorjem, ki so navadno uporabljeni za specializirane namene. Ti operatorji dodajo prilagodljivost delovanju GA.

3.9.1 Elitizem

Elitizem (angl. *eliticism*) izbere določen delež najboljših osebkov v populaciji in jih brez križanja ali mutiranja prenese v naslednjo generacijo. To preprečuje, da bi se najboljši osebki, pri operaciji križanja ali mutacije, poslabšali in zagotovi, da so dosegljivi za naslednje generacije. Ker se vedno prenesejo najboljše rešitve, smo ob koncu izvajanja lahko prepričani, da je rezultat resnično najboljša rešitev, ki jo je GA odkril. Brez elitizma je

mogoče ustvariti osebke, ki niso tako dobri kot nekatere predhodne rešitve (to še vedno velja za osebke, ki niso v deležu elitnih rešitev). [15]

3.9.2 Vstavi in izbriši

Operator vstavljanja (angl. *insert*) omogoča dodajanje novih osebkov v populacijo. Glede na vmesne rezultate GA lahko generiramo nove rešitve z visoko fitnes vrednostjo. Te nove osebke lahko zatem vključimo v populacijo z operacijo vstavljanja. Izbriši (angl. *delete*) deluje ravno obratno in omogoča odstranitev osebkov iz populacije. To je mogoče najbolj zaželeno v primeru osebkov z zelo nizkimi fitnes vrednostmi. [15]

3.9.3 Trda in mehka mutacija

V številnih primerih lahko operator mutacije razdelimo na dva dela. Prvi del oziroma mehka mutacija (angl. *soft mutation*) je odgovorna za majhne spremembe v osebkih. Ker so spremembe majhne, lahko pričakujemo, da bodo tudi učinki na rešitve in njihovo obnašanje relativno majhne. Ta operator se lahko uporablja kot pomoč pri konvergenci populacije ali kot način vstavljanja novih karakteristik v sistem. Drugi del oziroma trda mutacija (angl. *hard mutation*) ustvarja velike spremembe v posameznikih. To lahko povsem spremeni osebek in njegovo uspešnost. To obliko mutacije je potrebno uporabljati še posebej redko in je namenjena samo vnašanju novih karakteristik v populacijo. [15]

3.9.4 Popravilo

Zaradi narave delovanja GA se pogosto zgodi, da med izvajanjem nastane rešitev, ki je nemogoča oziroma neveljavna. V takšnem primeru lahko uporabimo popravilo (angl. *repair*), ki popravi neveljavne osebke, preden so vneseni v populacijo. Operacija spremeni neveljavno rešitev v obliko, ki je sprejemljiva. Za to je potrebno specifično poznavanje omejitev problema in razumevanje, zakaj posamezna rešitev ni veljavna. Operacija nad neveljavnim osebkom izvede karseda majhne spremembe, ki naredijo rešitev veljavno. [15]

3.10 Delovanje algoritma

Do sedaj je to poglavje opisalo vse komponente GA, njihovo delovanje in kako ustvarjajo nove ter boljše rezultate. To poglavje bo opisalo še, kako algoritem deluje in konvergira populacijo, da najde najboljšo rešitev.

Ne glede na to kako je posamezen genotip oziroma rešitev predstavljena v GA, je ta sestavljena iz več manjših delov – genov, na katere lahko gledamo kot gradnike, ki sestavljajo celoto. Vsak takšen gradnik prispeva neko karakteristiko posamezniku, katerega del je. Če torej za primer vzamemo genotip, ki je v celoti sestavljen iz znakov 0 in 1, je populacija začetnih kandidatov množica nizov, ki sestojijo iz naključnega vrstnega reda možnih vrednosti. Populacija je zagotovo premajhna, da bi vsebovala vse možne rešitve, ampak obstaja velika verjetnost, da sta 0 in 1 zastopani na vsakem mestu v kromosomu (če nista, bo to mutacija s časom popravila). V tej točki je populacija povsem naključna in v njej ni mogoče opaziti nobenega reda. Stanje populacije lahko opazujemo s pomočjo sheme, ki je v osnovi izpis kromosoma z znakom * za poljubno vrednost ali pa predstavimo celotno iskalno območje v obliki n-dimenzionalnega grafa, kjer je n število genov v kromosomih. Osebki se na takem grafu prikažejo kot točke. Skozi izvajanje algoritma lahko opazujemo karakteristike rešitev, ki ostajajo v populaciji. V primeru, da je za končno rešitev zaželeno, da ima na prvem mestu znak 1, se bo število takšnih osebkov v populaciji povečevalo, medtem ko bodo osebki, ki se začnejo z 0, začeli izginjati iz populacije. Po določenem času bodo v populaciji ostali samo še osebki, katerih prvi znak je 1. Enako delovanje velja za vsa druga mesta v kromosomih in na takšen način algoritem konvergira. Hitrost, s katero posamezni geni konvergirajo, je odvisna od pomembnosti samih genov. Če ima gen velik vpliv na funkcijo učinkovitosti, potem bo ta gen tipično konvergiral hitreje kot gen, ki nima tako pomembne vloge. S takšnim delovanjem hitro zmanjšamo iskalni prostor algoritma – če vemo, da se rešitev začne z znakom 1, nam ni potrebno več iskati med rešitvami, ki se začnejo z 0. Na začetku izvajanja del boljših rešitev svoje karakteristike razširi med ostale osebke, tako da postane veliko osebkov med seboj podobnih v določenih genih, kar ima dve posledici. Prva je zbiranje osebkov na nekaj obetavnih mestih. To je dobra stvar, saj je sedaj veliko več osebkov namenjenih iskanju globalne rešitve v območjih z visokim fitnesom. Druga posledica je zmanjšanje iskalnega prostora, ker so za določene gene začetni osebki pokazali, da so njihove rešitve zelo dobre in posledično iskanje na tistem področju ni več potrebno. Na takšen način se iskalni prostor krči. Čez nekaj časa začnejo najboljše rešitve postajati bolj in bolj podobne med seboj, dokler celotna populacija dokončno ne konvergira v eno rešitev. Medtem poizkuša mutacija naključno spremeniti

karakteristike majhnega števila osebkov. Če se na takšen način odkrije nova dobra vrednost gena, se bodo drugi osebki kmalu pridružili iskanju v tem območju, sicer pa bo mutirana lastnost prej ali slej odstranjena iz populacije. [15]

3.11 Raznolikost

O raznolikosti je bilo povedanega že veliko. Izpostavili smo, kdaj je raznolikost zaželena in kdaj se je poizkušamo znebiti. Opisali smo jo kot različnost med osebki v populaciji, vendar nismo govorili o tem, kako izračunamo raznolikost v populaciji. V prejšnjem poglavju smo omenili, da lahko osebke v populaciji predstavimo kot točke v n-dimenzionalnem grafu. V takšni obliki lahko definiramo raznolikost kot povprečno razdaljo med točkami v grafu. Raznolikost je velika, če je razdalja med osebki velika. Ker n-dimenzionalni graf predstavlja celotno iskalno območje in če so razdalje med osebki v njem velike, to pomeni, da populacija pokriva velik del iskalnega območja. Seveda velja ravno obratno, če so razdalje med osebki majhne. Raznolikost je zelo pomembna v GA. [15]

3.12 Prednosti in pomanjkljivosti genetskih algoritmov

Genetski algoritmi imajo številne prednosti pred drugimi optimizacijskimi in iskalnimi tehnikami, vendar imajo tudi svoje slabosti in pomanjkljivosti. Zato si bomo pogledali, katere so dobre in slabe strani delovanja in uporabe GA.

3.12.1 Prednosti genetskih algoritmov

Začeli bomo s številnimi dobrimi lastnostmi GA, ki pomagajo algoritmu pri iskanju rezultatov.

Prva in zagotovo najpomembnejša lastnost je možnost paralelnega izvajanja, ki jo imajo GA že sami po sebi. Ta je posledica večjega števila potomcev v GA, kar pomeni iskanje rešitve v širokem spektru nabora rezultatov, medtem ko so drugi algoritmi večinoma zaporedni in posledično lahko iščejo rešitev le v določeni smeri. Če se iskana smer izkaže kot napačna, se vse to delo zavrže in ponovno začne iskanje od začetka. Drugi način paralelnega izvajanja je t. i. Schema izrek (angl. *Schema theorem*) ali osnovni izrek genetskih algoritmov (angl. *fundamental theorem of genetic algorithms*), ki je tudi splošno sprejet kot osnova za obrazložitev moči oziroma učinkovitosti GA. Izrek opisuje sposobnost

genetskih algoritmov, da na podlagi ocenjevanja fitnesa omejenega nabora vseh možnih kromosomov pridobi vedno bolj natančno povprečno fitnes vrednost za vsak posamezni gen. Posledično, kljub temu da GA eksplicitno ocenjujejo manjši vzorec vseh osebkov, posredno preverjajo veliko večjo množico. Primer podobnega delovanja je anketiranje, kjer se intervjuja zgolj vzorec neke populacije ljudi in kjer se rezultati lahko zatem na podlagi tega vzorca zanesljivo posplošujejo na celotno populacijo. Takšno delovanje omogoča GA, da zelo hitro najdejo območje rešitev, kjer so fitnes vrednosti zelo dobre in nato v tem izboru poišče še celostno najboljšo rešitev. [12]

Posledica istočasnega vzporednega posrednega preverjanja več rezultatov pomeni, da so GA zelo primerni za reševanje problemov, kjer je nabor možnih končnih rešitev posebej velik in je zato izčrpno preverjanje časovno nemogoče. Večina takšnih problemov spada med nelinearne probleme. V linearnih problemih so vse komponente neodvisne med seboj in posledično izboljšava na katerikoli izmed njih pomeni izboljšavo celotnega sistema. Večina realnih problemov na žalost ne deluje na takšen način. V nelinearnih problemih ima lahko sprememba ene komponente drastičen vpliv na celoten sistem ali pa lahko spremembe določene kombinacije komponent, ki vsaka zase škodljivo vplivajo na končni rezultat, omogoči veliko izboljšavo. GA so zmožni preverjanja resnično ogromnega števila možnosti in najti optimalno ali vsaj zelo dobro rešitev v relativno kratkem času. [12]

Naslednja dobra lastnost GA je, da delujejo učinkovito tudi na problemih, kjer je funkcija uspešnosti zapletena – npr. fitnes funkcija je nezvezna, vsebuje šum, se spreminja s časom, ali ima številne lokalne optimume. Kot je bilo omenjeno v prejšnji točki, ima večina realnih problemov zelo veliko število možnih rešitev, ki jih je v celoti nemogoče preveriti v doglednem času. Za GA je zato zelo pomembno, da se pri iskanju rešitve izogne lokalnim optimumom. Lokalni optimumi so rešitve, ki so najboljše med podobnimi rešitvami, vendar niso globalno najboljše – obstajajo drugačne rešitve iz drugega spektra množice možnih rešitev, ki so boljše. Številni iskalni algoritmi se ujamejo v past lokalnih optimumov, ker v neposredni bližini ne najdejo boljše rešitve in posledično sklepajo, da je to absolutno najboljša rešitev. Nasprotno so vse oblike evolucijskih algoritmov zelo uspešne pri izogibanju takšnim pastem in odkrivanju globalnega optimuma. Sicer je potrebno omeniti, da je v realnih problemih tipično nemogoče vedeti, ali je neka rešitev resnično globalno najboljša, vendar so se GA izkazali kot zelo zanesljivi pri pridobivanju, če ne že najboljših vsaj zelo dobrih rešitev. K takšnemu delovanju prispevajo vse štiri pomembne komponente GA: paralelizem, selekcija, mutacija in križanje. V prvem koraku GA se generira tipično naključna in posledično raznolika populacija, ki zajema vrednosti čez celotno funkcijo

uspešnosti. Mutacije omogočajo iskanje v neposredni bližini osebkov v populaciji, medtem ko križanje služi pretakanju informacij med posameznimi osebki in omogoča generiranje potomcev, ki vsebujejo dobre lastnosti obeh staršev brez njihovih pomanjkljivosti. Težava pri iskanju globalnega optimuma v prostoru s številnimi lokalnimi optimumi je pogosto poimenovana dilema raziskovanja proti izkoriščanju (angl. *dilemma of exploration vs. exploitation*). Ta govori o dilemi, pred katero je postavljen algoritem ali človek potem, ko najde zadovoljivo dobro rešitev na problem in se mora odločiti, ali naj se osredotoči na izkoriščanje najboljše najdene rešitve, ali naj nadaljuje iskanje v upanju na boljše rešitve. Če se iskanje nadaljuje, lahko pričakujemo (vsaj kratkoročno) poslabšanje končne rešitve in njene učinkovitosti. V primeru, da se najdena rešitev obdrži in ostale izumrejo, nastane tveganje obstoja boljše rešitve, ki še ni bila odkrita. Ponovno so se GA izkazali kot zelo dobri pri uravnovešanju obeh možnosti in odkrivanju dobrih rešitev v zmernem času ter ob uporabi zmernega računskega dela. [12]

GA so zelo dobri v istočasni manipulaciji številnih parametrov. Kot smo že vzpostavili, so realni problemi kompleksni in tipično jih ni mogoče predstaviti z eno samo vrednostjo, ampak morajo biti izraženi v obliki več parametrov. Ti so pogosto še obratno sorazmerni, tako da je cena za izboljšavo določenega parametra poslabšanje drugega. GA so ponovno zelo dobri pri reševanju takšnih problemov. Možnost vzporednega izvajanja omogoča algoritmu, da proizvede več enako dobrih rešitev za isti problem. Nekatere od teh rešitev optimizirajo en parameter, druge optimizirajo drug parameter in človeški nadzornik lahko zatem izbere, katero izmed rešitev želi uporabiti. [12]

Zadnja pomembna prednost, ki jo posedujejo GA, je na prvi pogled videti kot pomanjkljivost. Genetski algoritmi ne vedo nič o problemu, ki ga rešujejo. Namesto uporabe predznanja o danem problemu za usmerjanje algoritma k izboljševanju rešitev, GA vnašajo naključne spremembe v potencialne rešitve. Koristnost sprememb zatem preverjajo s funkcijo uspešnosti. To pomeni, da algoritem ne začne z nobenimi predpostavkami, o končnih rešitvah in ker so vse odločitve pri iskanju osnovane na naključnosti, so teoretično odprte vse poti do rezultata. Algoritmi, ki uporabljajo predhodno znanje, tipično izključijo številne možnosti in posledično lahko zgrešijo nove rešitve, ki se nahajajo v izključenem območju. Dodatno takšni algoritmi ne bodo delovali v primeru, ko potrebne predhodne informacije niso dosegljive, medtem ko GA ne bodo prizadeti. Posledice takšnega delovanja so, da mora GA preveriti večje število potencialnih rešitev, ker jih nič ne izloči na podlagi predznanja. Vendar je ta pomanjkljivost zaradi paralelnega izvajanja in zmožnosti iskanja v

zelo velikih množicah potencialnih rešitev (prvi dve opisani prednosti GA), v primerjavi z neokrnjenim iskalnim spektrom, ki ga GA pridobijo v zameno, zanemarljiva. [12]

3.12.2 Pomanjkljivosti genetskih algoritmov

Genetski algoritmi so se izkazali kot učinkovita in zanesljiva tehnika reševanja problemov, vendar niso brez svojih omejitev, za katere pa bomo pokazali, da so premostljive. Opisali bomo tudi, zakaj te pomanjkljivosti ne vplivajo na veljavnost oziroma učinkovitost naravnega izbora.

Prvi in najpomembnejši pomislek pri ustvarjanju GA je definicija problema. Za območje možnih rešitev je pomembno, da se določi na način, ki omogoča naključne spremembe brez posledic, kot so pogoste napake ali rezultati brez pomena. To lahko dosežemo na dva načina. Prvi in najpogosteje uporabljen, je reprezentacija kromosoma kot tabele vrednosti, ki so lahko predstavljene kot binarne vrednosti in cela ali realna števila. Vsaka takšna vrednost oziroma polje v tabeli predstavlja en gen v kromosomu. Mutacija, ki vnaša naključne spremembe, lahko sedaj spreminja te vrednosti, obrača bite ali prišteva oziroma odšteva naključne vrednosti. Delovanje algoritma se ne spremeni, zagotoviti je potrebno le, da so rezultati teh sprememb realne in možne vrednosti. Drugi način zagotavljanja robustnosti mutacije je t. i. genetsko programiranje (angl. *genetic programming*), kjer se spremeni tudi programska koda samega GA. V tem primeru so kromosomi predstavljeni kot izvedljiva drevesa kode, mutacija pa se izvede s spreminjanjem ali menjavo poddreves. Obe metodi zagotavljata robustnost sistema ob operacijah mutacije, čeprav lahko nosita s seboj dodatne težave. Ta problem v naravi sploh ne obstaja, saj je genetski kod že sam po sebi robusten in z zelo redkimi izjemami je vsaka DNK sekvenca veljavna. [12]

Fitnes funkcija mora biti napisana tako, da so višje fitnes vrednosti dosegljive in da večje vrednosti predstavljajo boljše rešitve za dan problem. Če je fitnes funkcija slabo izbrana ali nepravilno opredeljena, GA morda ne bo mogel najti rešitve ali pa bo rešitev iskal za povsem drug problem. Ta pomanjkljivost v naravnem izboru ne obstaja, ker je fitnes funkcija enaka za vse organizme – preživetje in reprodukcija. [12]

Poleg uporabe primerne fitnes funkcije je pomembno, da so tudi ostali GA parametri pravilno izbrani. Če je velikost populacije premajhna, je možno, da GA ne bo imel priložnosti raziskati dovolj velikega deleža množice možnih rešitev in posledično ne bo mogel vedno

najti dobre rešitve. V primeru, da je stopnja mutacij previsoka, oziroma je princip delovanja selekcije slabo izbran, je posledica prehitro spreminjanje populacije in nezmožnost selekcije, da vzpostavi konvergenco osebkov v populaciji, kar vodi v katastrofalno delovanje. Naravni izbor se sooča s podobnim problemom. Vsaka od naštetih napak lahko tudi v naravi vodi do izumrtja vrste, vendar je evolucija poskrbela za mehanizme, ki zmanjšujejo verjetnost takšnih napak. Večina živih organizmov je razvila način preverjanja in odstranjevanja napak v procesu podvajanja DNK in s tem znižala stopnjo mutacij. Obratno nekateri bakterijski organizmi v času visokega okoljskega stresa vstopijo v posebno stanje, v katerem se število napak med podvajanjem DNK poviša z namenom odkritja nove mutacije, ki bo omogočila organizmu nadalen obstoj v spremenjenem okolju. [12]

Tip problema, s katerimi imajo GA težave, so problemi z zavajajočimi fitnes funkcijami. Takšne funkcije se z odkrivanjem vedno boljših rešitev postopoma oddaljujejo od globalno najboljše rešitve. Preprost primer je problem, kjer je iskalni prostor osemmestni binarni niz, fitnes osebka v populaciji pa je proporcionalen s številom znakov 1 v nizu (niz 00000001 ima slabši fitnes od 00000011, ki je ponovno slabši od niza 00000111, itd.). Fitnes funkcija ime le dve izjemi: osem zaporednih enic ima zelo nizek fitnes in osem zaporednih ničel ima najboljšo fitnes vrednost v celotni populaciji. V takšnem primeru je verjetnost, da GA (in številni drugi algoritmi) najde pravilno rešitev, nič boljša od naključnega iskanja. Za to pomanjkljivost ne obstaja rešitev kot takšna, vendar je odgovor nanjo v genetskih algoritmih enak kot v naravnem izboru – evolucija ni proces, ki mora vedno najti globalni optimum, ampak je namenjena iskanju zadovoljivih rezultatov glede na vložen čas in trud. V večini situacij bo zadostovalo, če GA najde samo zelo dober lokalni optimum, četudi iz te točke zatem ni mogoče doseči globalno najboljše rešitve. Dodatno velja omeniti, da so realni problemi redko, če sploh kdaj, tako zavajajoči, kot je bil podan primer. Tipično lahko iz izboljševanja rezultatov razberemo vsaj nekaj informacij o lokaciji globalnega optimuma. [12]

Naslednja dobro poznana težava je prehitra konvergenca populacije. Če se v zgodnjih fazah izvajanja GA v populaciji pojavi osebek, ki ima izrazito boljšo fitnes vrednost od ostalih, se lahko tako številčno razmnoži, da se posledično začne prezgodnje zmanjševanje raznolikosti populacije. To lahko vodi v konvergenco populacije na lokalni optimum, ki mu je bil zelo blizu tisti začetni osebek, namesto dovolj podrobnega iskanja za odkritje globalnega optimuma. Ta problem GA je še posebej izrazit v primerih majhnih populacij, kjer je verjetnost pojava dominantnega osebka toliko večja. Najpogostejše metode za

omejevanje tega problema se vrtijo okrog upravljanja delovanja selekcije, tako da osebki populacije, ki imajo posebej visoko fitnes vrednost, nimajo prevelikega vpliva na naslednje generacije. Najpogostejši primeri tega so skaliranje z rangiranjem in turnirska selekcija, manj pogosti pa sta skaliranje s sigmoidno funkcijo in Boltzmannova selekcija, ki povečuje verjetnost selekcije skozi izvajanje. Prehitra konvergenca je težava tudi v naravi, le da se tam imenuje genetski zdrs (angl. *genetic drift*) in je veliko manj verjetna, ker je večina izboljšav v naravi majhnih in posledično osebkom ne daje velike prednosti pri reprodukciji. [12]

Strokovnjaki na področju priporočajo, da se GA ne uporablja v primerih, ko so problemi analitično rešljivi. GA lahko ravno tako najdejo rešitve na takšne probleme, vendar tradicionalne analitične metode vzamejo veliko manj časa in so manj zahtevne za izvedbo ter tipično matematično zagotavljajo odkritje globalno najboljše rešitve. Ker naravni izbor nima matematično najboljše rešitve, se ta problem v naravi ne pojavi. [12]

4 TESTNI PROBLEMI

Za namene te naloge bomo v nadaljevanju implementirali GA in z njimi testirali postavljene hipoteze. Za širši zajem podatkov o spremembah, ki jih bomo vnesli v algoritme, smo se odločili za uporabo GA na treh različnih problemih. To poglavje je namenjeno kratki predstavitvi teh problemov.

4.1 Weasel problem

Izrek o neskončni opici (angl*. infinite monkey theorem*) je star miselni preizkus. V njem opica naključno tipka na tipkalnem stroju. Ob zagotovitvi zadostnega časa bo opica sčasoma napisala zbrana dela Williama Shakespeara. To idejo je na področju evolucijskih algoritmov popularizira l Richard Dawkins leta 1986 v knjigi z naslovom *The Blind Watchmaker* [19]. Slovenska abeceda vsebuje 25 različnih črk. Ko mednje vključimo še znak za presledek, je verjetnost, da bi opica naključno uganila prvo črko v nizu $1/26$. Ta verjetnost se za dve zaporedni pravilno postavljeni črki zniža na $(1/26)^2$. Opazimo, da število kombinacij črk narašča eksponentno in tudi, če bi lahko opice preizkusile ogromno število kombinacij naenkrat, bi opravilo še vedno trajalo v nedogled. Dawkins je predlagal računalniški program, ki je nadomestil opice in je deloval po evolucijskem postopku, namesto da se je pri izgradnji besedila zanašal na popolno naključnost. V knjigi je dodatno iskano besedilo poenostavil na niz *"methinks it is like a weasel"*, ki je del igre Hamlet in od koder je problem tudi dobil ime – weasel (slo. *podlasica*). Namen nastalega programa je bil predvsem predstaviti moč mutacij, katerih delovanje je tudi nekoliko prikrojil, zato v tej nalogi ne bomo uporabljali enake implementacije, ampak bomo uporabili le isti problem, kjer bo moral GA iz začetnega naključnega niza znakov (začetna populacija) razviti niz znakov, ki bo enak vhodnemu iskanemu nizu. Natančnejše delovanje in parametri GA bodo predstavljeni v nadaljevanju. [20, 21]

4.2 Problem nahrbtnika

Problem nahrbtnika (angl*. knapsack problem*) si lahko predstavljamo kot problem, ki ga ima popotnik štopar, ko mora napolniti svoj nahrbtnik s čim več uporabnimi predmeti, vendar mora pri tem biti pozoren, da njegov nahrbtnik ne postane pretežak. Problem nahrbtnika je

bil tema številnih raziskav, saj je zelo primeren za uporabo v številnih industrijskih primerih. Matematično predstavimo problem kot: [22]

$$\text{čim večji} \quad \sum_{j=1}^{n} p_j x_j$$
$$\text{kjer velja} \quad \sum_{j=1}^{n} w_j x_j \leq c \tag{4.1}$$

Tu je:
n – število vseh predmetov,
j – indeks premeta, ki lahko zavzame vrednosti med 1 in n,
p_j – koristnost j-tega premeta,
x_j – pove ali nahrbtnik vsebuje j-ti predmet ali ne in lahko zavzame vrednosti 1 ali 0,
w_j – teža j-tega predmeta in
c – kapaciteta nahrbtnika oziroma skupna teža, ki jo lahko vstavimo v nahrbtnik.

Z drugimi besedami je cilj v nahrbtnik zbrati predmete s skupno čim večjo uporabnostjo, ne da bi pri tem presegli omejitev teže, ki je dovoljena za nahrbtnik. Primer istega problema, vendar v bolj realnih in resnih okoliščinah bi bilo vlaganje c količine denarja v n različnih investicij. V tem primeru je p_j dobiček, ki ga pričakujemo od investicije j, w_j pa kapital, ki bi ga bilo potrebno v to investicijo vložiti. Optimalna rešitev bi v tem primeru pokazala kombinacijo najboljših investicij. S takšnim razlogom bi marsikdo želel rešiti ta problem, vendar rešitev ni enostavna. Če bi želeli preveriti vse možne kombinacije bitov v danem vektorju, bi to trajalo zelo dolgo, saj je vseh možnih kombinacij 2^n. [22]

Problem nahrbtnika, kot smo ga predstavili, se podrobneje imenuje 0-1 problem nahrbtnika (angl*. 0-1 knapsack problem*), ker lahko en predmet izberemo samo enkrat. Takšna oblika problema bo tudi uporabljena v tej nalogi. Druge oblike so še: omejen problem nahrbtnika (angl*. bounded knapsack problem*), kjer je ponovna izbira omejena glede na tip predmeta, več-izbirni problem nahrbtnika (angl*. multiple-choice knapsack problem*), kjer je potrebno predmete izbirati iz različnih razredov in večkratni problem nahrbtnika (angl*. multiple knapsack problem*), kjer je potrebno naenkrat napolniti več nahrbtnikov. [23]

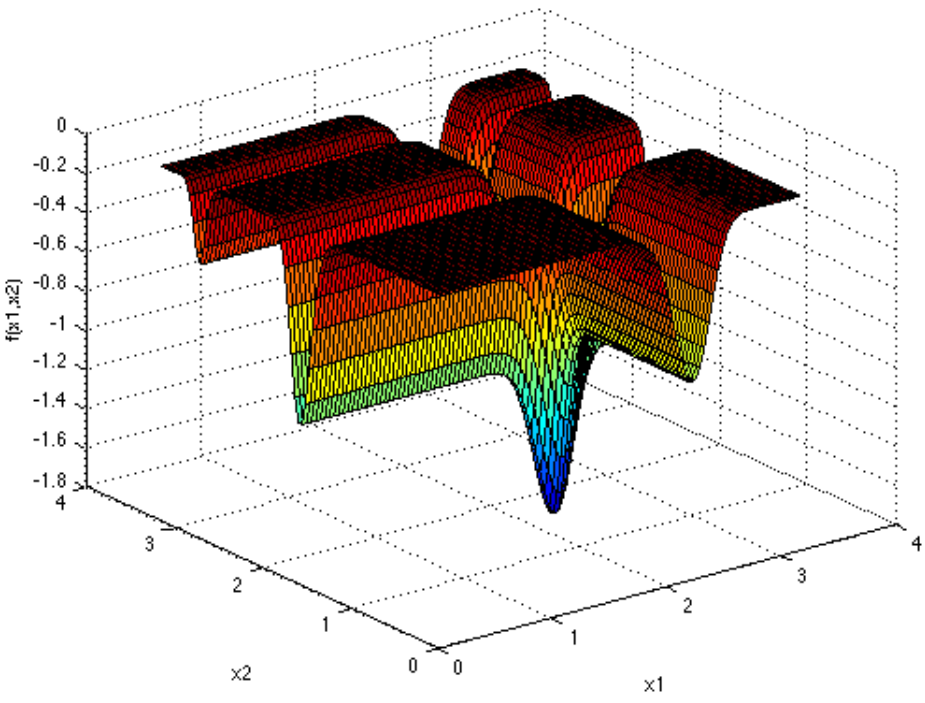
4.3 Funkcija Michalewicz

Funkcija Michealewicz je multimodalna testna funkcija (ima $n!$ lokalnih optimumov). Funkcija je primerna oziroma je dovolj zahtevna za preizkušanje delovanja GA, ker točke zunaj dolin (Slika 4.1) dajo zelo malo informacij o lokaciji globalnega optimuma in ker ima funkcija več lokalnih minimumov. Cilj GA je odkriti oziroma se tem bolj približati globalnemu minimumu funkcije. Definicija funkcije je zapisan v (4.2). [24]

$$f(x) = -\sum_{i=1}^{n} \sin(x_i)\left(\sin(\frac{ix_i^2}{\pi})\right)^{2m} \tag{4.2}$$

Tu je:
$f(x)$ – vrednost funkcije Michalewicz,
n – število dimenzij,
i – dana dimenzija, z vrednostmi $[1, \dots, n]$ in
m – definira strmino dolin v funkciji.

Spodaj (Slika 4.1) je predstavljen še grafični prikaz funkcije Michalewicz, kjer je $n = 2$.



Slika 4.1: Funkcija Michalewicz, kjer je $n = 2$. [25]

Parameter m definira strmine dolin, ki so vidne na sliki, vendar ne vpliva na minimum. Ob dovolj veliki vrednosti parametra m, je iskanje po funkciji podobno iskanju igle v kopici sena – zelo težko je najti območje, v katerem se nahaja optimum in posledično je iskanje zahtevnejše. Območje iskanja je običajno omejeno na $0 \leq x_i \leq \pi$. Dodatno je tipično $m = 10$. Globalni optimum za $n = 2$ se nahaja v $x_1 = 2{,}20$ in $x_2 = 1{,}57$ ter je enak $-1{,}8013$. Pri $n = 10$ je globalni minimum enak $-9{,}66015$ (to vrednost bomo iskali z GA). [24, 25]

5 IZBIRA GENOV NA OSNOVI STAROSTI

Namen te naloge je raziskati vpliv, ki ga ima izbira genov na osnovi starosti na delovanje in rezultate GA. Pri tem govorimo o prednosti, ki jih imajo različno stari geni pri izbiri v operatorjih križanja in mutacije. Obstaja veliko različnih načinov, na katere bi lahko uvedli starostno odvisnost. Mi smo izbrali nekaj tistih, za katere smo menili, da bi bili zanimivi in bi pokrili različne starostne skupine, tako da bi vedno imela različna skupina prednost pri izbiri. To poglavje je namenjeno konceptualni predstavitvi teh načinov delovanja. Specifična implementacija se seveda razlikuje od problema do problema, zaradi različnih predstavitev genoma. Kako bomo takšno delovanje vpeljali v algoritme, ki rešujejo različne probleme, bo predstavljeno v poglavju 6 (Izdelava aplikacije in implementacija genetskih algoritmov), kjer je opisana sama implementacija takšnega delovanja in kjer so vidne razlike med različnimi predstavitvami genomov.

V raziskavi želimo preveriti, kakšne bi bile posledice, če bi enakomerno razmerje med verjetnostmi izbora genov za križanje in mutacijo nekoliko spremenili glede na starost udeleženih genov. Pri tem se starost gena meri kot število generacij brez mutacije oziroma število prednikov, ki so prispevali svoj dedni material za izgradnjo trenutnega gena. Ob začetku izvajanja je starost vseh genov enaka 0 in se z vsako generacijo poveča za 1. Edina operacija, ki spremeni starost gena, je mutacija. Starost gena, ki je mutiral, se ponastavi na 0. Odvisnost izbire genov od njihove starosti smo vpeljali za operatorja križanja in mutacije, vendar na različne načine, zato ju bomo obravnavali ločeno.

5.1 Verjetnost izbire genov na osnovi starosti v operacijah križanja

Izbiro genov s starostno odvisnostjo bomo v operacijah križanja vpeljali na štiri različne načine: prednost mlajših genov, prednost starejših genov, prednost povprečno starih genov in istočasna prednost najmlajših ter najstarejših genov. Čeprav so to štiri različne situacije, kjer imajo geni v različnih starostnih skupinah prednost pri izbiri, opazimo, da lahko to dosežemo z dvema različnima načinoma delovanja. Funkcija, ki omogoča večjo možnost izbire mlajših genov, je zgolj obratna funkcija tiste, ki daje prednost starejšim genom in isto velja za drugi par postopkov preferenčne izbire. Kot osnovo bomo uporabili uniformno križanje, ki ga bomo nadgradili s spremenjeno verjetnostjo izbire genov. V tradicionalnem

delovanju takšnega GA je pri izbiri med dvema genoma v operaciji križanja razmerje verjetnosti enako – oba gena imata 50 % možnost izbora.

Odstotek starostne razlike je vhodni parameter v GA s starostno odvisnim delovanjem. Njegova naloga je omejevati prednost, ki jo lahko en gen pridobi pred drugim v operaciji križanja. V osnovi parameter določi obseg vpliva, ki ga bo imelo starostno odvisno križanje. Odstotek starostne razlike je podan kot decimalna vrednost, ki predstavlja maksimalno vrednost, za katero lahko gen izboljša svoje možnosti pri izbiri. Vzemimo za primer situacijo, kjer je odstotek starostne razlike enak 0,1. To pomeni, da lahko gen svojo verjetnost za izbor poveča za največ 10 % – namesto 50-50 % se razmerje spremeni na 60-40 %. To je seveda največja možna razlika. Možne so tudi vse vmesne vrednosti in se določijo glede na razmerje starosti obeh genov. V primeru križanja smo se odločili, da bomo takšno delovanje dosegli s pomočjo Gaussove funkcije, ki je prikazana na sliki v nadaljevanju poglavja (Slika 5.1). Za naše potrebe bo zadostovala eno dimenzijska Gaussova funkcija, ki je identična funkciji gostote verjetnosti za normalno (Gaussovo) porazdelitev. Definiramo jo kot: [26]

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} \tag{5.1}$$

Tu je:

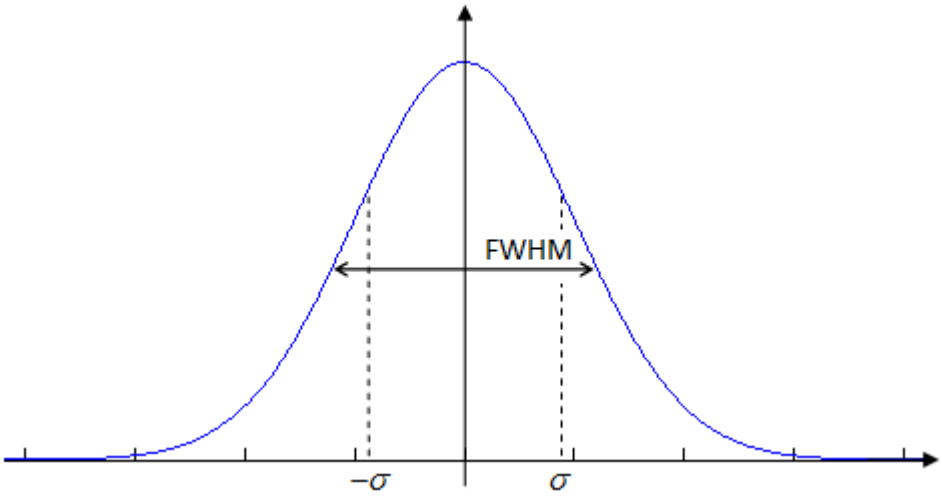
$f(x)$ – eno dimenzijska Gaussova funkcija,

σ – standardni odklon,

μ – pričakovana vrednost (tudi matematično upanje) in

σ^2 – varianca (tudi verjetnost distribucije).

Spremembe pričakovane vrednosti bi se na grafični predstavitvi poznale kot vodoravni zamik krivulje, medtem ko standardni odklon določa širino oziroma strmino krivulje. Na podlagi (5.1) lahko izračunamo, da standardni odklon predstavlja polovico širine vrha, pri približno 60 % višine krivulje. Alternativa temu je polna širina na polovični višini (angl. *full width at half maximum – FWHM*). FWHM je približno $2{,}3548\sigma$ in se nahaja točno na polovici višine krivulje. Standardni odklon in polna širina na polovični višini sta prikazana na naslednji sliki (Slika 5.1). [27]



Slika 5.1: Normalna porazdelitev, standardni odklon in polna širina na polovični višini. [28]

Sedaj je potrebno Gaussovo funkcijo prirediti tako, da jo bomo lahko uporabili v namene spreminjanja verjetnosti izbire genov. Ponovno lahko to dosežemo na različne načine in brez testiranja je zelo težko predhodno predvideti, kateri je najboljši. Preverjanje vseh pristopov k rešitvi problema bi bilo zelo časovno potratno in v osnovi ni namen te naloge, zato smo izbrali načine, ki zajamejo različne starostne skupine. Podobno velja za odločitev o uporabi Gaussove funkcije, kjer bi lahko ponovno izbrali katerokoli drugo funkcijo.

Za začetek bi želeli, da je povprečna pričakovana starost genov točno na polovici višine krivulje. Abscísna os (x-os) v tem primeru prikazuje starost, ordinatna os (y-os) pa podatek o vplivu starosti gena na izbiro. Pri tem je (zaenkrat) potrebno odmisliti levi del krivulje, ki vsebuje negativne vrednosti, ker je starost lahko samo 0 ali pozitivna cela vrednost. Takšna oblika krivulje bi pomenila, da bi se v povprečju polovica osebkov iz populacije nahajala desno od sredine višine krivulje, druga polovica pa levo. Kot smo že omenili, se starost genov vsako generacijo poveča za 1. Starost gena se poleg tega spremeni samo še v primeru mutacije. Verjetnost mutacije gena je relativno enostavna za izračun – stopnja mutacij (verjetnost, da bo genom izbran za mutacijo) vlomljeno s številom genov v genomu. Povprečna pričakovana starost je potem približno ena polovica te vrednosti, ker se bo starost mutiranih genov ponastavila na vrednost 0. Ta ocena pričakovane povprečne starosti genov bo seveda povsem napačna na začetku izvajanja, ko je povprečna starost genov enaka 0, vendar se bo skozi izvajanje ta vedno bolj približevala napovedani vrednosti. Posledica odstopanja začetne povprečne starosti načeloma ni problem. Za pričakovano povprečno starost sploh ni potrebno, da je natančna. Namen postavitve te vrednosti na mesto sredine višine krivulje je zgolj zagotovitev konsistentnosti med različnimi

problemi, kjer so pričakovane starosti genov lahko zelo različne in še pomembneje - vzpostavitev primerne strmine krivulje, tako da pri križanju dveh genov, kjer je njihova razlika v starosti glede na povprečno pričakovano starost relativno majhna, ne pride do velike prednosti enega izmed genov. V naši funkciji je standardni odklon edini parameter, ki vpliva na obliko krivulje, zato je to edina vrednost, ki jo moramo pravilno nastaviti, da dosežemo želeno obliko krivulje. Standardni odklon, kjer je sredina višine krivulje enaka pričakovani povprečni starosti, izrazimo kot:

$$\sigma = \frac{p}{2{,}3548/2} \tag{5.2}$$

Tu je:
σ – standardni odklon in
p - povprečna pričakovana starost.

Enačba nastane iz razmerja med FWHM in σ. FWHM je približno 2,3548-krat večji od standardnega odklona. Abscisna vrednost, ki na krivulji označuje polovično višino, se nahaja za polovico FWHM vrednosti desno (in levo, vendar nas zanima samo ena točka) od pričakovane vrednosti funkcije. Enačbo (5.2) lahko izrazimo, ker vemo višino krivulje in σ ter ker poznamo razmerje med polovico višine krivulje in FWHM ter razmerje med FWHM in σ. [27]

Sedaj, ko imamo Gaussovo funkcijo, ki je primerna za našo uporabo, lahko pridobimo vpliv na izbiro (y-vrednost na grafu) za vsako starost (x-vrednost na grafu). S spreminjanjem standardnega odklona, s čimer smo pridobili ustrezno strmino krivulje, se je istočasno spremenila tudi višina krivulje (največji možen vpliv na izbiro). Zato je treba rezultate, ki jih pridobimo iz Gaussove funkcije, primerno normirati. Prirediti jih moramo tako, da bodo vrednosti spadale v območje med 0 in odstotek starostne razlike (parameter, ki omejuje največjo spremembo verjetnosti izbire). To dosežemo z množenjem rezultata Gaussove funkcije s količnikom odstotka starostne razlike in največjo vrednostjo Gaussove funkcije. Čeprav je implementacija predstavljena v nadaljevanju, je na naslednji strani za lažje razumevanje še odrezek kode (Koda 5.1), ki izvaja opisano delovanje.

```
moznostMutacijeGena = stopnjaMutacije / stGenovVGenomu;
povprecnaPricakovanaStarost = (1 / moznostMutacijeGena) / 2;
sigma = povprecnaPricakovanaStarost / 2.3548 / 2;
normaliziranjeGauss = odstotekStarostneRazlike / Gaussian.phi(0, 0, sigma);
```

Koda 5.1: Izdelava parametrov Gaussove funkcije.

5.1.1 Privilegiranje mlajših in starejših genov

Privilegiranje mlajših in starejših genov sta dva zelo podobna načina delovanja, ki dajeta mlajšim oziroma starejšim genom prednost pri izbiri v operaciji križanja. V postopku spremembe verjetnosti izbora dveh sodelujočih genov se starost prvega gena najprej odšteje od drugega. Za nastalo razliko se izračuna Gaussova funkcija, rezultat katere se normira, kot je bilo opisano. Ker je Gaussova funkcija simetrična glede na matematično upanje in je v našem primeru to enako 0, je rezultat funkcije enak za pozitivna in negativna števila. Nastalo vrednost je treba še odšteti od odstotka starostne razlike, da dobimo vrednost, za katero se bo povečala možnost izbora enega izmed genov. Zadnji korak je potreben, ker v funkcijo vstavljamo razliko starosti. Ko je ta razlika majhna, bo vrnjena vrednost Gaussove funkcije relativno velika (maksimum se nahaja v točki 0). Torej, če sta dva gena podobno stara, bi eden izmed njiju prejel veliko prednost pred drugim. To je ravno obratno delovanje od želenega. Namen je genu, ki je veliko starejši ali mlajši od drugega gena (in posledično je med njima velika razlika v starosti), omogočiti večjo možnost izbire. Zato to vrednost odštejemo od odstotka starostne razlike in spremenimo velike vrednosti v majhne in obratno. Sedaj je potrebno pogledati, kakšna je bila razlika med starostmi genov. V primeru, da je bila ta negativna, je prvi gen mlajši in ustrezno glede na to ali želimo takšnim genom dati prednost ali ne, prištejemo ali odštejemo izračunano vrednost od začetnih 50 %. Zatem ustrezno prilagodimo verjetnost drugega gena in izbor se lahko izvede. Delovanje je skrčeno na naslednjo enačbo:

$$v = 0{,}5 \pm o - \varphi(s_1 - s_2, 0, \sigma) * n \tag{5.3}$$

Tu je:

v – verjetnost izbora prvega gena,

o – odstotek starostne razlike,

s – starost gena,

σ – standardni odklon (spremenljivka *sigma* v Koda 5.1),

$\varphi(s_1 - s_2, 0, \sigma)$ – Gaussova funkcija pri iskani vrednosti $s_1 - s_2$, pričakovani vrednosti 0 in odklonu σ ter
n – konstanta za normalizacijo Gaussove funkcije (spremenljivka *normaliziranjeGauss* v Koda 5.1).

5.1.2 Privilegiranje genov povprečnih in robnih starosti

Privilegiranje genov povprečnih in robnih starosti povečuje možnost izbire genov, ki so povprečno stari oziroma genov, ki so mladi ali stari. Načina delovanja sta si ponovno nasprotna. Starostno skupino genov, ki jo en način delovanja nagrajuje z boljšo verjetnostjo izbora, drugi način kaznuje in obratno. Izračun verjetnosti izbire, kjer imajo prednost povprečno stari geni, začnemo z vrednostjo 0,50 (verjetnost izbora v uniformnem križanju). Tej vrednosti prištejemo normaliziran rezultat Gaussove funkcije za starost prvega gena. Pri tem je potrebno biti pozoren, da matematično upanje, ki je vhodni parameter za funkcijo, nastavimo na prej predstavljeno povprečno pričakovano starost. To zamakne krivuljo v desno, tako da se vrh nahaja na mestu, kjer pričakujemo povprečno starost genov. Ponovno je ideja, da bi se polovica osebkov nahajala na vsako stran te vrednosti. Od tako pridobljene vrednosti odštejemo normirano vrednost Gaussove funkcije za starosti drugega gena. Tudi tu spremenimo matematično upanje funkcije na vrednost povprečne pričakovane starosti. Rezultat je verjetnost izbora prvega gena. Možnost izbora drugega preprosto izračunamo tako, da od 1 odštejemo verjetnost prvega. To delovanje opisano v (5.4).

$$v = 0{,}5 + (\varphi(s_1, p, \sigma) * n) - (\varphi(s_2, p, \sigma) * n) \tag{5.4}$$

Tu je:
v – verjetnost izbora prvega gena,
s_1 – starost prvega gena,
s_2 – starost drugega gena,
p – povprečna pričakovana starost (spremenljivka *povprecnaPricakovanaStarost* v Koda 5.1),
σ – standardni odklon (spremenljivka *sigma* v Koda 5.1)
$\varphi(s_i, p, \sigma)$ – Gaussova funkcija pri iskani vrednosti s_i, pričakovani vrednosti 0 in odklonu σ ter
n – konstanta za normalizacijo Gaussove funkcije (spremenljivka *normaliziranjeGauss* v Koda 5.1).

Verjetnost izbora gena v privilegiranju robnih starosti se izračuna z enakim postopkom, le da vedno preden prištejemo ali odštejemo normaliziran rezultat Gaussove funkcije prejšnji vrednosti, tega odštejemo od odstotka starostne razlike in zatem uporabimo to vrednost. S tem tako kot pri privilegiranju mlajših in starejših genov obrnemo majhne in velike vrednosti.

Privilegiranje genov povprečnih in robnih starosti nosi s seboj tveganje. Kot smo že govorili, povprečna pričakovana starost ni realen odraz povprečne starosti genov, ker je začetna povprečna starost genov v algoritmu vedno enaka 0. In se zato ta šele s časom približuje predvideni vrednosti. To pomeni, da je lahko delovanje pri privilegirani povprečni starosti postane enako tistemu pri privilegiranju starejših genov. Isto velja za privilegiranje genov z robnimi starostmi in mlajših genov. To še posebej velja v začetnih fazah izvajanja, ko geni še niso imeli časa pridobiti ustrezne starosti.

5.2 Verjetnost izbire genov na osnovi starosti v operacijah mutacije

Operator mutacije je v osnovi preprostejši operator od križanja, zato smo tudi starostno odvisnost pri izbiri genov za mutacijo zgradili na preprostejšem delovanju. Spreminjanje verjetnosti pri izbiri genov za mutacijo smo vpeljali na dva načina - enkrat imajo prednost starejši geni, drugič mlajši. Osnoven algoritem uporablja uniformno mutacijo, kjer je verjetnost izbire enaka za vse gene. To smo spremenili oziroma nadgradili v mutacijo, katere proces izbire genov za mutacijo je enak izbiri osebkov v selekciji z ruleto. Začnimo z mutacijo, kjer imajo starejši geni prednost. Ko je osebek za mutacijo izbran, se seštejejo starosti vseh genov. Zatem se izbere naključna vrednost med 0 in seštevkom starosti. Ponovno se začne seštevanje starosti vseh genov. Prvi gen, katerega prišteta starost preseže naključno izbrano število, bo mutiral. Starejši geni imajo prednost pri izbiri, ker njihova večja starost predstavlja večji delež seštevka vseh starosti. Prednost mlajših genov dosežemo na enak način, le da pri seštevanju uporabimo obratno vrednost starosti gena namesto same vrednosti. Obratna vrednost celega števila (starosti) je 1 ulomljeno s samim številom. Tu je potrebno omeniti, da je starost vseh genov do časa, ko pride do mutacije vsaj 1 (ni nikoli 0), tako da obratna vrednost vedno obstaja. Verjetnost izbire ostane enako linearno razmerje, kot je veljalo pri privilegiranju starejših genov. Kot primer vzemimo gena s starostma 3 in 9. Drugi gen je trikrat starejši, torej ima pri privilegiranju starejših genov trikrat večjo verjetnost, da bo izbran za mutacijo. Pri prednosti mlajših genov je gen z vrednostjo $\frac{1}{3}$ natančno trikrat mlajši in ima trikrat večjo možnost, da bo izbran kot gen z

vrednostjo $\frac{1}{9}$. Primeri implementacije tega delovanja za vse tri algoritme se nahajajo v naslednjem poglavju.

6 IZDELAVA APLIKACIJE IN IMPLEMENTACIJA GENETSKIH ALGORITMOV

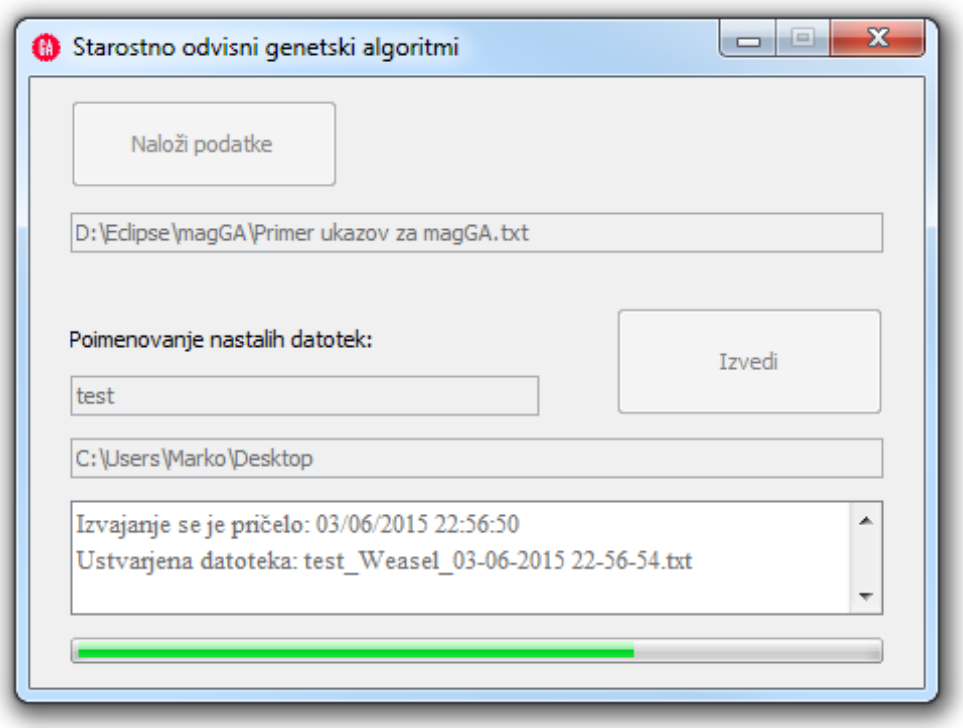
To poglavje je namenjeno predstavitvi aplikacije in genetskih algoritmov, katere bomo implementirali za namene preverjanja hipotez predpostavljenih na začetku te naloge. Opisan bo način izgradnje in delovanje programa in algoritmov. Poglavje je v osnovi razdeljeno na pet večjih delov. Prvi del je namenjen predvsem aplikaciji in vsebuje dele kode oziroma delovanje, ki se izvede pred katerimkoli algoritmom. Drugi del opisuje delovanje, ki je skupno vsem trem algoritmom. Zadnji trije deli so namenjeni predstavitvi implementacij specifičnih rešitev za vsak problem posebej.

Cilj naloge je preučiti, kakšen vpliv ima upoštevanje starosti genov pri delovanju križanja in mutiranja na uspešnost in učinkovitost GA. Opisana aplikacija je namensko orodje, ki je nastalo, zato da lahko pridobimo ustrezne podatke za raziskavo. Iz tega razloga je aplikacija precej osnovna in nima naprednega grafičnega vmesnika. Pri implementaciji algoritmov smo upoštevali dve navodili. Prvo je zmožnost preklapljanja načina delovanja algoritmov iz tradicionalnega v izvajanje z upoštevanjem starosti genov. Drugo navodilo, ki smo ga želeli izpolniti v programu, je berljivost kode. Probleme je potrebno rešiti s konvencionalnimi GA in s starostno odvisno različico, zato da bomo v nadaljevanju lahko primerjali rezultate. Berljivost kode in dejstvo, da oba načina izvajanja opravlja ista koda, sta pomembni za transparentnost delovanja. Iz transparentnosti in razumevanja kode postane razvidno, da lahko različni rezultati izhajajo le iz drugačnih operacij križanja in mutacije. Iz tega razloga in ker za naše namene hitrost delovanja oziroma izraba virov ni pomembna, algoritmi niso optimizirani. Aplikacija je napisana v programskem jeziku Java in s pomočjo integriranega razvojnega okolja Eclipse. Celotna aplikacija je priložena na zgoščenki kot priloga A.

6.1 Aplikacija

Predstavitev izgradnje aplikacije bomo začeli z zaključenim produktom. Na naslednji sliki (Slika 6.1) je prikazan grafični vmesnik aplikacije. Iz slike je razvidno, da lahko uporabnik proži dve operaciji: nalaganje podatkov in začetek izvajanja. Nalaganje podatkov vstavi v aplikacijo podatke, ki so zahtevani za izvajanje. Ti so posredovani v tekstovni datoteki. Ko izberemo veljavno datoteko s podatki, se lokacija te prikaze v oknu pod gumbom za nalaganje podatkov. V primeru, da posredujemo neustrezno datoteko, se uporabniku izpiše

opozorilo o napaki. Točno kateri podatki morajo biti vsebovani v vhodni datoteki in kako morajo biti strukturirani, si bomo pogledali v nadaljevanju. Gumb za začetek izvajanja sproži proces izvajanja GA ob parametrih, ki so bili posredovani aplikaciji. Produkt operacije so nove tekstovne datoteke, ki vsebujejo rezultate izvedenih algoritmov. Ob kliku na gumb se uporabniku najprej odpre novo okno, kjer izbere lokacijo, na kateri bi želel, da se zapišejo datoteke z rezultati. Ker lahko naenkrat posredujemo aplikaciji navodila za izvajanje več različnih algoritmov z različnimi parametri, obstaja možnost dodajanja predpone pri poimenovanju nastalih datotek. Tako bodo vse datoteke, ki so nastale na podlagi ene datoteke s parametri imele enak začetek imena. S tem smo želeli zmanjšati zmedo, ki lahko nastane ob večjem številu generiranih datotek. Tako lokacija shranjevanja rezultatov kot predpona za poimenovanje nastalih datotek sta vidni v poljih zraven gumba za sprožitev izvajanja. Nižje se nahaja večje polje, ki vsebuje informacije o poteku izvajanja. Tu se izpiše čas ob začetku in koncu izvajanja ter imena vseh izhodnih datotek, ki nastanejo v procesu. Kot vidimo na sliki, so celotna poimenovanja datotek sestavljena iz predpone, ki smo jo izbrali, imena algoritma in časa začetka izvajanja tega algoritma. Povsem na dnu okna aplikacije se nahaja še indikator napredka, ki označuje napredek v izvajanju. Na spodnji sliki so vse komponente onemogočene, ker se trenutno izvajajo posredovani ukazi, kot nakazuje tudi indikator napredka, ki ni še v celoti zelen.



Slika 6.1: Aplikacija za izvajanje testnih genetskih algoritmov.

6.1.1 Nalaganje podatkov

Za začetek si poglejmo strukturo vhodnih podatkov. Vhod v aplikacijo je tekstovna datoteka, v kateri so vsi potrebni parametri za izvajanje. Vsaka vrstica v datoteki predstavlja en ukaz, ki izvede en algoritem ob podanih parametrih. Datoteke so sestavljene po principu vrednosti, ločenih z vejicami (angl*. comma separated values – CSV*), kjer vejica loči dva parametra. V našem primeru se kot razmejevalnik med parametri uporablja podpičje, medtem ko je vejica uporabljena kot razmejevalnik znotraj enega parametra (npr. zaporedje vrednosti oziroma tabela). Pri zapisu decimalnih vrednosti je potrebno uporabiti decimalno piko. Vrstica, ki se začne z dvema poševnicama ali številskim znakom (lojtra) se smatra kot komentar in ne bo upoštevana kot izvedbena vrstica. Takšna vrstica lahko vsebuje poljubno vsebino. Različni algoritmi zahtevajo različne vhodne podatke, vendar je začetek v vseh primerih enak. V spodnji tabeli (Tabela 6.1) so predstavljeni vhodni podatki in njihovo zaporedje. Kjer je zaporedna številka parametra sestavljena, število za piko predstavlja algoritem, za katerega so parametri potrebni.

Tabela 6.1: Zaporedje in predstavitev vhodnih parametrov.

#	Ime	Opis	Tip podatka
1	Algoritem	Vsebuje podatek o tem, kateri algoritem se bo izvedel.	1 = D. Weasel 2 = p. nahrbtnika 3 = f. Michalewicz
2	Število ponovitev	Pove, kolikokrat bomo izvedli dani algoritem ob podanih parametrih. Ponovitve so potrebe za statistično vzorčenje.	Pozitivno celo število
3	Tip izvajanja	Izbere način križanja in mutacije, ki bodo uporabljene. Na primer tip izvajanja 0 je tradicionalno delovanje GA, kjer starost genov ne vpliva na križanje ali mutacijo in oznaka 1 pripada GA s spremenjeno mutacijo, pri kateri imajo starejši osebki večjo možnost izbire za mutacijo. Čeprav aplikacija omogoča kombiniranje različnih mutacij in križanj, rezultati teh ne bodo predstavljeni v tej nalogi.	0 = tradicionalno 1 = m. starih 2 = m. mladih 3 = k. starih 4 = k. mladih 5 = k. povprečnih 6 = k. ekstremov 7-14 so kombinacije prejšnjih.
4	Velikost populacije	Določi velikost populacije.	Celo pozitivno število

5	Število generacij	Določi število generacij, do katere se bo algoritem izvajal. Če se izvajanje pred tem še ni zaključilo, služi kot zaustavitveni pogoj.	Celo pozitivno število
6	Odstotek starostne razlike	Določi, za kakšno vrednost se lahko verjetnost izbire gena v operaciji križanja razlikuje od tradicionalnega delovanja. Ta parameter se upošteva le pri tipih izvajanja, kjer je uporabljeno križanje s starostno odvisno izbiro genov.	Decimalno število
7.1	Iskani niz	Niz v Dawkins Weasel problemu, ki ga bo GA želel poustvariti.	Niz poljubnih znakov
7.2	Kapaciteta nahrbtnika	Določi skupno težo predmetov, ki jih lahko vstavimo v nahrbtnik.	Celo pozitivno število
8.2	Teže predmetov	Tabela teže predmetov. Vrednosti, ki predstavljajo težo predmetov, nanizamo eno za drugo in jih ločimo z vejicami. Parameter lahko obdamo z oglatimi oklepaji za lažjo berljivost. Predmet na i-tem indeksu ima uporabnost istoležne vrednosti v tabeli uporabnosti predmetov.	Vsaka vrednost je celo pozitivno število. Tabeli teže in uporabnosti predmetov morata biti enako dolgi.
9.2	Uporabnosti predmetov	Tabela uporabnosti predmetov. Vrednosti za uporabnost predmetov nanizamo eno za drugo in jih ločimo z vejicami. Parameter lahko obdamo z oglatimi oklepaji za lažjo berljivost. Predmet na i-tem indeksu ima težo istoležne vrednosti v tabeli tež predmetov.	Vsaka vrednost je celo pozitivno število. Tabeli teže in uporabnosti predmetov morata biti enako dolgi.
7.3	n	Parameter predstavlja število dimenzij v funkciji Michalewicz (glej poglavje 4.3).	Pozitivno celo število

Ko uporabnik potrdi izbiro vhodne datoteke, se ta vrstico po vrstico prebere v aplikacijo. Za vsako vrstico, ki ni komentar, se ustvari instanca razreda *Pripravni*, v katerega se ukaz vstavi preko konstruktorja. Ta razred je odgovoren za pripravo podatkov, ki bodo uporabljeni med izvajanjem in za nadzor ter proženje samega izvajanja. Po sprejetju ukazne vrstice se ta najprej razčleni glede na podpičja. Iz nastale tabele parametrov se kot prvi preberejo generični parametri - parametri, ki so skupni vsem algoritmom. Zatem se razbere kateremu algoritmu je ukaz namenjen in na podlagi tega se pripravijo še algoritmom specifični podatki. Vsi razbrani podatki se do začetka izvajanja hranijo v tem objektu. Čeprav se pri pridobivanju podatkov preverja nekaj bolj pogostih napak, ki jih uporabnik lahko vnese, in pravilnost tipov podatkov, se ne preverjajo vse možne semantične napake. Zato je mogoče vnesti podatke, ki bodo uspešno prestali vnos v aplikacijo, vendar lahko še vedno povzročijo

težave med izvajanjem. Če se med pripravo podatkov sproži izjema, se uporabnika obvesti, v katerem zaporednem ukazu se nahaja napaka. V nadaljevanju se nahajajo odrezki kode, ki dosežejo opisano delovanje. Ti in vsi ostali odrezki v tej nalogi se lahko malenkostno razlikujejo od izvorne kode. Vse spremembe so vnesene z razlogom lažjega sledenja delovanju in ne vplivajo na delovanje ali zavajajo o njem. Odrezki prikazujejo tri metode. Prva je konstruktor razreda (Koda 6.1), ki sprejme ukazno vrstico in upravlja nadaljnje izvajanje. Druga (Koda 6.2) pripravi splošne podatke za izvajanje in tretja (Koda 6.3) je primer priprave specifičnih podatkov, v tem primeru za Dawkins Weasel algoritem.

```
public Pripravni (String ukaz) throws Exception{
	this.ukaz = ukaz;
	String[] parametri = ukaz.split(";");
	pripraviGeneralno(parametri);
	switch(algoritem = Integer.parseInt(parametri[0].trim())){
		case 1:
			pripraviDawkinsWeasel(parametri);
			break;
		case 2:
			pripraviNahrbtnik(parametri);
			break;
		case 3:
			pripraviMichalewicz(parametri);
			break;
		default:
			throw new Exception();
	}
}
```

Koda 6.1: Konstruktor razreda *Pripravni*.

```
private void pripraviGeneralno(String [] parametri) throws Exception{
	ponovitve = Integer.parseInt(parametri[1].trim());
	tipIzvajanja = Integer.parseInt(parametri[2].trim());
	if (tipIzvajanja < 0 || tipIzvajanja > 14){
		throw new Exception();
	}
	velikostPopulacije = Integer.parseInt(parametri[3].trim());
	steviloGeneracij = Integer.parseInt(parametri[4].trim());
	odstotekStarostneRazlike = Double.parseDouble(parametri[5].trim());
}
```

Koda 6.2: Metoda za pripravo vsem algoritmom skupnih vhodnih podatkov.

```
private void pripraviDawkinsWeasel(String [] parametri) throws Exception{
	vhodniNiz = parametri[6].trim();
	if(vhodniNiz.startsWith("\"") || vhodniNiz.startsWith("'")){
		vhodniNiz = vhodniNiz.substring(1);
	}
	if(vhodniNiz.endsWith("\"") || vhodniNiz.endsWith("'")){
		vhodniNiz = vhodniNiz.substring(0,vhodniNiz.length()-1);
	}
}
```

Koda 6.3: Priprava vhodnih podatkov za Dawkins Weasel algoritem.

6.1.2 Izvajanje genetskih algoritmov

Klik na gumb za začetek izvajanja brez vnosa ustrezne datoteke z vhodnimi podatki ustvari pojavno okno, ki o tem opomni uporabnika. V nasprotnem primeru pa se uporabniku najprej odpre okno za izbiro lokacije, na katero bi rad nastale datoteke shranil. Ko je ta izbrana, se mu odpre še okno za vpis predpone, ki bi jo rad uporabil pri poimenovanju nastalih datotek. Izvajanje se začne z ustvarjanjem nove izvajalne niti, kar omogoča odziven grafični vmesnik med izvajanjem algoritmov. Zatem se v objektih, ki smo jih ustvarili z vstavitvijo vhodnih podatkov, požene metoda *izvedi*. Ta metoda je prikazana na naslednji strani. Izvajanje teh metod poteka zaporedno. To pomeni, da se mora prejšnja metoda zaključiti, preden se proži metoda v naslednjem objektu. Če bi želeli delovanje optimizirati, je paralelizacija tega dela najpreprostejši in najučinkovitejši način. Metoda začne s sestavo imena za datoteko z rezultati in vzpostavitvijo objekta *writer*, ki skrbi za pisanje v to datoteko. S tem takoj v datoteko zapiše ukazno vrstico, ki jo bomo izvedli. Tako lahko za vse nastale datoteke, na podlagi prve vrstice v njej, točno vemo, za kateri algoritem in ob kakšnem delovanju so rezultati (preostala vsebina datoteke) nastali. Zatem se glede na opredeljen algoritem ustvari objekt za izvajanje samega GA, v katerega se vstavijo vsi potrebni podatki. Tega poženemo tolikokrat, kot je bilo opredeljeno v vhodnih podatkih. Isti algoritem je potrebno izvesti večkrat, ker se GA ob vsakem izvajanju obnaša drugače. Ob dovolj ponovitvah lahko učinkovitost delovanja ocenimo na podlagi povprečnih rezultatov. Vsaka takšna ponovitev predstavlja eno vrstico v datoteki z rezultati. Te ponovitve ponovno predstavljajo dobro mesto za optimizacijo. Predstavitev implementacije in opis delovanja GA za vse tri probleme sledi v nadaljevanju. Po zaključku vsakega posameznega zagona algoritma posodobimo indikator napredka. Ko se izvede zahtevano število ponovitev, se v informativno polje na grafičnem vmesniku izpiše obvestilo o dokončanju dane datoteke z rezultati. Program se

nadaljuje z naslednjim objektom *Pripravni,* ki vsebuje navodila za izvajanje naslednjega ukaza. Sledi odrezek metode (Koda 6.4), ki izvede opisano delovanje.

```
public void izvedi(String izhodnaLokacija, String predpona, Task task){
	PrintWriter writer;
	String datotekaIme = "";
	try {
		datotekaIme = predpona + algoritem + sdfDate.format(new Date())+".txt";
		writer = new PrintWriter(izhodnaLokacija + "\\" + datotekaIme, "UTF-8");
		writer.println(ukaz);
		if(algoritem == 1){
			algoritemClass = new DawkinsWeaselAlgoritem(tipIzvajanja,
			    velikostPopulacije, steviloGeneracij,
			    odstotekStarostneRazlike, vhodniNiz, writer);
			for(int i = 0 ; i < ponovitve; i++){
				algoritemClass.Pozeni();
				task.updateProgressBar();
			}
		}else if (algoritem == 2){
			algoritemClass = new NahrbtnikAlgoritem(tipIzvajanja,
			    velikostPopulacije, steviloGeneracij,
			    odstotekStarostneRazlike, nosilnost, stMutacijZaGenom,
			    tezaPredmetov, uporabnostPredmetov, writer);
			for(int i = 0 ; i < ponovitve; i++){
				algoritemClass.Pozeni();
				task.updateProgressBar();
			}
		}else if (algoritem == 3){
			algoritemClass = new MichalewiczFunctionAlgoritem(tipIzvajanja,
			    velikostPopulacije, steviloGeneracij,
			    odstotekStarostneRazlike, n, writer);
			for(int i = 0 ; i < ponovitve; i++){
				algoritemClass.Pozeni();
				task.updateProgressBar();
			}
		}
		writer.close();
		task.updateIzpis("Ustvarjena datoteka: " + datotekaIme + "\n");
	} catch (Exception e) {
		throw new Exception("...");
	}
}
```

Koda 6.4: Začetek izvajanja.

6.2 Enako delovanje algoritmov

Način implementacije GA je v veliki meri odvisen od predstavitve genoma. V tej nalogi imamo opravka s tremi različnimi algoritmi in v vsakem je genom zapisan v nekoliko drugačni obliki. Posledično je način manipulacije tega zapisa v operacijah križanja in mutacije tudi različen. Te in funkcije uspešnosti si bomo za vsak algoritem posebej pogledali

v nadaljevanju. To poglavje je namenjeno delom GA, ki smo jih implementirani na enak način za vse tri probleme.

6.2.1 Cikel izvajanja

Cikel izvajanja je prva in očitna metoda, ki je enaka v vseh implementacijah. Vsi GA imajo enako osnovno obliko, zato je razumljivo, da tudi ti sledijo tipičnemu zaporedju izvajanja. Cikel izvajanja se v našem primeru nahaja znotraj metode *Pozeni*, ki je prikazana spodaj (Koda 6.5). Ta metoda začne izvajanje algoritma. Prvi korak je inicializacija, kjer se vse spremenljivke, ponastavijo na začetne vrednosti (potrebno, če algoritem ni pognan prvič). Zatem se populacija napolni z naključnimi vrednostmi, kar je tipičen način ustvarjanja začetne populacije v GA. Za vse nastale osebke v populaciji se izračunajo fitnes vrednosti. Na koncu se začne ponavljajoče se izvajanje križanja, mutiranja in ponovnega izračuna fitnes vrednosti, ki traja, dokler se ne izpolni zaustavitveni pogoj. Preden se izvajanje povsem zaključi, se rezultati GA zapišejo v datoteko.

```
public void Pozeni(){
	inicializacija();
	for(int i = 0; i < velikostPopulacije; i++){
		populacija.add(ustvariNakljucenGenom());
	}
	izracunajFitnesPopulaciji();
	while(trenutnaGeneracija < steviloGeneracij){
		krizaj();
		mutiraj();
		izracunajFitnesPopulaciji();
		trenutnaGeneracija++;
	}
	writer.println(rezultat);
}
```

Koda 6.5: Cikel izvajanja.

6.2.2 Selekcija

Operacija selekcije je v vseh algoritmih implementirana na enak način. Uporabljena tehnika selekcije je tristrani turnir, ker je, kot smo omenili v teoretičnem delu naloge, turnir ena boljših tehnik za izbor staršev naslednje generacije. Proces selekcije enega starša se začne z naključnim izborom treh osebkov iz populacije. Če je isti osebek izbran več kot enkrat, ponovimo naključen izbor. Kot je razvidno iz spodnjega odrezka kode (Koda 6.6), so objekti kandidatov tipa *DawkinsWeaselGenom*. Razredi takšnega tipa so namenjeni hranjenju

vrednosti genoma. V tem primeru je koda del Dawkins Weasel algoritma, zato je uporabljen ta genom. Algoritem zatem sodelujoče kandidate glede na njihovo fitnes vrednost razporedi od najboljšega (*kand1*) do najslabšega (*kand3*). V primeru funkcije Michalewicz moramo biti pozorni, saj je namen odkritje minimuma in so posledično nižje fitnes vrednosti boljše. Razmerje za izbor posameznega kandidata je konstantna vrednost v vseh algoritmih. Tipično razmerje uporabljeno za dvostranski turnir je 3:1 v korist boljšega osebka. Za ohranjanje podobnega razmerja, smo ga za tristrani turnir razširili tako, da ima najboljši osebek 73 %, drugi 23 % in tretji 4 % verjetnost, da postane zmagovalec turnirja in prenese svoj dedni material v naslednjo generacijo.

```
private DawkinsWeaselGenom pridobiKandidata (){
	DawkinsWeaselGenom kand1 = populacija.get(rand.nextInt(populacija.size()));
	DawkinsWeaselGenom kand2 = populacija.get(rand.nextInt(populacija.size()));
	DawkinsWeaselGenom kand3 = populacija.get(rand.nextInt(populacija.size()));
	DawkinsWeaselGenom kandidatTemp ;
	if(kand1.equals(kand2) || kand1.equals(kand3) || kand2.equals(kand3)){
		return pridobiKandidata();
	}
	if(kand2.getFitnes() > kand1.getFitnes() ){
		kandidatTemp = kand1;
		kand1 = kand2;
		kand2 = kandidatTemp;
	}
	if(kand3.getFitnes() > kand1.getFitnes() ){
		kandidatTemp = kand1;
		kand1 = kand3;
		kand3 = kand2;
		kand2 = kandidatTemp;
	}else if(kand3.getFitnes() > kand2.getFitnes() ){
		kandidatTemp = kand2;
		kand2 = kand3;
		kand3 = kandidatTemp;
	}
	int i = rand.nextInt(100);
	return ( i < RAZMERJE1) ? kand1: (i < RAZMERJE2)? kand2: kand3;
}
```

Koda 6.6: Primer operacije selekcije.

6.3 Dawkins Weasel problem

Dawkins Weasel problem je prvi, za katerega si bomo pogledali implementacijo GA. Ker je prvi, bo tudi vseboval obsežnejše primere kode, v primerjavi z naslednjimi algoritmi, ki bodo v namene jedrnatosti vsebovali le ključne dele kode ali kodo, ki se razlikuje od te. Posebnost tega algoritma je predpriprava vhodnih podatkov. Iskani niz, ki je edini podatek, specifičen za ta algoritem, lahko vsebuje poljubne znake. Večje, kot je število znakov, ki jih GA skuša

ujemati, zahtevnejši je problem. Zato smo se v implementaciji omejili na male črke slovenske abecede in znak za presledek. Iz tega razloga se v inicializaciji spremenljivk vse črke v iskalnem nizu spremenijo v male črke in vsi ostali znaki, ki ne spadajo med prej omenjenih 26, so odstranjeni iz niza. Genom je sestavljen iz enakega števila genov, kot je znakov v prirejenem iskalnem nizu. Gen je predstavljen kot eden izmed veljavnih znakov. Začetna populacija oziroma osebki, ki jo sestavljajo, so ustvarjeni z izborom naključnega znaka za vsak gen.

6.3.1 Funkcija uspešnosti

Izračun fitnesa za Dawkins Weasel algoritem je zelo preprost. Fitnes vrednost je enaka številu pravilno postavljenih črk v genomu. Implementacija celotne funkcije je prikazana v nadaljevanju (Koda 6.7). Začne se z ohranjanjem optimuma. Ta funkcija kopira najboljše osebke iz prejšnje generacije v trenutno. To delovanje smo v teoretičnem delu imenovali elitizem. Kopirani osebki so dodani na izbrano velikost populacije, kar pomeni, da se velikost populacije skozi izvajanje nekoliko spreminja glede na število trenutno najboljših rešitev. Zatem se za vsak osebek v populaciji preveri, koliko črk v genomu se ujema z iskanim nizom. Ta vrednost postane fitnes osebka. Vsem genom trenutnega osebka se poveča starost. Istočasno se v populaciji iščejo tudi osebki z največjim fitnesom. Ti se shranijo v tabelo optimalnih osebkov. Ob naslednjem računanju fitnes vrednosti se bodo ti osebki pri ohranjanju optimuma avtomatsko prenesli v naslednjo generacijo. Da takšno ohranjanje ni prekomerno oziroma škodljivo, se prenese samo osebek z najboljšim fitnesom oziroma do enega več od 4 % velikosti populacije, če si delijo največji fitnes. Čeprav je morda odrezek kode pri ciklu izvajanja nakazoval, da je preseženo število generacij edini zaustavitveni pogoj, ima Dawkins Weasel algoritem še enega. V primeru, da se najde osebek, katerega genom je identičen iskanemu nizu se število generacij, ki bi jih morali izvesti, izenači z zaporednim številom trenutne generacije. Ob naslednjem preverjanju zaustavitvenega pogoja bo ta izpolnjen in algoritem se bo zaključil. Delovanje se ne prekine takoj, ker je še vedno potrebno izračunati vse fitnes vrednosti zadnje generacije. Spremeni se spremenljivka, ki hrani število generacije, do katere bi se algoritem sicer izvajal in ne tista s trenutno generacijo, saj podatek o trenutni generaciji potrebujemo za oceno uspešnosti delovanja. Kot smo pokazali v predstavitvi cikla izvajanja, se zaustavitveni pogoj preveri takoj za izračunom fitnesa, tako da zakasnjen konec izvajanja v nobeni meri ne vpliva na rezultate.

```
private void izracunajFitnesPopulaciji(){
	ohranjanjeOptimuma();
	for(DawkinsWeaselGenom dwg : populacija){
		int fitnes = 0;
		for(int i = 0 ; i < dwg.getGenom().size(); i++){
			if(dwg.getGenom().get(i).equals(iskaniNiz.get(i))){
				fitnes++;
			}
		}
		dwg.setFitnes(fitnes);
		dwg.povecajStarost();
		if(fitnes >maxFitnes){
			optimalniOsebki.clear();
			optimalniOsebki.add(dwg);
			maxFitnes = fitnes;
		}else if(fitnes == maxFitnes && optimalniOsebki.size() < (0.04 *
		    velikostPopulacije + 1)){
			optimalniOsebki.add(dwg);
		}
		if(fitnes == iskaniNiz.size()){
			steviloGeneracij = trenutnaGeneracija;
		}
	}
}
```

Koda 6.7: Funkcija uspešnosti za Dawkins Weasel algoritem.

6.3.2 Križanje

Križanje se v naši aplikaciji lahko opravi na pet različnih načinov. Prvi je uniformno križanje, kot je opisano v 3.7.3 (Uniformno križanje). Naslednji štirje so različni načini upoštevanja starosti genov pri njihovi uspešnosti propagiranja v naslednjo generacijo. Teoretičen opis vseh štirih postopkov je predstavljen v 5.1 (Verjetnost izbire genov na osnovi starosti v operacijah križanja). Metoda križanja, ki je predstavljena v nadaljevanju (Koda 6.8), se začne z zanko, ki zagotavlja, da se bo križanje izvajalo, dokler ne bo populacija vsebovala toliko osebkov, kot jih je bilo specificiranih v vhodnih podatkih. S selekcijo, ki je opisana v 6.2.2 (Selekcija), se pridobita dva osebka, ki sta potrebna za operacijo križanja. Če je dvakrat izbran isti osebek, se izbor ponovi. V uniformnem delovanju se križanje izvaja za vsak lokus posebej. Tradicionalno uniformno križanje se izvede v primerih, ko je tip delovanja 0, 1 ali 2. Verjetnost je v tem primeru izražena kot met kovanca. Spremenljivka *tmp* hrani podatek o tem ali bo izbran gen iz prvega starša ali ne, v katerem primeru je jasno izbran gen drugega. Zatem se zvrstijo štirje načini delovanja, ki smo jih ustvarili za namen te naloge. Najprej primer, kjer imajo starejši osebki prednost pri izbiri, zatem mlajši, povprečno stari in nazadnje še geni, ki so ali zelo stari ali zelo mladi. Za prva dva načina delovanja se po že predstavljenih postopkih izračuna vrednost *rez*, ki predstavlja prednost,

ki jo bo privilegiran gen prejel. Osnovni verjetnosti izbire prvega gena se zatem ta prednost prišteje ali odšteje (glede na to, kateri gen ima preferenčno starost). V vseh načinih delovanja je največja prednost, ki jo gen lahko pridobi, enaka odstotku starostne razlike. Nazadnje se na podlagi *tmp* spremenljivke v otroka prepiše ustrezen gen in njegova starost. Ko je genom vsakega otroka poln se ga doda v novo populacijo.

```
private void krizaj (){
ArrayList <DawkinsWeaselGenom> pt = new ArrayList<DawkinsWeaselGenom>();
   boolean tmp = false; int razlika; double rez ;
   while(pt.size() < velikostPopulacije){
       DawkinsWeaselGenom stars1 = pridobiKandidata();
       DawkinsWeaselGenom stars2 = pridobiKandidata();
       if(stars1.equals(stars2)){ continue;}
       DawkinsWeaselGenom otrok = new DawkinsWeaselGenom();
       for(int i = 0 ; i < iskaniNiz.size(); i++){
           switch (tipIzvajanja){
           case 0: case 1: case 2:  //uniformno križanje
              tmp = rand.nextBoolean();
              break;
           case 3: case 7: case 11:  //starejši imajo večjo verjetnost izbire
              razlika = stars1.getStarost().get(i) - stars2.getStarost().get(i);
              rez = odstotekStarostneRazlike - Gaussian.phi(razlika, 0, sigma) *
                normaliziranjeGauss;
              tmp = (rand.nextDouble() < 0.5 + ((razlika < 0)? -rez : rez) )? true:
                false;
              break;
           case 4: case 8: case 12:  //mlajši imajo večjo verjetnost izbire
              razlika = stars1.getStarost().get(i) - stars2.getStarost().get(i);
              rez = odstotekStarostneRazlike - Gaussian.phi(razlika, 0, sigma) *
                normaliziranjeGauss;
              tmp = (rand.nextDouble() < 0.5 + ((razlika < 0)? rez : -rez) )? true :
                false;
              break;
           case 5: case 9: case 13:  //povprečni imajo večjo verjetnost izbire
              tmp = (rand.nextDouble() < 0.5 +
                (Gaussian.phi(stars1.getStarost().get(i), povprecnaPricakovanaStarost,
                sigma) * normaliziranjeGauss) - (Gaussian.phi(
                stars2.getStarost().get(i), povprecnaPricakovanaStarost, sigma) *
                normaliziranjeGauss) )? true : false;
              break;
           case 6: case 10: case 14:  //ekstremi imajo večjo verjetnost izbire
              tmp = (rand.nextDouble() < 0.5 + (odstotekStarostneRazlike -
                Gaussian.phi(stars1.getStarost().get(i), povprecnaPricakovanaStarost,
                sigma) * normaliziranjeGauss) - (odstotekStarostneRazlike -
                Gaussian.phi(stars2.getStarost().get(i), povprecnaPricakovanaStarost,
                sigma) * normaliziranjeGauss) )? true : false;
              break;
           }
           otrok.getGenom().add((tmp)? stars1.getGenom().get(i) :
             stars2.getGenom().get(i));
           otrok.getStarost().add(( tmp)? stars1.getStarost().get(i) :
             stars2.getStarost().get(i));
       } pt.add(otrok); } populacija = pt; }
```

Koda 6.8: Operacija križanja za Dawkins Weasel algoritem.

6.3.3 Mutacija

Za vse osebke v populaciji velja enaka in konstantna stopnja mutacije, ki je fiksno določena v aplikaciji in znaša 0,05. Za stalno vrednost smo se odločili, zato ker spreminjanje stopnje mutacije ni del namena aplikacija in je zadovoljiva konstantna vrednost. V vseh algoritmih lahko mutacija deluje na tri različne načine. Prvi je standardna uniformna mutacija. Pri takšnem delovanju se iz genoma naključno izbere en gen. V primeru Dawkins Weasel (Koda 6.9) zamenjamo znak, ki je do sedaj predstavljal mutirani gen z novim naključno izbranim znakom iz dovoljenega nabora. Starost spremenjenega gena se ponastavi na 0. Naslednji dve možnosti delovanja preferenčno obravnavata starejše oziroma mlajše gene. Starostno odvisni mutaciji sta opisani v poglavju 5.2 (Verjetnost izbire genov na osnovi starosti v operacijah mutacije). V osnovi se to, kar je bila prej naključna izbira gena, spremeni v izbiro z ruleto (deluje na enak način kot selekcija z ruleto). Določi se naključna vrednost med 0 in seštevkom starosti vseh genov v osebku. Prvi gen, ki ob ponovnem seštevanju starosti preseže to vrednost, mutira. Mutacija se opravi na enak način kot pri uniformnem mutiranju – stara vrednost gena se zamenja z naključnim veljavnim znakom. Postopek v primeru privilegiranja mlajših genov je identičen, le da se uporabljajo obratne vrednosti starosti. Po mutaciji se starost gena vedno nastavi na 0.

```
private void mutiraj(){
   int tmp = 0; double skupnaStarost;
   for(int i = 0; i < populacija.size(); i++){
       if (rand.nextDouble() < stopnjaMutacije){
           stMutacij++;
           switch(tipIzvajanja){
           case 0: case 3: case 4: case 5: case 6:  //uniformno mutiranje
              tmp = rand.nextInt(iskaniNiz.size());
              populacija.get(i).getGenom().set(tmp,
                SEZNAMCRK.get(rand.nextInt(SEZNAMCRK.size())));
              populacija.get(i).getStarost().set(tmp, 0);
              break;
           case 1: case 7: case 8: case 9: case 10: //starejši imajo večjo verjetnost
              skupnaStarost = 0;
              for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
                  skupnaStarost += populacija.get(i).getStarost().get(s);
              }
              int rnd = rand.nextInt((int) skupnaStarost); skupnaStarost = 0;
              for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
                  skupnaStarost +=populacija.get(i).getStarost().get(s);
                  if(skupnaStarost > rnd){
                     populacija.get(i).getGenom().set(s,
                       SEZNAMCRK.get(rand.nextInt(SEZNAMCRK.size())));
                     populacija.get(i).getStarost().set(s, 0);
                     break;
                  }
```

```
                }
                break;
            case 2: case 11: case 12: case 13: case 14: //mlajši imajo večjo verjetnost
                skupnaStarost = 0;
                for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
                    skupnaStarost += ((double)1 /
                      populacija.get(i).getStarost().get(s));
                }
                double rndd = rand.nextDouble()*skupnaStarost;
                skupnaStarost = 0;
                for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
                    skupnaStarost += ((double)1 /populacija.get(i).getStarost().get(s));
                    if(skupnaStarost > rndd){
                        populacija.get(i).getGenom().set(s,
                          SEZNAMCRK.get(rand.nextInt(SEZNAMCRK.size())));
                        populacija.get(i).getStarost().set(s, 0);
                        break;
                    }
                }
                break;
} } } }
```

Koda 6.9: Operacija mutacije za Dawkins Weasel algoritem.

6.4 Problem nahrbtnika

Predmeti, ki jih lahko vstavimo v nahrbtnik, so v našem GA implementirani kot dve polji celih števil, v kateri se vstavijo vrednosti teže in uporabnosti predmetov, pridobljene iz vhodnih podatkov. Osebek v problemu nahrbtnika je predstavljen kot tabela Boolovih vrednosti enake dolžine kot prejšnji polji. Boolove vrednosti nakazujejo, ali je predmet s težo in uporabnostjo na istoležnih mestih v preostalih dveh tabelah vstavljen v nahrbtnik. Osebki prve populacije niso ustvarjeni s popolnim naključjem, saj bi v večini primerov to vodilo v ekstremno slabo začetno populacijo. Če bi vsak predmet imel naključnih 50 %, da je vstavljen v nahrbtniku, bi velika večina nahrbtnikov krepko presegla omejitev teže. S tem namenom se verjetnost postavitve predmeta v nahrbtnik pri ustvarjanju začetne populacije izrazi kot količnik med nosilnostjo in skupno težo vseh predmetov. To zagotavlja, da imajo začetni osebki v povprečju podobne teže kot je nosilnost nahrbtnika.

6.4.1 Funkcija uspešnosti

Za vsak osebek v populaciji funkcija uspešnosti začne z izračunom teže nahrbtnika, kjer se seštejejo teže vseh vključenih predmetov. Tipično se nahrbtnike, ki presegajo dovoljeno nosilnost, oceni s fitnes vrednostjo 0. V našem primeru (Koda 6.10) smo se odločili, da želimo osebkom, ki so to mejo prekoračili za manjšo vrednost, vseeno dati nekaj prednosti

pred še težjimi. Zato smo to izrazili kot razliko med nosilnostjo in težo nahrbtnika. Vsi osebki, ki prekoračijo nosilnost, bodo tako imeli negativno fitnes vrednost, začenši z osebki, pretežkimi za eno enoto, katerih fitnes bo enak -1. Izračun fitnes vrednosti za ostale osebke se začne s seštevkom uporabnosti vseh predmetov v nahrbtniku. Če bi bila to celotna funkcija uspešnosti, bi imela dva osebka z enakim seštevkom uspešnosti enako verjetnost razmnoževanja. To delovanje smo želeli razširiti, tako da bi imel prednost tisti osebek med njima, ki je lažji (ker to pomeni, da lahko damo več dodatnih predmetov v nahrbtnik). Zato Fitnes vrednosti prištejemo še razliko nosilnosti in teže genoma.

```
for(NahrbtnikGenom ng: populacija){
	izracunajTezoNahrbtnika(ng);
	int fitnes = 0;
	if(ng.getTezaGenoma() > nosilnost){
		fitnes = nosilnost - ng.getTezaGenoma();
	}else{
		for(int i = 0 ; i < ng.getGenom().size(); i++){
			if(ng.getGenom().get(i)){
				fitnes+=uporabnostPredmetov.get(i);
			}
		}
		fitnes += (nosilnost - ng.getTezaGenoma());
	}
	ng.setFitnes(fitnes);
	...
}
```

Koda 6.10: Funkcija uspešnosti za problem nahrbtnika.

6.4.2 Križanje

Križanje je v problemu nahrbtnika identično križanju v Dawkins Weasel algoritmu, kot prikazuje naslednji odrezek kode (Koda 6.11). Podrobnejše informacije o delovanju so v poglavjih 5.1 (Verjetnost izbire genov na osnovi starosti v operacijah križanja) in 6.3.2 (Križanje).

```
switch (tipIzvajanja){
case 0 : case 1: case 2:
	tmp = rand.nextBoolean();
	break;
case 3: case 7: case 11: //starejši imajo večjo verjetnost izbire
	razlika = stars1.getStarost().get(i) - stars2.getStarost().get(i);
	rez = odstotekStarostneRazlike - Gaussian.phi(razlika, 0, sigma) *
	    normaliziranjeGauss;
	tmp = (rand.nextDouble() < 0.5 + ((razlika < 0)? -rez : rez) )? true : false;
	break;
```

```
case 4: case 8: case 12:  //mlajši imajo večjo verjetnost izbire
      razlika = stars1.getStarost().get(i) - stars2.getStarost().get(i);
      rez = odstotekStarostneRazlike - Gaussian.phi(razlika, 0, sigma) *
          normaliziranjeGauss;
      tmp = (rand.nextDouble() < 0.5 + ((razlika < 0)? rez : -rez) )? true : false;
      break;
case 5: case 9: case 13: //povprečno stari imajo večjo verjetnost izbire
      tmp = (rand.nextDouble() < 0.5 + (Gaussian.phi(stars1.getStarost().get(i),
          povprecnaPricakovanaStarost, sigma) * normaliziranjeGauss) -
          (Gaussian.phi(stars2.getStarost().get(i), povprecnaPricakovanaStarost,
          sigma) * normaliziranjeGauss) )? true : false;
      break;
case 6: case 10: case 14: //ekstremi imajo večjo verjetnost izbire
      tmp = (rand.nextDouble() < 0.5 + (odstotekStarostneRazlike - Gauss-
          ian.phi(stars1.getStarost().get(i), povprecnaPricakovanaStarost, sigma) *
          normaliziranjeGauss) - (odstotekStarostneRazlike - Gauss-
          ian.phi(stars2.getStarost().get(i), povprecnaPricakovanaStarost, sigma) *
          normaliziranjeGauss) )? true : false;
      break;
}
otrok.getGenom().add( ( tmp)?stars1.getGenom().get(i):stars2.getGenom().get(i));
povprecnaStarostKrizanje += (tmp)? stars1.getStarost().get(i) :
      stars2.getStarost().get(i);
otrok.getStarost().add((tmp)?stars1.getStarost().get(i):stars2.getStarost().get(i));
```

Koda 6.11: Operacija križanja za problem nahrbtnika.

6.4.3 Mutacija

Mutacija je v problemu nahrbtnika prav tako enaka primeru Dawkins Weasel algoritma. Zato smo tu vključili le kodo mutacije, kjer imajo starejši geni prednost pri izboru za sodelovanje v operaciji (Koda 6.12). Edina razlika je, da se mutirani gen obrne – izbran predmet se odstrani ali doda v nahrbtnik. Podrobnejši opisi delovanja mutacije se nahajajo v 5.2 (Verjetnost izbire genov na osnovi starosti v operacijah mutacije) in 6.3.3 (Mutacija).

```
case 1: case 7: case 8: case 9: case 10: //starejši imajo večjo verjetnost izbire
    skupnaStarost = 0;
    for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
        skupnaStarost +=populacija.get(i).getStarost().get(s);
    }
    int rnd = rand.nextInt((int) skupnaStarost); skupnaStarost = 0;
    for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
        skupnaStarost += populacija.get(i).getStarost().get(s);
        if(skupnaStarost > rnd){
              populacija.get(i).getGenom().set(s, populacija.get(i).getGenom().get(s)
                  ^ true);
              povprecnaStarostMutacija +=populacija.get(i).getStarost().get(s);
              populacija.get(i).getStarost().set(s, 0);
              break; } } break;
```

Koda 6.12: Operacija mutacije za problem nahrbtnika.

6.5 Funkcija Michalewicz

Problem izračuna funkcije Michalewicz je zanimiv, ker je edini v tej nalogi implementiran GA, v katerem so geni predstavljeni kot številske vrednosti. Genom je sestavljen iz n genov, kjer je n število dimenzij v funkciji Michalewicz in je predstavljen kot polje realnih števil. Funkcija je podrobneje opisana v 4.3 (Funkcija Michalewicz). Vsak gen lahko zasede vrednost med 0 in π. Začetni osebki nastanejo z vstavitvijo naključne vrednosti iz tega območja v vsak gen. Dodatna posebnost tega algoritma je tudi starost genov, ki ni vedno celo število.

6.5.1 Funkcija uspešnosti

Funkcija uspešnosti je v tem primeru sama funkcija Michalewicz. Pri uporabi te funkcije moramo biti pozorni na to, da iščemo globalni minimum. Posledično se funkcija selekcije za ta algoritem nekoliko razlikuje od ostalih dveh, kot je prikazano v nadaljevanju (Koda 6.13).

```
for(MichalewiczFunctionGenom mfg : populacija){
	fitnes = 0;
	for(int i = 0; i < mfg.getGenom().size(); i++){
		fitnes -= Math.sin(mfg.getGenom().get(i)) * Math.pow(Math.sin(((i + 1) *
		    mfg.getGenom().get(i) * mfg.getGenom().get(i))/Math.PI), 2*m);
	}
	...
}
```

Koda 6.13: Funkcija uspešnosti za funkcijo Michalewicz.

6.5.2 Križanje

Številska predstavitev genov povzroči določene razlike pri operaciji križanja v primerjavi z drugimi algoritmi. Implementacija je predstavljena v nadaljevanju (Koda 6.14). Kot osnova je uporabljeno vmesno križanje. Pri tradicionalnem izvajanju se vrednost potomca naključno izbere iz območja vrednosti med obema staršema. Meji tega območja se v naslednji kodi nahajata v spremenljivkah *zgornja* in *spodnja*. Križanje, kjer imajo starejši geni prednost, se začne kot vedno do sedaj – z izračunom prednosti enega gena pred drugim. Razlika proti drugim algoritmom nastopi za tem. Namesto, da bi s to vrednostjo povečali verjetnost izbora privilegiranega gena, z njim zmanjšamo območje, v katerem lahko nastane potomec. To območje zmanjšamo na strani manj želenega gena. Območje zmanjšamo za odstotek, ki

ga predstavlja prednost enega gena pred drugim. Ko sedaj naključno izberemo vrednost med obema mejama, je verjetnost, da bo izbrana vrednost bližja preferenčnemu genu večja. Starost novega gena je povprečna starost obeh staršev, kar je ponovno posebnost, ker je to edina operacija križanja, kjer se starost gena neposredno ne podeduje od staršev.

```
MichalewiczFunctionGenom otrok = new MichalewiczFunctionGenom();
    for(int i = 0 ; i < n; i++){
       spodnja = Math.min(stars1.getGenom().get(i), stars2.getGenom().get(i));
       zgornja = Math.max(stars1.getGenom().get(i), stars2.getGenom().get(i));
       switch (tipIzvajanja){
       case 0: case 1: case 2: break;
       case 3: case 7: case 11: //starejši imajo večjo verjetnost izbire
          razlika = stars1.getStarost().get(i) - stars2.getStarost().get(i);
          rez = odstotekStarostneRazlike - Gaussian.phi(razlika, 0, sigma) *
            normaliziranjeGauss;
          if(razlika < 0){
              if(zgornja == stars1.getGenom().get(i)){
                  zgornja -= (zgornja - spodnja) * rez;
              }else{
                  spodnja += (zgornja - spodnja) * rez;
              }
          }else{
              if(zgornja == stars2.getGenom().get(i)){
                  zgornja -= (zgornja - spodnja) * rez;
              }else{
                  spodnja += (zgornja - spodnja) * rez;
              }
          }
          break;
          ...
       case 5: case 9: case 13: //povprečno stari imajo večjo verjetnost izbire
          rez = (Gaussian.phi(stars1.getStarost().get(i), povprecnaPricakovanaStarost,
            sigma) * normaliziranjeGauss) - (Gaussian.phi(stars2.getStarost().get(i),
            povprecnaPricakovanaStarost, sigma) * normaliziranjeGauss);
          if(rez < 0){
              if(zgornja == stars1.getGenom().get(i)){
                  zgornja -= (zgornja - spodnja) * rez;
              }else{
                  spodnja += (zgornja - spodnja) * rez;
              }
          }else{
              if(zgornja == stars2.getGenom().get(i)){
                  zgornja -= (zgornja - spodnja) * rez;
              }else{
                  spodnja += (zgornja - spodnja) * rez;
              }
          }
          break;
       ... }
       otrok.getGenom().add(spodnja + (zgornja - spodnja) * rand.nextDouble());
       otrok.getStarost().add((stars1.getStarost().get(i)+stars2.getStarost().get(i))
          /2);
}
```

Koda 6.14: Operacija križanja za funkcijo Michalewicz.

6.5.3 Mutacija

Operacija mutacije deluje na podoben način kot v obeh algoritmih do sedaj. Vrednost mutiranih genov je izbrana naključno iz območja možnih vrednosti, kot je razvidno iz spodnjega odrezka kode (Koda 6.15).

```
case 0: case 3: case 4: case 5: case 6:
	int tmp = rand.nextInt(n);
	populacija.get(i).getGenom().set(tmp, rand.nextDouble()*max);
	povprecnaStarostMutacija +=populacija.get(i).getStarost().get(tmp);
	populacija.get(i).getStarost().set(tmp, 0.0);
	break;
case 1: case 7: case 8: case 9: case 10: //starejši imajo večje možnosti izbire
	skupnaStarost = 0;
	for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
		skupnaStarost +=populacija.get(i).getStarost().get(s);
	}
	rndd = rand.nextDouble()*skupnaStarost;
	skupnaStarost = 0;
	for(int s = 0 ; s < populacija.get(i).getStarost().size(); s++){
		skupnaStarost +=populacija.get(i).getStarost().get(s);
		if(skupnaStarost > rndd){
			populacija.get(i).getGenom().set(s, rand.nextDouble()*max);
			populacija.get(i).getStarost().set(s, 0.0);
			break;
		}
	}
	break;
case 2: case 11: case 12: case 13: case 14: //mlajši imajo večjo možnosot izbire
	...
	break;
```

Koda 6.15: Operacija mutacije za funkcijo Michalewicz.

7 PRIPRAVA IN IZVEDBA EKSPERIMENTA

Ta del naloge bomo namenili predstavitvi okoliščin, v katerih smo opravili eksperiment. Vse GA smo izvajali v enakem številu in ob enakih osnovnih parametrih. Te parametre smo nastavili na podlagi predhodnega testiranja. Vsak algoritem smo ob določenem naboru parametrov izvedli 1000-krat. S takšnim številom ponovitev želimo izključiti naključnost iz delovanja GA in omogočiti statistično analizo delovanja in rezultatov. Velikost populacije je bila v vseh primerih 400 in če se optimalna rešitev ne doseže predčasno, se je izvajanje nadaljevalo do generacije številka 1000. Za takšne vrednosti smo se odločili, ker pričakujemo, da bodo GA v tem številu operacij uspeli izkoristiti svoj potencial in bi bilo nadaljnje izvajanje neproduktivno. Izjema pri tem je funkcija Michalewicz, kjer smo opazili še precej izboljšav okrog tisoče generacije. Zato smo število generacij, za ta problem povečali na 1100. Iz dosedanjih vhodnih podatkov lahko razberemo, da je bilo za reševanje vsakega problema ob določenih parametrih ustvarjenih in ocenjenih ogromno število osebkov. Za primer lahko podamo problem nahrbtnika, kjer je bilo uporabljenih 400 milijonov osebkov. Vsak takšen primer algoritma smo izvedli v vseh sedmih različnih načinih delovanja (kombinacij operacij križanja in mutacije s starostno odvisnim izborom genov ne bomo obravnavali v tej nalogi). Vse probleme, ki smo jih rešili v tej nalogi, so bili izvedeni dvakrat. Prvič ob odstotku starostne razlike 0,15 in drugič 0,5. To pomeni, da je največja možna razlika v razmerju verjetnosti izbora dveh genov v operaciji križanja 65-35 % oziroma 100-0 % (način vpeljave starostne odvisnosti zagotavlja, da so takšne velike razlike zelo redke in se pojavijo samo med geni z ekstremnimi razlikami v starosti). Algoritem Dawkins Weasel smo reševali za dva različna iskalna niza, zato smo prvi problem označili kot A in drugega kot B. Prvi iskalni niz je relativno kratko besedilo – "Vpeljava verjetnosti izbire genov na osnovi starosti pri genetskih algoritmih". Za ta problem nismo pričakovali, da bo potrebnih 1000 generacij za odkritje optimuma in bomo zato lahko opazovali razlike v delovanju pri hitrejši konvergenci osebkov v končno rešitev. A problem, katerega rešitev vsebuje 77 znakov, smo nadgradili s problemom B, ki ima iskalni niz dolžine 255 znakov. Izvajanje istega algoritma nad dvema različnima iskalnima nizoma nam bo omogočalo preveriti konsistentnost delovanja kot tudi možne razlike, ki so nastale med izvajanjem preprostejšega in naprednejšega problema. Podobno smo tudi problem nahrbtnika izvedli z dvema različnima naboroma specifičnih vhodnih podatkov (osnovni še vedno ostanejo enaki). Primera se razlikujeta v nosilnosti nahrbtnika in veliko pomembneje v številu premetov, ki jih lahko uporabimo. Nosilnost nahrbtnika A je 105, njegovo vsebino pa je GA

lahko izbiral med 250 predmeti, medtem ko je nosilnost B primera 189 in je nahrbtnik mogoče napolniti s 450 različnimi predmeti. Vrednosti teže in uporabnosti predmetov v obeh primerih so bile naključno generirane ob enakomerni porazdelitvi in zavzemajo vrednosti med 1 in 100. Minimum funkcije Michalewicz smo se odločili iskati na problemu z desetimi dimenzijami (n = 10), ker se je problem z dvema ali petimi, ob izbranih parametrih izvajanja, izkazal kot trivialen. Parameter m je v aplikaciji fiksno nastavljen na vrednost 10.

Pri izvajanju algoritmov je potrebno omeniti temo, o kateri nismo veliko govorili pri osnovanju delovanja, niti pri njihovi implementaciji – hitrost. Tega zelo pomembnega aspekta delovanja GA pri izgradnji nismo upoštevali, ker za nas ni posebej pomemben. Namen je raziskati zgolj vpliv starostno odvisne izbire genov na rezultate, ki jih GA lahko pridobijo. Pri tem je seveda pomembno tudi, kako hitro pridobimo rezultate, vendar se takšna metrika meri v številu generacij. Ne glede na to je hitrost oziroma čas, potreben za izvajanje vsakega algoritma, zelo pomemben in delovanje z upoštevanjem starosti genov zagotovo upočasni izvajanje. Ker algoritmi niso posebej optimizirani, so razlike še toliko bolj očitne. Spremembe v operaciji mutacije povzročijo minimalne razlike, medtem ko so spremembe v operaciji križanja veliko bolj časovno zahtevne (predvsem Gaussova funkcija) in čas izvajanja podvojijo proti tradicionalnemu delovanju.

Med izvajanjem merimo do sedem različnih metrik. Točen nabor se nekoliko razlikuje od algoritma do algoritma. Prva metrika je najboljši fitnes, kjer se v izhodno datoteko izpiše najboljši fitnes, ki je bil dosežen med izvajanjem. Ker uporabljamo elitizem, je ta vrednost enaka tudi najboljšemu osebku končne populacije. Ta podatek je seveda najpomembnejši pri ugotavljanju učinkovitosti algoritma in predstavlja sposobnost algoritma, da doseže najboljše rezultate. Naslednji izhodni podatek je generacija najboljšega fitnesa, ki nosi podatek o generaciji, v kateri se je končna najboljša fitnes vrednost prvič pojavila. Generacija pojava najboljšega fitnesa govori o hitrosti, s katero je algoritem sposoben pridobiti najboljšo rešitev. Naslednji trije izhodni podatki beležijo povprečne starosti: povprečna starost genov, povprečna starost mutacije in povprečna starost križanja. Povprečna starost genov je povprečna starost vseh genov, ki so kadarkoli obstajali v katerem koli osebku za dano izvedbo algoritma. Povprečna starost mutacije je povprečna starost vseh genov, ki so kadarkoli tekom delovanja določene instance ali ponovitve algoritma mutirali. Podobno je povprečna starost križanja povprečna vrednost starosti vseh genov, ki so napredovali v operaciji križanja (so se prenesli v naslednjo generacijo). Slednja metrika je nesmiselna v primeru funkcije Michalewicz, kjer ni izbran eden izmed genov, ampak je vrednost nastalega gena izbrana iz območja med obema staršema in posledično

starost potomcev ni odvisna od prednosti enega gena (starost otroka je povprečna starost obeh staršev). Zadnji dve spremenljivki sta posebni, ker vsebujeta po eno vrednost za vsako generacijo - če se algoritem izvaja skozi 1000 generacij, se v izhodno datoteko izpiše 1000 vrednosti. Prvo takšno spremenljivko bomo imenovali trenutni povprečni fitnes, ki za vsako generacijo izpiše povprečen fitnes populacije. Ta podatek zaradi variabilnega števila generacij ni izpisan v primeru Dawkins Weasel algoritma, ker bi bilo iskanje točke med obema metrikama v CSV datoteki prezahtevno. Druga metrika je trenutni najboljši fitnes, ki vsebuje najboljše fitnes vrednosti za vsako generacijo.

8 ANALIZA REZULTATOV

V tem poglavju bomo podatke, ki smo jih zbrali o delovanju algoritmov, izdelanih v tej nalogi, analizirali in izpostavili pomembne razlike, ki nastanejo ob različnih načinih izvajanja. V ta namen smo uporabili paket programske opreme SPSS Statistics (v nadaljevanju SPSS), medtem ko smo za izdelavo statističnih testov uporabili Mann-Whitney U test.

8.1 Priprava na analizo

Preden lahko izvedemo analizo, se moramo nanjo ustrezno pripraviti. Najprej je potrebno dobro preučiti statistični test, ki ga bomo v raziskavi uporabili. S tem bomo zagotovili pravilno uporabo testa in veljavnost ugotovitev. Zatem je treba podatke o izvajanju različnih GA prirediti tako, da jih bomo lahko uporabili v programu SPSS in v Mann-Whitney U testu. Zadnji del tega podpoglavja je namenjen predstavitvi poteka analize. Vsi problemi, ki smo jih v tej nalogi reševali s pomočjo GA, bodo v preostanku poglavja analizirani na enak način.

8.1.1 Mann-Whitney U test

Za primerjavo rezultatov bomo uporabili Mann-Whitney U test, ki je enakovreden Wilcoxonovemu testu vsote rangov. Preizkus je namenjen primerjavi razlik med dvema neodvisnima skupinama podatkov, za katere ni potrebno, da so normalno porazdeljeni. V test sta vključeni dve spremenljivki. Odvisna spremenljivka je zbirka podatkov, v kateri želimo pokazati signifikantno razliko med dvema skupinama vrednosti (npr. najboljši fitnes in generacija najboljšega fitnesa), medtem ko neodvisna spremenljivka, v našem primeru tip izvajanja, odvisno spremenljivko razdeli na skupine, ki bodo primerjane. Iz dosedanjih lastnosti testa je razvidno, da je narejen za točno takšne primere, kot je naš in je bil zato izbran za statistični test, s katerim bomo v tej nalogi dokazovali zastavljene hipoteze. [29]

Za pravilno izvedbo Mann-Whitney U testa je potrebno zadovoljiti nekaj predpostavk. Odvisna spremenljivka mora biti merjena v urejenem (angl. *ordinal*) ali intervalnem (angl. *scale*) merilu. Vse odvisne spremenljivke uporabljene v tej raziskavi so intervalnega tipa in zadovoljijo prvo predpostavko. Naslednja predpostavka zahteva, da neodvisna spremenljivka vsebuje dve kategorično neodvisni skupini. Mann-Whitney U test lahko v

programu SPSS izvedemo na več različnih načinov. Odvisno od izbranega načina mora neodvisna spremenljivka vsebovati natančno dve skupini ali pa jih lahko vsebuje poljubno število, kjer lahko naenkrat še vedno primerjamo samo dve naenkrat. Parna primerjava je povsem zadovoljiva za naš primer, kjer je potrebno vsak starostno odvisen tip izvajanja primerjati le s tradicionalnim in jih ni potrebno primerjati tudi med seboj. SPSS dodatno omogoča izključitev določenih primerkov iz testov, tako da te predpostavke ni težko zadovoljiti. Tretja predpostavka zahteva neodvisnost opazovanja, kar pomeni, da med primerki znotraj ene skupine in med primerki različnih skupin ne sme obstajati nobeno razmerje. Tipično to pomeni, da morajo biti sodelujoči v vsaki skupini različni in vsak sodelujoči lahko zastopa samo eno skupino. V prejšnjem poglavju smo videli, da se vsak tip delovanja algoritmov začne ob enakih okoliščinah, vendar z naključno začetno populacijo. Posledično so rezultati takšnih izvajanj med seboj povsem neodvisni in zadovoljujejo tretjo predpostavko. [29]

Mann-Whitney U test se uporablja za ugotavljanje, ali med dvema skupinama podatkov obstajajo razlike v distribuciji podatkov, ali če med dvema skupinama podatkov obstajajo razlike v medianah obeh skupin. Prva možnost preverja razlike v distribuciji dveh skupin in se izvede, ko distribuciji obeh skupin nista podobne oblike. Mann-Whitney U test deluje tako, da vse vrednosti odvisne spremenljivke spremeni v range, kjer imajo nižje vrednosti nižje range in višje vrednosti višje range (razporeditev v range se izvede brez razporeditve v skupine). Seštevek rangov za vsako skupino se povpreči. Če sta povprečji enaki, potem imata skupini enako distribucijo – to je tudi ničta hipoteza za Mann-Whitney U testu. V nasprotnem primeru, če sta povprečji obeh skupin signifikantno različni, sta distribuciji različni oziroma so pričakovane vrednosti rangov (angl. *mean ranks*) različne. Nadgradnja tega delovanja omogoča dokazovanje signifikantne razlike med medianama obeh skupin. Takšen način je bolj zaželen, ker lahko z njim kvantificiramo razliko med skupinama, medtem ko smo prej za podatke lahko podali le splošno informacijo o tem ali razlika med skupinami obstaja. Da se Mann-Whitney U test uporabi v ta namen je potrebno izpolniti še dodatno predpostavko – distribuciji obeh skupin morata biti podobni. Za dve distribuciji lahko rečemo, da sta podobni, če imata v grafični predstavitvi podobno obliko. Pri tem sta distribuciji lahko na različnih lokacijah. Obstajajo tudi bolj napredni matematični testi, ki lahko dajo določene informacije o podobnosti dveh distribucij, vendar se ti zelo redko uporabljajo in jih zato tudi v tej nalogi ne bomo uporabili. Ničta hipoteza ostane enaka, kot je bila v primeru neenakih distribucij, medtem ko alternativna hipoteza sedaj trdi, da mediani obeh skupin nista enaki (med njima obstaja signifikantna razlika). Čeprav za naše namene zadnje predpostavke ni potrebno zadovoljiti, ker nas zanima obstoj razlike in ne nujno tudi

njen obseg, je ta za praktično vse primere v naši raziskavi izpolnjena. Razlog za to je, kljub različnim načinom delovanja, zelo podobno napredovanje populacij iz generacije v generacijo. [30]

8.1.2 Priprava podatkov

Preden začnemo s samo analizo pridobljenih podatkov je potrebno te pripraviti oziroma jih preoblikovati v obliko primerno za uporabo v programu SPSS. Iz datotek, nastalih pri izvajanju GA, je bilo potrebno odstraniti prvo vrstico, v kateri se nahaja vhodni ukaz na podlagi katerega je nastala datoteka. V nadaljevanju smo datoteke z enakimi vhodnimi parametri (z izjemo tipa delovanja) združili v eno samo datoteko, katero smo zatem uvozili v program SPSS. Po uvozu smo ustvarili še dodatna polja. V ta polja smo vnesli tip izvajanja ter povprečne vrednosti parametrov, ki so se izpisali za vsako generacijo. Povprečne meritve zadnjih dveh smo poimenovali povprečen fitnes skozi izvajanje, ki je nastal na podlagi vrednosti parametra trenutnih povprečnih vrednosti in povprečen najboljši fitnes skozi izvajanje, katerega smo izrazili iz parametra trenutnih najboljših fitnesov.

8.1.3 Potek analize

Analizo bomo razdelili na tri dele, tako da bo vsak problem, ki smo ga reševali z GA, zajet v svojem podpoglavju. Glavni razlog za takšno razdelitev je pričakovanje različnih rezultatov za različne probleme – način delovanja, ki izboljšuje reševanje enega problema, ne bo nujno koristen tudi za druge. Vsi problemi bodo analizirani na identičen način. Ta se začne z grafičnim prikazom vseh štirih lastnosti posameznega izvajanja, ki jih bomo v tej analizi primerjali. Štirje parametri, ki nastanejo oziroma jih lahko izrazimo za vsako ponovitev istega algoritma, so:

1. Najboljši fitnes (NF) – je fitnes najboljšega osebka po končanem izvajanju vsakega algoritma.
2. Generacija najboljšega fitnesa (GNF) – je generacija, v kateri je algoritem prvič dosegel fitnes, ki je ob zaključku izvajanja pripadal najboljšemu osebku.
3. Povprečen fitnes skozi izvajanje (PF) – ne obstaja za Dawkins Weasel problem, pri katerem ne beležimo povprečnega fitnesa celotne populacije za vsako generacijo, sicer pa se izračuna kot povprečje vseh teh vrednosti.

4. Povprečen najboljši fitnes skozi izvajanje (PNF) – pri vseh problemih beležimo najboljši fitnes v vsaki generaciji. Iz teh podatkov izrazimo povprečen najboljši fitnes skozi izvajanje kot njihovo povprečje.

Najboljši fitnes je najpomembnejši parameter za dokazovanje uspešnosti posameznega GA. Pri tem uspešnost predstavlja kvaliteto končne rešitve algoritma. Generacija najboljšega fitnesa je namenjena prikazu razlik v učinkovitosti algoritmov med različnimi načini delovanja. Učinkovitost definiramo kot hitrost, s katero posamezen način delovanja pridobi najboljšo rešitev. Povprečen fitnes skozi izvajanje na prvi pogled služi prikazovanju uspešnosti algoritmov, kjer bi pričakovali višja povprečja za bolj uspešne algoritme. Izkaže se, da to pogosto ne drži in da je parameter lahko uporabljen za dokazovanje hitrosti, s katero algoritem doseže končno rešitev (četudi ta ni najboljša). Algoritmi z najboljšimi povprečnimi fitnesi v celotnih populacijah so namreč tisti, ki zelo hitro dosežejo lokalen optimum. Takšen relativno dober osebek ima zatem več časa, da se razmnoži v populacije naslednjih generacij, kot dobre rešitve drugih algoritmov, ki nastanejo pozneje v izvajanju. Posledica je veliko večji povprečni fitnes prvega algoritma, katerega večina osebkov v končni populaciji ima zelo dobre fitnes vrednosti (čeprav mogoče nobena ni tako dobra kot najboljša rešitev drugega algoritma). Interpretacija primerjave tega parametra ima še eno dodatno izjemo – primere, kjer se algoritem ne izvaja vedno do iste generacije. V našem primeru je to A primer Dowkins Weasel problema. Ker se v tem primeru izvajanje konča takoj, ko je dosežen globalen optimum, imajo najhitrejši algoritmi zelo nizke povprečne fitnes vrednosti. Ker je bila najboljša rešitev najdena tako hitro, je zelo majhna množica osebkov znotraj populacije dosegla boljše fitnese, medtem ko ima velika večina osebkov še zelo slabe in posledično je povprečje celotne populacije zelo slabo. Parameter povprečja najboljših fitnesov skozi izvajanje trpi za enakimi problemi, čeprav v nekoliko milejši obliki. Iz teh razlogov bomo zadnja dva parametra smatrali kot sekundarna pokazatelja razlik med različnimi načini delovanja GA. To pomeni, da njuni rezultati niso pomembni, če ne nastanejo v navezi z enim izmed prvih dveh metrik. Kako interpretirati njihove rezultate pa je odvisno od okoliščin oziroma v kakšnem razmerju so vrednosti med primerjanima tipoma delovanja.

Prvi kazalnik okoliščin, ki smo ga uporabili v analizi, so histogrami. Za vsak parameter posebej smo ustvarili svoj graf, ki je razdeljen na posamezne tipe delovanja. Številne histograme smo vključili v ta dokument, ker so preprosti za razumevanje in ker nosijo številne informacije s pomočjo katerih lažje razumemo delovanje posameznih algoritmov. Za natančnejše razumevanje rezultatov Mann-Whitney U testa lahko s pomočjo teh

histogramov tudi primerjamo podobnosti distribucij med različnimi skupinami podatkov. Kot del analize smo izvedli tudi teste opisne statistike. Ti lahko zelo pomagajo pri razumevanju podatkov, še posebej, kjer je med histogrami težko opaziti razlike. V takšnih primerih smo namesto grafov vključili tabele z opisnimi podatki. Vsi testi, ki smo jih izvajali, so ločeni glede na tip izvajanja, zato je tu seznam vseh možnih tipov, začenši s cifro, ki bo v nadaljevanju predstavljala posamezno delovanje:

0 - Tradicionalno oziroma konvencionalno delovanje GA.
1 - Tip 0 s tako spremenjeno mutacijo, da imajo starejši geni večjo verjetnost izbora v operaciji mutacije (podrobnosti v poglavju 5.2).
2 - Tip 0 s tako spremenjeno mutacijo, da imajo mlajši geni večjo verjetnost izbora v operaciji mutacije (podrobnosti v poglavju 5.2).
3 - Tip 0 s tako spremenjenim križanjem, da imajo starejši geni večjo verjetnost izbora v operaciji križanja (podrobnosti v poglavjih 5.1 in 5.1.1).
4 - Tip 0 s tako spremenjenim križanjem, da imajo mlajši geni večjo verjetnost izbora v operaciji križanja (podrobnosti v poglavjih 5.1 in 5.1.1).
5 - Tip 0 s tako spremenjenim križanjem, da imajo geni, katerih starost je bližje povprečni pričakovani starosti, večjo verjetnost izbora v operaciji križanja (podrobnosti v poglavjih 5.1 in 5.1.2).
6 - Tip 0 s tako spremenjenim križanjem, da imajo geni, katerih starost je bolj oddaljena od povprečne pričakovane starosti (ekstremi), večjo verjetnost izbora v operaciji križanja (podrobnosti v poglavjih 5.1 in 5.1.2).

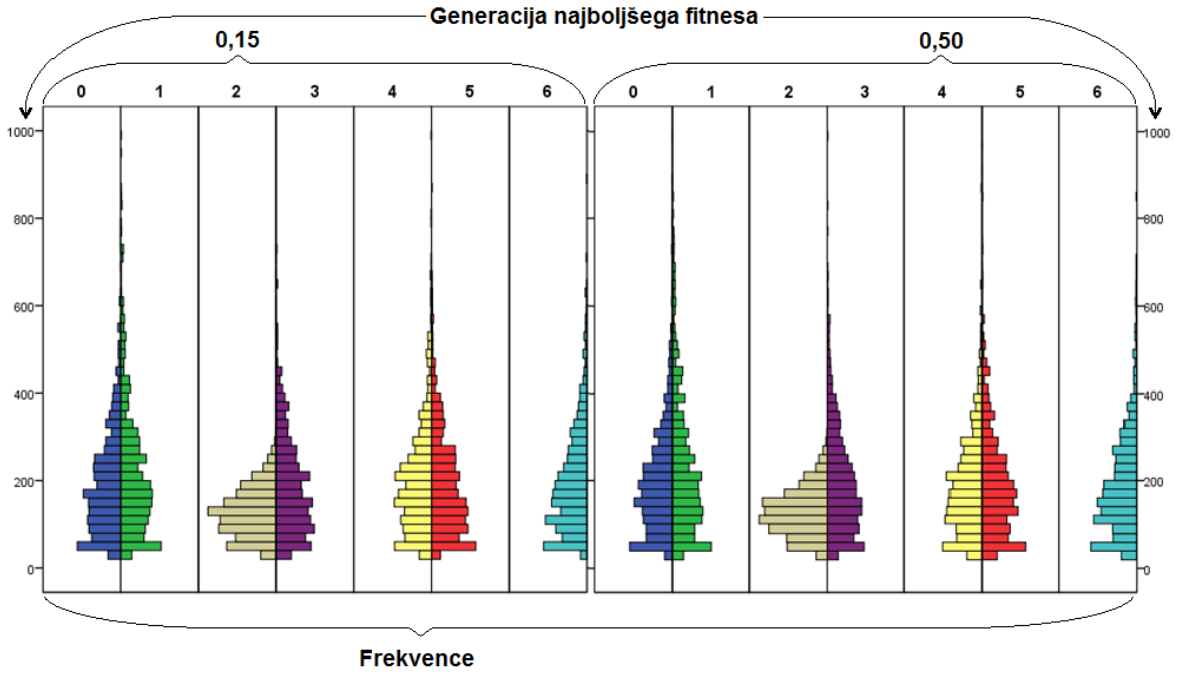
Nazadnje smo za vsako kombinacijo med tipom 0 in preostalimi tipi delovanja izvedli Mann-Whitney U test. Rezultate tega testa in opisne statistike bomo kot primer vključili samo pri prvem primeru analize podatkov, čeprav smo jih izdelali in uporabili pri vseh. Za vse algoritme bomo na podlagi Mann-Whitney U testa podali samo ali je starostno odvisen način delovanja za posamezen parameter signifikantno boljši, slabši ali med njima ni razlike. Za to krajšo obliko predstavitve rezultatov smo se odločili zaradi obsega vseh izpisov testov, ki bi jih sicer morali vključiti v to nalogo. Vsi ti rezultati pridobljeni s programom SPSS in podatkovne tabele, iz katerih so nastali, smo zato dodali na zgoščenko, ki je vključena poleg naloge kot priloga. Zatem, ko nam Mann-Whitney U test poda rezultate o signifikantnosti razlik med posameznimi parametri različnih tipov delovanja, nam preostane samo še interpretacija razlogov zanje in še pomembneje njihovih posledicah.

Za konec še nekaj besed o parametrih, ki smo jih pridobili pri izvajanju algoritmov, vendar jih nismo uporabili pri analizi. Prvi takšni parametri so povprečne starosti genov v populaciji,

genov ob mutaciji in genov ob križanju. Te vrednosti so namenjeni predvsem spremljanju delovanja algoritmov in omogočajo uporabniku vpogled v postopek delovanja. Isto velja tudi za parametra, ki podajata povprečni fitnes populacije in najboljši fitnes v populaciji za vsako posamezno generacijo. Noben od teh parametrov sami po sebi ne nakazujejo nobenih prednosti ali pomanjkljivosti algoritmov pri iskanju rešitev in zato niso vključeni v analizo.

8.2 Dawkins Weasel

Začeli bomo z najpreprostejšim problemom v tej nalogi - Dawkins Weasel problem A z iskanim nizom dolgim 77 znakov. Primer je bil dovolj enostaven, da so ga vsi načini izvajanja pri obeh uporabljenih odstotkih starostne razlike rešili pravilno znotraj 1000 generacij v vseh ponovitvah z le eno izjemo. To pomeni, da med različnimi načini izvajanja ne bo razlik v parametru najboljšega fitnesa, zato bomo tega tokrat izjemoma izpustili in nadaljevali z generacijami najboljših fitnesov. Vsi histogrami v nadaljevanju bodo zgrajeni po enakem principu, zato bomo tu najprej pogledali njegovo strukturo. Naslednja slika (Slika 8.1) prikazuje dva histograma. Levi graf prikazuje izvajanje Dawkins Weasel A problema ob 0,15 vrednosti odstotka starostne razlike, medtem ko desni prikazuje rezultate za odstotek starostne razlike 0,5. Na vrhu slike je izpisan parameter, ki je prikazan v diagramih. Njegove vrednosti so zapisane v ordinatni osi, medtem ko abscisa vsebuje število ponovitev določene vrednosti opazovanega parametra. V histogramu torej višina stolpca predstavlja pogostost pojava vrednosti na y-osi. Vsak graf je naknadno razdeljen v sedem pasov, kjer vsak predstavlja en tip izvajanja. Vsak drugi pas je zrcaljen za lažjo primerjavo s sosednjim tipom izvajanja. Tip izvajanja 0 je konvencionalen način izvajanja GA in predstavlja standard delovanja, s katerim bodo primerjani ostali tipi izvajanja. Tipi izvajanja 0, 1 in 2 niso odvisni od odstotka starostne razlike, zato za te načine delovanja pričakujemo, da bodo zelo podobni na obeh grafih. Dvojno izvajanje nam pokaže konsistentnost delovanja algoritmov, ki je ključnega pomena za zagotavljanje zanesljivosti meritev. Če se sedaj vrnemo na sliko, hitro opazimo, da se tip izvajanja 2 vidno razlikuje od vseh ostalih v obeh diagramih. Njegove vrednosti ne segajo niti približno tako visoko kot tip 0, ampak so zgoščene nižje na lestvici generacij najboljšega fitnesa. Iz tega lahko že zelo močno sumimo, da je takšen tip izvajanja veliko hitreje dosegel najboljše fitnes vrednosti.



Slika 8.1: Grafa generacij najboljših fitnes vrednosti za Dawkins Weasel A problem.

Diagramom sledi opisna statistika, ki je prikazana v naslednji tabeli (Tabela 8.1). V njej lahko vidimo, da so vsi tipi izvajanja res vedno dosegli najboljšo možno fitnes vrednost, z eno izjemo v primeru delovanja 1. Takoj zatem lahko potrdimo, kar smo opazili že na diagramih - tip izvajanja 2 ima občutno manjšo povprečno vrednost generacije najboljšega fitnesa proti vsem ostalim tipom. Poleg tega ima tudi nižje povprečje najboljšega fitnesa skozi izvajanje, medtem ko ima tip izvajanja 1, ravno obratne razlike v primerjavi s tipom 0. Ostali tipi izvajanja so v vseh merjenih parametrih dosegli zelo podobne rezultate tradicionalnemu izvajanju.

Tabela 8.1: Opisna statistika za Dawkins Weasel A problem.

TipIzvajanja		Minimum	Maximum	Mean	Std. Deviation
0	NajboljsiFitnes	77	77	77,00	,000
	GeneracijaNajboljsegaFitnesa	37	782	192,55	122,497
	PovprecenNajboljsiFitnesSkoziIzvajanje	43,88	73,70	65,3452	7,48042
1	NajboljsiFitnes	76	77	77,00	,045
	GeneracijaNajboljsegaFitnesa	36	996	226,54	159,804
	PovprecenNajboljsiFitnesSkoziIzvajanje	44,53	73,75	66,2527	7,42459
2	NajboljsiFitnes	77	77	77,00	,000
	GeneracijaNajboljsegaFitnesa	37	303	126,42	56,341
	PovprecenNajboljsiFitnesSkoziIzvajanje	44,05	72,06	62,9190	6,67883
3	NajboljsiFitnes	77	77	77,00	,000

	GeneracijaNajboljsegaFitnesa	37	736	192,14	120,008
	PovprecenNajboljsiFitnesSkoziIzvajanje	43,28	73,74	65,4913	7,25768
	NajboljsiFitnes	77	77	77,00	,000
4	GeneracijaNajboljsegaFitnesa	36	980	201,48	128,098
	PovprecenNajboljsiFitnesSkoziIzvajanje	44,43	73,74	65,8677	7,19041
	NajboljsiFitnes	77	77	77,00	,000
5	GeneracijaNajboljsegaFitnesa	37	865	192,65	120,960
	PovprecenNajboljsiFitnesSkoziIzvajanje	44,36	73,77	65,5163	7,11899
	NajboljsiFitnes	77	77	77,00	,000
6	GeneracijaNajboljsegaFitnesa	37	880	196,60	122,982
	PovprecenNajboljsiFitnesSkoziIzvajanje	44,41	73,71	65,7672	7,03078

Vse kar je še potrebno storiti za Dawkins Weasel A problem so Mann-Whitney U testi. Rezultat vsake primerjave s tem testom sta dve tabeli. Prva tabela (Tabela 8.2) prikazuje povprečen rang in seštevek vseh rangov, ki so bili dodeljeni posamezni skupini. Druga tabela (Tabela 8.3) vsebuje sam podatek o signifikantnosti razlik med dvema tipoma izvajanja - *Asymp. Sig. (2-tailed)*, navadno imenovan kar p-vrednost, je rezultat dvostranskega preizkusa. Če je ta vrednost večja kot stopnja značilnosti, potem ničte hipoteze ne moremo zavreči, sicer pa lahko sprejmemo alternativno hipotezo. Stopnja značilnosti je v naših testih enaka 0,05 in zato te trditve sprejmemo ob 5 % tveganju. Mann-Whitney U test med tipoma izvajanja 0 in 1 nam pokaže, da obstaja signifikantna razlika med generacijo najboljšega fitnesa in povprečnim najboljšim fitnesom skozi izvajanje (ker je p-vrednost manjša od 0,05). Sedaj, ko vemo, da razlika obstaja, lahko v prvi tabeli razberemo, katera skupina ima signifikantno boljše rezultate za dane parametre. Povprečje rangov za generacijo najboljšega fitnesa je manjša v skupini 0. Ker so v tem parametru boljše nižje vrednosti, je tip izvajanja 1 signifikantno počasnejši pri iskanju optimalne rešitve od tipa 0. Parameter povprečnega najboljšega fitnesa skozi izvajanje je, kot smo že omenili, zelo zavedljiv. V tem primeru nakazuje na boljši končni fitnes tipa izvajanja 1, za kar pa iz povprečja najboljših fitnes vrednosti v prejšnji tabeli (Tabela 8.1) vemo, da ne drži.

Tabela 8.2: Prvi del Mann-Whitney U testa za tip delovanja 1 v Dawkins Weasel A problemu.

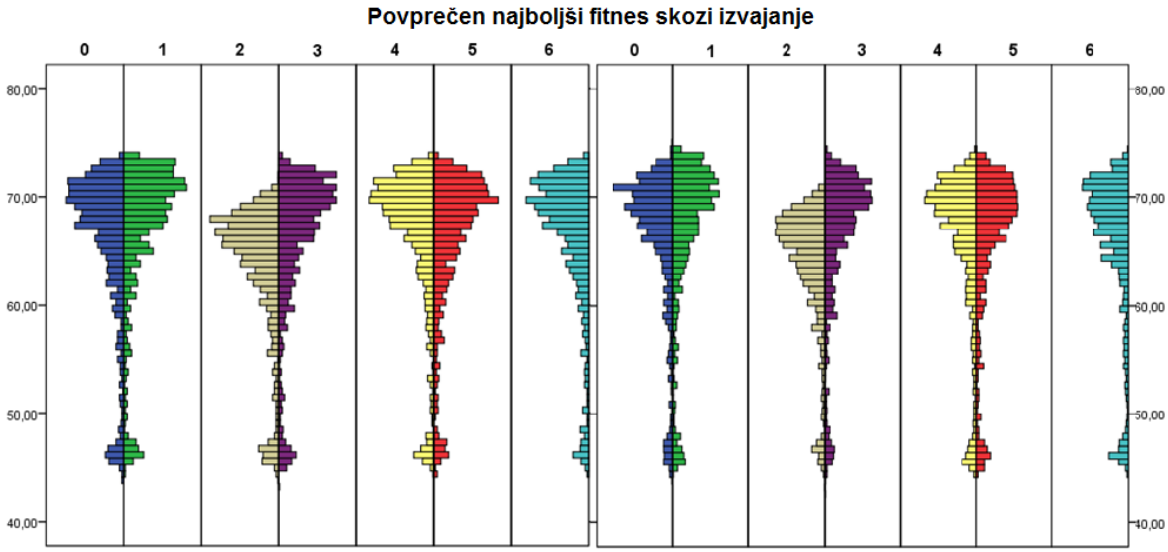
	TipIzvajanja	Mean Rank	Sum of Ranks
NajboljsiFitnes	0	1001,50	1001500,00
	1	999,50	999500,00
GeneracijaNajboljsegaFitnesa	0	950,41	950409,50
	1	1050,59	1050590,50
PovprecenNajboljsiFitnesSkoziIzvajanje	0	945,18	945180,50
	1	1055,82	1055819,50

Tabela 8.3 Drugi del Mann-Whitney U testa za tip delovanja 1 v Dawkins Weasel A problemu.

	NajboljsiFitnes	GeneracijaNajboljsegaFitnesa	PovprecenNajboljsiFitnesSkoziIzvajanje
Mann-Whitney U	499000,000	449909,500	444680,500
Wilcoxon W	999500,000	950409,500	945180,500
Z	-1,415	-3,879	-4,284
Asymp. Sig. (2-tailed)	,157	,000	,000

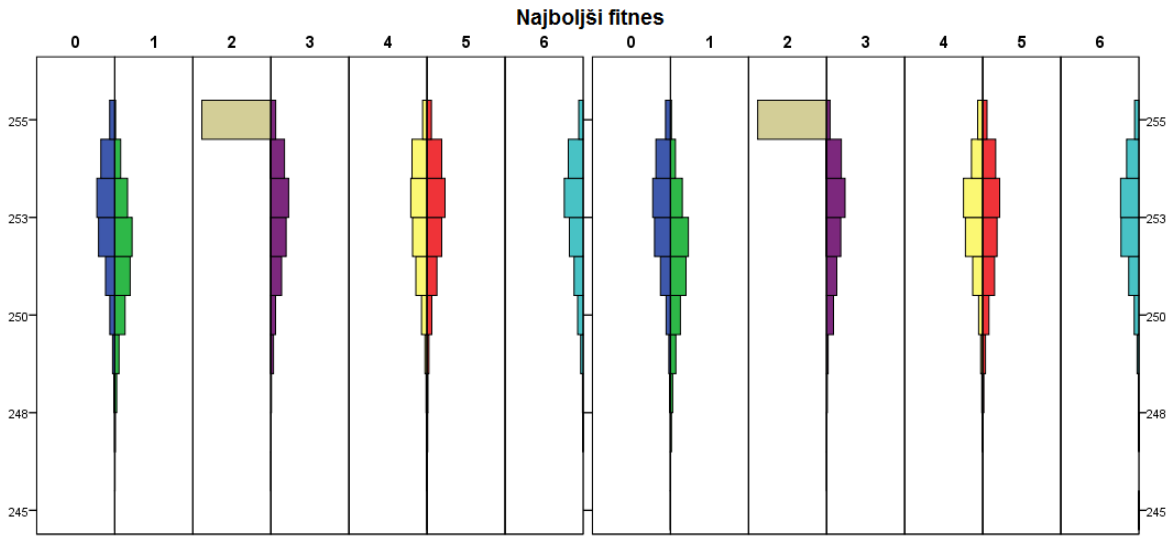
Razlog za višje vrednosti povprečnega najboljšega fitnesa skozi izvajanje najdemo na naslednjih diagramih (Slika 8.2). Histograma za tip izvajanja 1 sta v primeru obeh odstotkov starostne razlike veliko širša na vrhu grafov, kar pomeni, da je veliko število generacij imelo zelo visoke fitnes vrednosti. Razlog za več visokih fitnes vrednosti je počasnejše iskanje optimalne rešitve, kar omogoča trenutnim najboljšim rešitvam, da se razmnožijo v naslednjih populacijah. Signifikantna razlika v tem parametru med tipom izvajanja 0 in 1, zato ne nakazuje boljših rešitev, temveč zgolj potrdi že dokazano počasnejše iskanje teh. Na istem diagramu opazimo tudi redkost zelo dobrih fitnes vrednosti v 2. tipu izvajanja. To smo posredno opazili že v tabeli z opisno statistiko, vendar šele iz diagrama vidimo razlog za nižjo povprečno vrednost tega parametra. Vemo, da sta bila tipa izvajanja 0 in 2 v tem primeru enako uspešna pri odkrivanju najboljše rešitve, vendar na spodnjih grafih zelo dobri fitnesi skoraj niso zaznani. To pomeni, da algoritmi s takšnim delovanjem zelo hitro preidejo iz fitnesa okrog 70 do končne vrednosti 77. Zato imajo malo visokih fitnes vrednosti med izvajanjem in posledično nizek povprečen najboljši fitnes skozi izvajanje. Mann-Whitney U test potrdi obe naši domnevi o tipu 2. Njegova generacija najboljšega fitnesa je signifikantno manjša od tradicionalnega delovanja GA. Isto velja tudi za povprečen najboljši fitnes skozi izvajanje, kar kot smo že dejali, samo potrdi domnevo o hitrejšem iskanju optimalne rešitve. Nobeden od preostalih tipov izvajanja ni, v katerikoli merjeni spremenljivki, pokazal signifikantne razlike v primerjavi z delovanjem tipa 0. Vsi ti rezultati Dawkins Weasel A

problema, ki smo jih opisali, držijo pri uporabi obeh odstotkov starostne razlike. Vsi pomembni rezultati izvajanja Dawkins Weasel problema so strnjeni na koncu tega podpoglavja (Tabela 8.4).



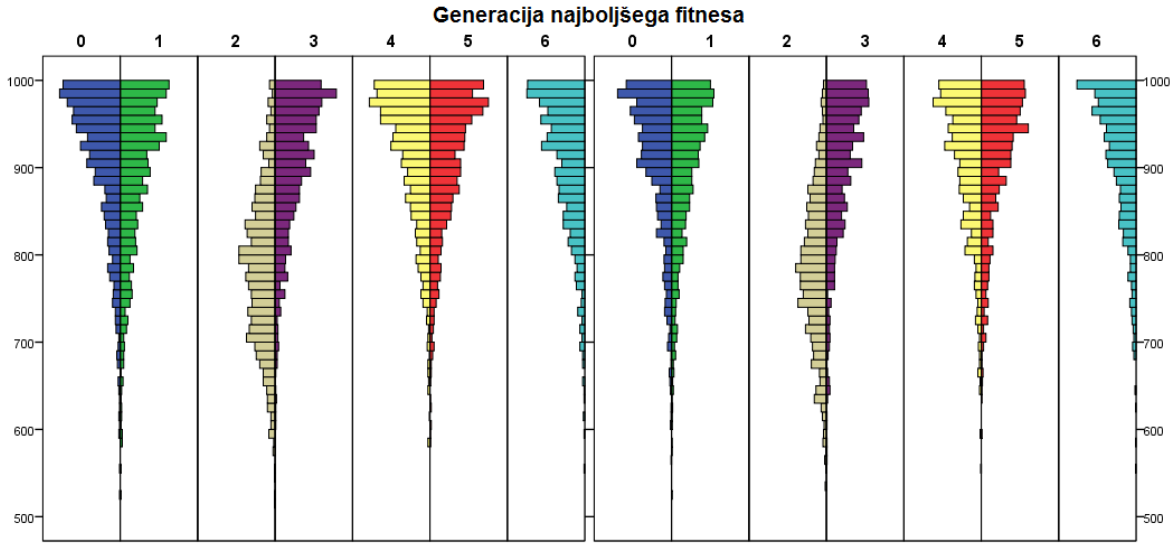
Slika 8.2: Grafa povprečnega najboljšega fitnesa skozi izvajanje za Dawkins Weasel A problem.

Nadaljujemo z B primerom Dawkins Weasel problema, kjer je iskalni niz dolg 255 znakov. Že na prvih grafih (Slika 8.3) opazimo očitni razliki v distribuciji najboljših fitnesov. Distribucija tipa 1 je pomaknjena za eno vrednost navzdol, kar se v Mann-Whitney U testu pokaže kot signifikantna razlika. Drugi histogram, ki močno odstopa od konvencionalnega, je histogram za tip izvajanja 2. Čeprav tokrat ni našel optimalne rešitve v sto odstotkih primerov, je število izjem tako majhno, da na grafih sploh niso zaznavne. Mann-Whitney U test tu seveda pokaže signifikantno boljše najdene rešitve za tip 2. Oba rezultata veljata za obe vrednosti odstotka starostne razlike. Edina druga signifikantna razlika, ki ni opazna na grafih ali z opisno statistiko, so slabši rezultati tipa 5 v primeru 0,5 odstotka starostne razlike.



Slika 8.3: Grafa najboljšega fitnesa za Dawkins Weasel B problem.

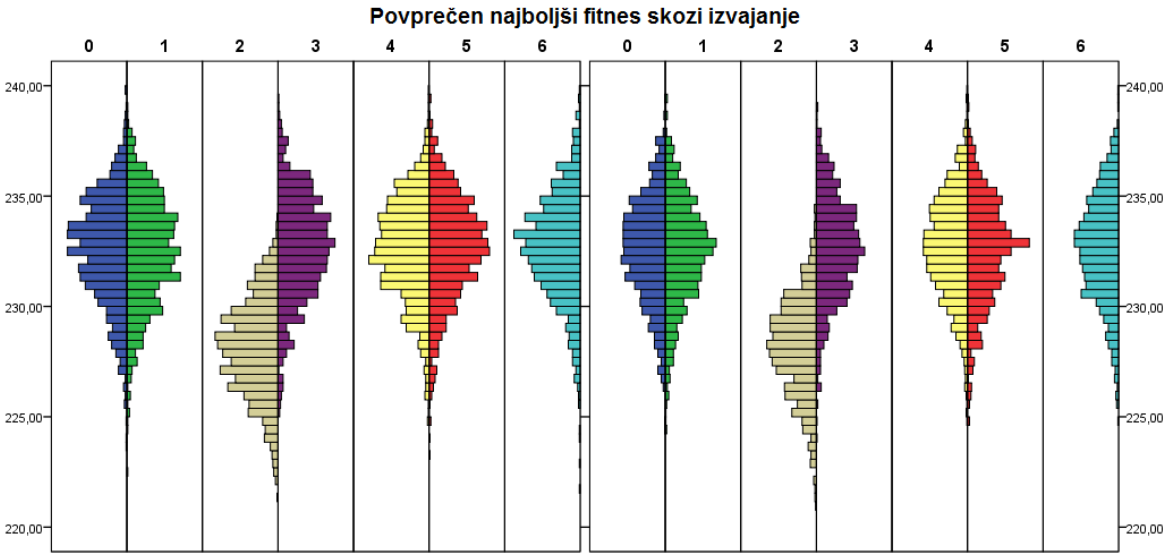
Sledi grafični prikaz generacije najboljšega fitnesa za Dawkins Weasel B problem (Slika 8.4). Tako kot v problemu A je tip delovanja 2 ponovno očitno hitrejši pri iskanju optimalnih rešitev, kar je lepo vidno tudi na grafih. Mann-Whitney U test to domnevo potrdi. Druga in manj očitna signifikantna razlika, čeprav še vedno lepo vidna na histogramu, pokaže, da je tudi tip izvajanja 1 hitrejši od tipa 0. Obe ugotovitvi držita za oba odstotka starostne razlike.



Slika 8.4: Grafa generacij najboljših fitnes vrednosti za Dawkins Weasel B problem.

Sklepni test in histogrami (Slika 8.5) pokažejo, da je obnašanje vrednosti povprečnega najboljšega fitnesa skozi izvajanje za tip 2 enako, kot je bilo v A primeru – vrednosti so, kot

posledica hitrejšega pridobivanja optimalnih rešitev v primeru tipa 2, veliko nižje. Drugih signifikantnih razlik med konvencionalnim in starostno odvisnimi načini delovanja ni.



Slika 8.5: Grafa povprečnega najboljšega fitnesa skozi izvajanje za Dawkins Weasel B problem.

Naslednja tabela (Tabela 8.4) združuje vse rezultate Mann-Whitney U testa za Dawkins Weasel problem. V stolpcih so glede na problem (A in B) ter odstotek starostne razlike (0,15 in 0,5) prikazane signifikantne razlike v parametrih med konvencionalnim delovanjem GA in starostno odvisnimi tipi delovanja (1-6). Primerjani parametri so po vrsti: najboljši fitnes (NF), generacija najboljšega fitnesa (GNF), povprečen fitnes skozi izvajanje (PF) in povprečen najboljši fitnes skozi izvajanje (PNF). Povprečnega fitnesa skozi izvajanje v primeru Dawkins Weasel ne beležimo, zato je tretji stolpec obarvan sivo. Tipi izvajanja, ki so imeli signifikantno boljše rezultate v določenem parametru so označeni s kljukico, medtem ko so tisti, ki so bili znatno slabši, označeni s križcem. V primeru, da parameter najboljšega fitnesa vsebuje katerega od obeh znakov, zavrnemo ničto hipotezo A in sprejmemo alternativno, ki pravi, da dotični starostno odvisen izbor genov signifikantno spremeni uspešnost genetskega algoritma. Isto storimo v primeru parametra generacije najboljšega fitnesa, le ta tokrat zavrnemo ničto hipotezo B in sprejmemo njeno alternativo, ki pravi, da dotični starostno odvisen izbor genov signifikantno spremeni učinkovitost genetskega algoritma. Za preostala parametra smo že dejali, da sta sekundarna in lahko zgolj dodatno podkrepita ugotovitve, pridobljene iz primarnih parametrov. Sekundarna parametra sta tudi težavna za prikazovanje v tabeli, ker je glede na okoliščine, včasih boljša višja vrednost in včasih nižja. Zato smo se odločili, da bomo te parametre prikazovali na način, odvisen od parametra najboljšega fitnesa. To pomeni, da bomo v primerih, kjer je

boljši višji fitnes, označevali signifikantno večje vrednosti sekundarnih parametrov s kljukicami (in pomembno nižje s križci), čeprav lahko ta razlika govori o boljših ali slabših rezultatih kateregakoli od obeh primarnih parametrov. Obratno bomo s kljukico označili nižjo vrednost teh parametrov, kjer je manjši fitnes boljši. Za sekundarne parametre je zato pri analizi posameznih parametrov pomembno poznati okoliščine, ki so opisane pri obravnavi vsakega parametra.

Tabela 8.4: Rezultati analize za Dawkins Weasel.

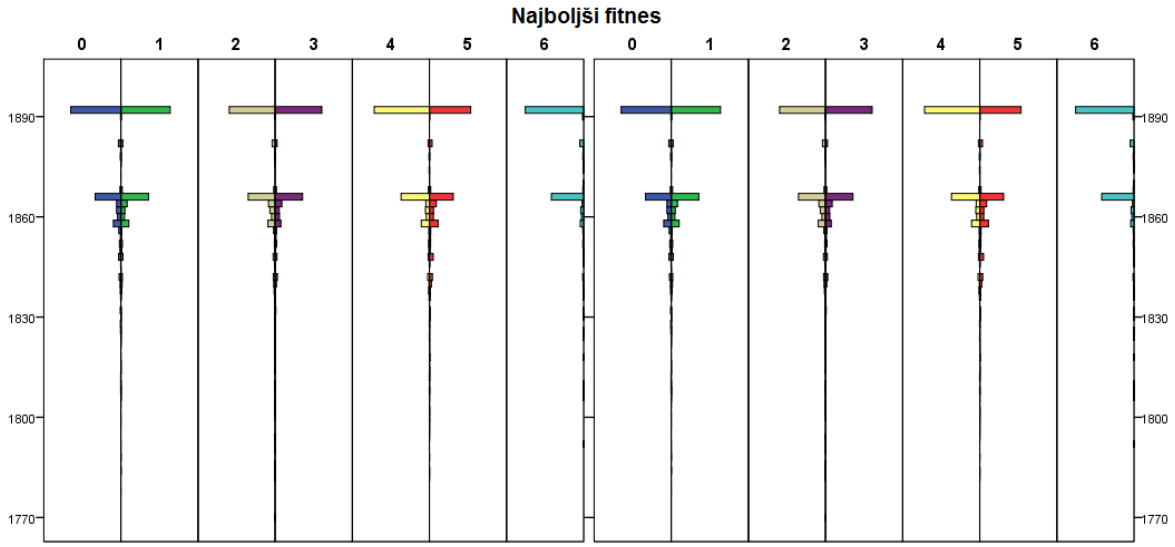
	A								B							
	0,15				0,5				0,15				0,5			
	NF	GNF	PF	PNF	NF	GNF	PF	PNF	NF	GNF	PF	PNF	NF	GNF	PF	PNF
1		×		✓		×		✓	×	✓			×	✓		
2		✓		×		✓		×	✓	✓		×	✓	✓		×
3																
4																
5													×			
6																

Iz pregleda vseh prednosti in pomanjkljivosti, ki jih starostno odvisna mutacija (tip 1 in 2) in križanje (tipi 3-6) vnesejo v GA, najprej opazimo razliko med dvema operacijama. Spremembe križanja, ki smo jih uvedli v delovanje, praktično niso imele pomembnega učinka na rezultate delovanja, medtem ko so spremembe v mutacijah povzročile drastične razlike. Mutacija, ki daje pri izbiri prednost starejšim genom (tip 1), je bila v A problemu počasnejša od tradicionalnega izvajanja, medtem ko je pri zahtevnejšem B problemu občutno hitreje našla dobre rešitve, čeprav se je izkazalo, da so te nekoliko slabše. Na drugi strani se privilegiranje mlajših genov v operaciji mutacije (tip 2) za Dawkins Weasel problem vedno izplača. Takšno delovanje hitreje najde dobre rešitve, ki so tudi občutno boljše od tradicionalnega delovanja GA. Zadnje pomembno potrdilo, ki nam ga tabela rezultatov pokaže, je ponovljivost prvih dveh tipov izvajanja znotraj problema z enakim iskalnim nizom. Različen odstotek starostne razlike ne vpliva na operacije mutacije in identični rezultati znotraj problemov A in B govorijo o konsistentnosti izvajanja GA in meritev njihovih rezultatov.

8.3 Problem nahrbtnika

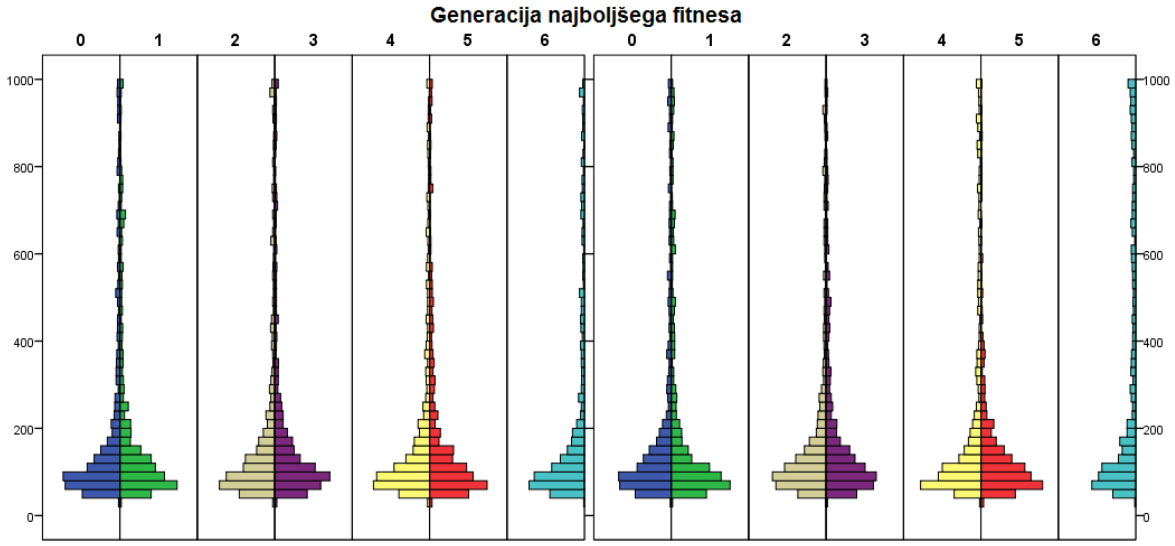
Problem nahrbtnika smo podobno kot Dawkins Weasel problem reševali za dva različna nabora vhodnih podatkov. Problem A je z 250 vhodnimi predmeti, med katerimi lahko

algoritem izbira, preprostejši izmed obeh, zato se bomo najprej posvetili njegovim rezultatom. Prvi diagram (Slika 8.6) predstavlja pogostost najboljših fitnes vrednosti za odstotek starostne razlike 0,15 (levo) in 0,5 (desno). Prva posebnost, ki jo opazimo je neobičajna distribucija. Ta nam pove, da večina ponovitev algoritma kot optimalno rešitev najde dva različna osebka. Osebek z nižjo fitnes vrednostjo (nižje na histogramu) je zelo dober lokalni optimum, preko katerega se ima veliko število ponovitev problem prebiti. Vsi načini delovanja imajo zelo podobno distribucijo, kar pomeni, da nobeden ne odpravi te pomanjkljivosti. Edina opazna razlika med njimi je v primeru tipa 5, kjer lahko opazimo nekoliko nižjo frekvenco ponavljanja teh dveh osebkov. Mann-Whitney U test potrdi našo domnevo o slabšem delovanju tega algoritma za oba odstotka starostne razlike. GA je dosegel signifikantno boljše rezultate ob uporabi tipov delovanja 4 in 6, vendar le v primeru, ko je odstotek starostne razlike znašal 0,5.



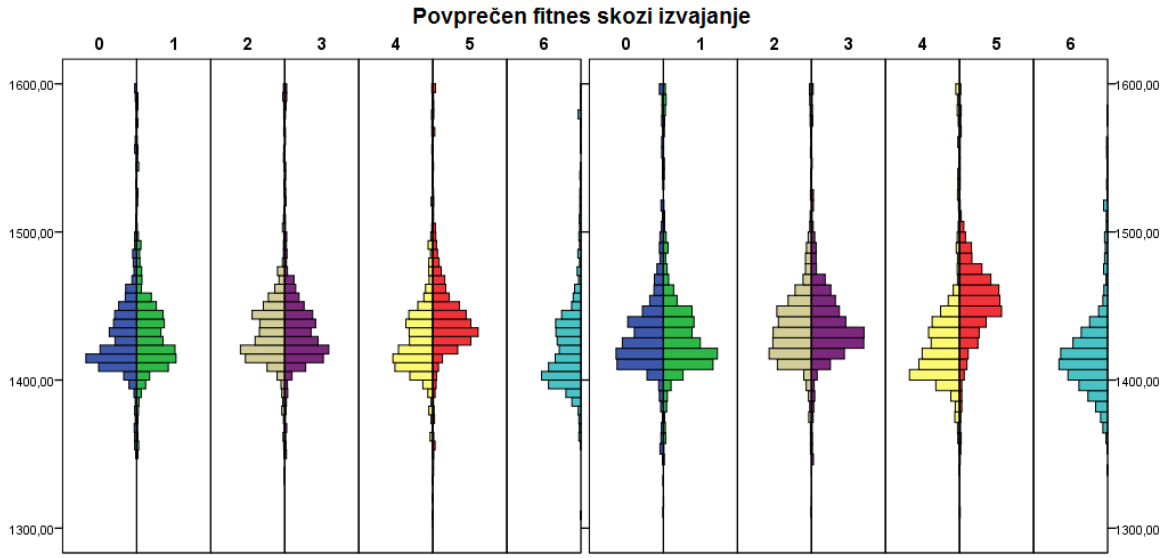
Slika 8.6: Grafa najboljšega fitnesa za problem nahrbtnika, primer A.

Distribucije vseh tipov delovanja so si pri hitrosti iskanja rešitev zelo podobne (Slika 8.7). Vizualno je tu zelo težko oceniti razlike, vendar lahko vseeno vidimo, da je na desnem grafu tip izvajanja 5 nekoliko širši na dnu in skorajda nima visokih vrednosti. Obratno ima tip izvajanja 6 občutno manj rešitev, najdenih v prvih dvesto generacijah, in vidno več v zadnjih dvesto. Sklepni test potrdi naše domneve o signifikantno hitrejšem iskanju najboljših osebkov s tipom izvajanja 5 ter počasnem delovanju tipa 6 in dodatno tudi tipa 4. Vse tri razlike se pojavijo samo pri odstotku starostne razlike 0,5.



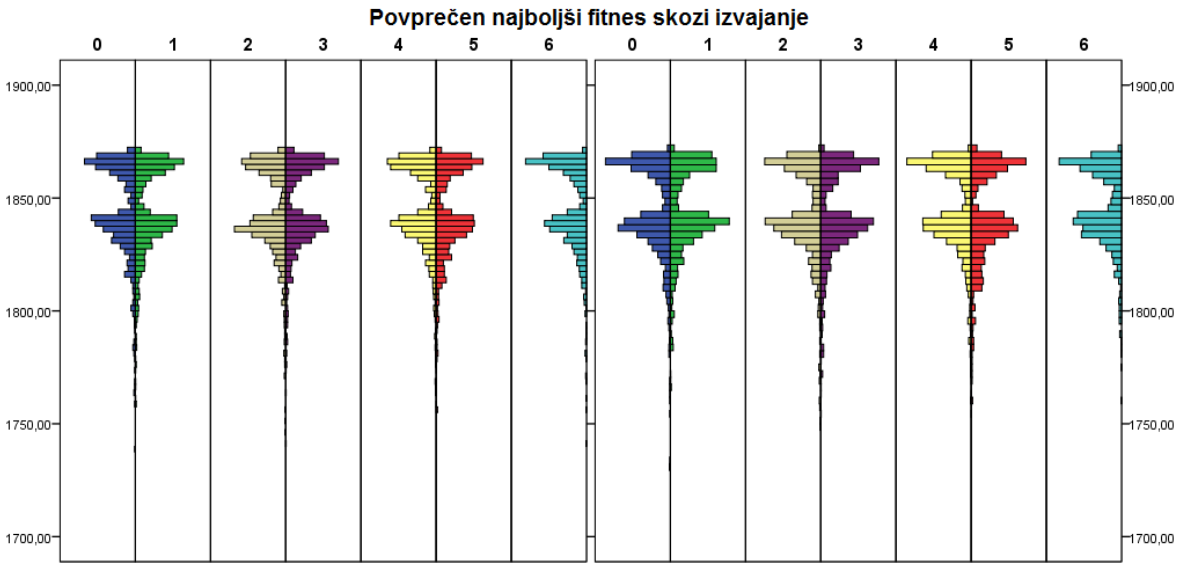
Slika 8.7: Grafa generacij najboljših fitnes vrednosti za problem nahrbtnika, primer A.

Na grafih povprečnega fitnesa skozi izvajanje (Slika 8.8) imamo prvič opravka z distribucijami, ki si niso vse med seboj podobne. Zato je mogoče na prvi pogled težje razumeti, kakšen pomen nosi vsak histogram. Izkaže se, da imata pri obeh odstotkih starostne razlike boljši povprečni fitnes skozi izvajanje tipa 2 in 5. Če sedaj pogledamo na grafa, opazimo, da sta ta dva tipa izvajanja proti tradicionalnemu delovanju postavljena nekoliko višje, kar tudi nakazuje na večje povprečje. Po enakem razmišljanju lahko vidimo, da bosta tipa izvajanja 4 in 6 najverjetneje signifikantno slabša, ker se njuna histograma začneta nižje. Mann-Whitney U test pokaže, da je tip 6 slabši pri obeh odstotkih starostne razlike, medtem ko je tip 4 občutno slabši samo na desnem grafu.



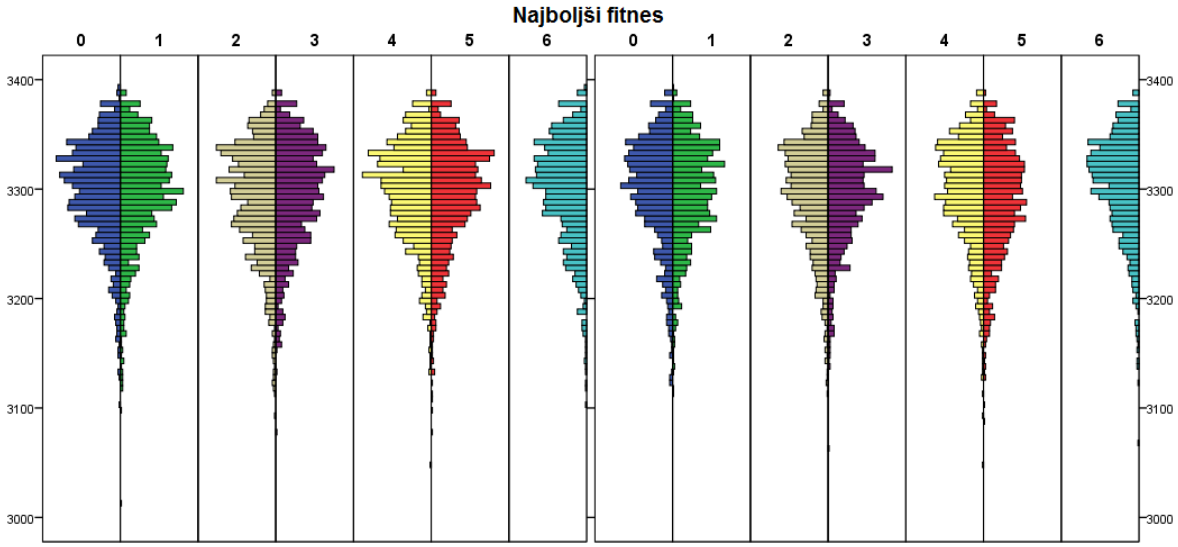
Slika 8.8: Grafa povprečnega fitnesa skozi izvajanje za problem nahrbtnika, primer A.

Histogrami za povprečen najboljši fitnes skozi izvajanje (Slika 8.9) so si med seboj zelo podobni in je težko iz njih razbrati kakršnekoli informacije o razlikah med njimi. Kar pa lahko takoj opazimo, je njihova podobnost z grafi najboljšega fitnesa. Iz tega lahko razberemo, kako tesno sta ta dva parametra v tem primeru povezana. Pomembno večje povprečje se v primerjavi s konvencionalnim izvajanjem pokaže v tipu 3 ob večjem odstotku starostne razlike. Signifikantno slabše delovanje najdemo pri tipu izvajanja 5, ki je tudi edini način izvajanja s pomembno slabšimi končnimi rešitvami. Tip 5 ima oba parametra slabša od primerjanega delovanja, ne glede na odstotek starostne razlike.



Slika 8.9: Grafa povprečnega najboljšega fitnesa skozi izvajanje za problem nahrbtnika, primer A.

Nadaljujemo z obsežnejšim in zahtevnejšim problemom nahrbtnika, ki vsebuje 450 predmetov za razvrstitev. Že takoj na prvem diagramu (Slika 8.10) opazimo, kako to vpliva na število različnih končnih rešitev, ki so jih GA našli napram primeru A, kjer sta bila samo dva končna osebka pogosta. Histogrami so si ponovno tako podobni med seboj, da je iz njih težko karkoli sklepati, zato si bomo za te primere pogledali opisno statistiko.



Slika 8.10: Grafa najboljšega fitnesa za problem nahrbtnika, primer B.

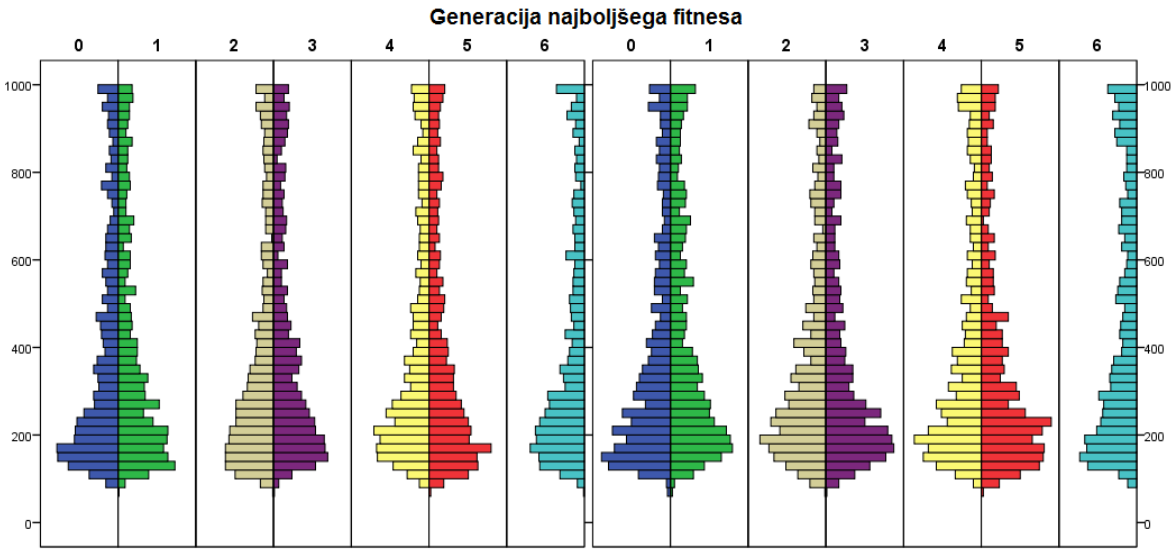
Povprečne vrednosti najboljših fitnesov (Tabela 8.5) so v tem primeru veliko bolj primerne za prikaz razlik med rešitvami različnih tipov izvajanja. Tabela prikazuje samo vrednosti, kjer je odstotek starostne razlike znašal 0,5. Največjo povprečno vrednost končnih rešitev ima tip izvajanja 6, medtem ko imata najslabšo tipa 2 in 5. Mann-Whitney U test potrdi te tri načine izvajanja kot signifikantno različne od konvencionalnega. Pri odstotku starostne razlike 0,15 je edina pomembna razlika pri tipu 2, ki se ponovno izkaže kot slabši. Pri tem je zanimivo, da je tip 1 edini, ki mu je uspelo najti rešitev s fitnesom preko 3390.

Tabela 8.5: Najboljši fitnes za problem nahrbtnika, primer B ob odstotku starostne razlike 0,5.

TipIzvajanja		Minimum	Maximum	Mean	Std. Deviation
0	NajboljsiFitnes	3123	3389	3297,86	49,883
1	NajboljsiFitnes	3115	3391	3297,38	47,871
2	NajboljsiFitnes	3112	3389	3292,73	49,667
3	NajboljsiFitnes	3065	3389	3295,67	47,215
4	NajboljsiFitnes	3048	3389	3296,09	49,386
5	NajboljsiFitnes	3086	3389	3288,14	51,310
6	NajboljsiFitnes	3070	3389	3302,81	46,419

Generacija najboljšega fitnesa (Slika 8.11) se ponovno razlikuje od primera A, kjer so bile vrednosti veliko bolj koncentrirane na dnu histogramov. Zahtevnejši problem zahteva več časa za odkritje najboljše rešitve in zato algoritmi odkrivajo boljše osebke do zadnje generacije. Za delovanje z odstotkom starostne razlike 0,5 Mann-Whitney U test pokaže, da je tip izvajanja 5 signifikantno hitrejši od ostalih, medtem ko sta tipa 1 in 6 občutno počasnejša pri pridobivanju dobrih osebkov, kar lahko na histogramih vidimo kot širše

vrhove. Pri manjšem odstotku starostne razlike ni odstopanja od konvencionalnega delovanja.



Slika 8.11: Grafa generacij najboljših fitnes vrednosti za problem nahrbtnika, primer B.

Histogrami parametrov povprečnega fitnesa skozi izvajanje in povprečnega najboljšega fitnesa skozi izvajanje so si bili med seboj preveč podobni, da bi lahko iz njih predčasno razbrali karkoli konkretnega. Opisna statistika na srečo nima tega problema, zato smo vključili tabelo z njenimi rezultati (Tabela 8.6), pri odstotku starostne razlike 0,5. Tipi izvajanja, katerih povprečne vrednosti najbolj odstopajo od konvencionalnega delovanja, imajo največ možnosti, da bodo signifikantno različni. V tej tabeli vidimo, kakšna mora za to biti povprečna razlika. Mann-Whitney U test potrdi rezultate opisne statistike. Povprečen fitnes skozi izvajanje je pomembno večji v tipih izvajanja 2 in 5 ter manjši v tipu 6, ki predstavlja tudi edino pomembno razliko v delovanju ob 0,15 odstotku starostne razlike. Povprečen najboljši fitnes skozi izvajanje je signifikantno slabši v tipu 2 ob manjšem odstotku starostne razlike in tipu 5 ob večjem odstotku starostne razlike. Ista tipa delovanja sta imela tudi signifikantno slabši najboljši fitnes.

Tabela 8.6: Generacijska parametra, primer B ob odstotku starostne razlike 0,5.

TipIzvajanja		Minimum	Maximum	Mean	Std. Deviation
0	PovprecenFitnesSkoziIzvajanje	2292,83	2757,71	2515,0547	59,88657
	PovprecenMaxFitnesSkoziIzvajanje	2991,58	3320,52	3212,8585	50,76432
1	PovprecenFitnesSkoziIzvajanje	2308,16	2799,33	2512,5009	60,88555
	PovprecenMaxFitnesSkoziIzvajanje	3016,05	3316,93	3211,4321	48,00083
2	PovprecenFitnesSkoziIzvajanje	2334,52	2769,35	2521,4075	61,77393

	PovprecenMaxFitnesSkoziIzvajanje	2997,52	3324,28	3209,4252	49,81267
3	PovprecenFitnesSkoziIzvajanje	2314,77	2735,21	2515,4858	58,99547
	PovprecenMaxFitnesSkoziIzvajanje	3002,66	3311,73	3210,8780	46,89311
4	PovprecenFitnesSkoziIzvajanje	2297,96	2709,19	2512,7695	62,13508
	PovprecenMaxFitnesSkoziIzvajanje	2964,26	3319,48	3210,2790	49,86755
5	PovprecenFitnesSkoziIzvajanje	2318,44	2873,10	2532,8926	68,89176
	PovprecenMaxFitnesSkoziIzvajanje	3022,36	3311,47	3204,9021	51,23221
6	PovprecenFitnesSkoziIzvajanje	2318,13	2686,48	2500,5479	61,84890
	PovprecenMaxFitnesSkoziIzvajanje	2990,27	3317,81	3215,8208	46,98738

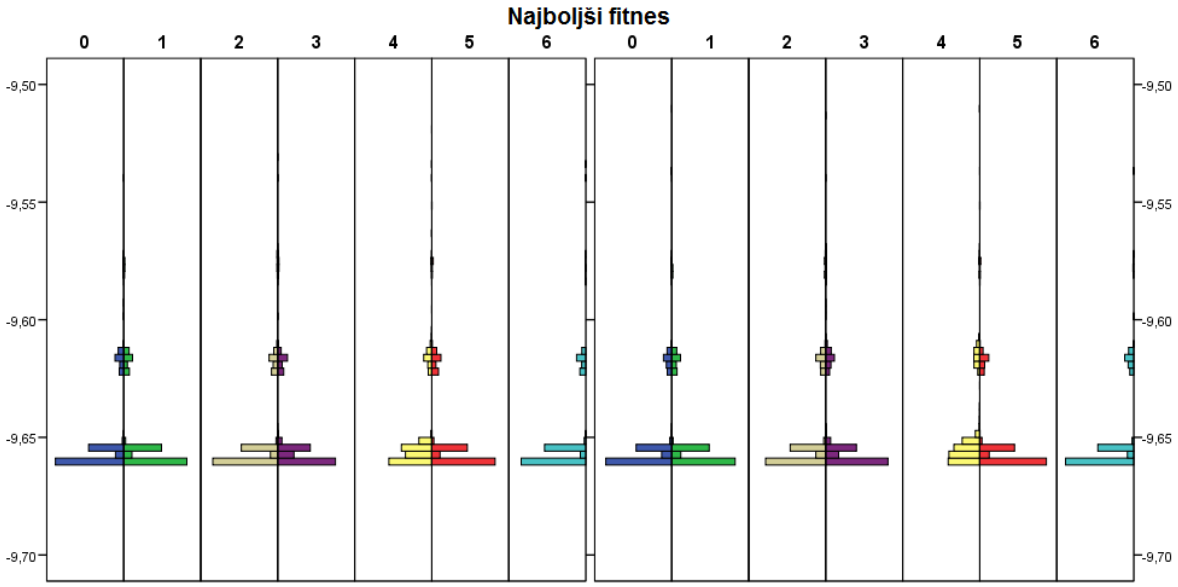
Tabela končnih rezultatov za problem nahrbtnika (Tabela 8.7) pokaže, da predvsem spremembe v operaciji križanja povzročijo razlike v rezultatih, medtem ko so bili v Dawkins Weasel algoritmih različni rezultati izvajanja produkt spremenjene mutacije. Starostno odvisne operacije mutacije v problemu nahrbtnika ne ponujajo velikih sprememb. Tip izvajanja 2, ki je bil v Dawkins Weasel najboljši tip delovanja, se na bolj kompleksnem primeru izkaže kot slaba izbira in ni nudil nobenih prednosti v A primeru. Tipa 1 in 3 ne spremenita delovanja v veliki meri. Način delovanja 4 je posebnost v celotni raziskavi, ker povzroči spremembe v rezultatih (čeprav slabe) ob zelo specifičnih okoliščinah (primer A, pri 0,5 odstotku starostne razlike). Največji učinek imata tipa 5 in 6, vendar ponovno samo ob večji stopnji odstotka starostne razlike. V rezultatih se pokaže tudi, da je povprečen fitnes skozi izvajanje odvisen predvsem od generacije najboljšega fitnesa, medtem ko je povprečen najboljši fitnes skozi izvajanje primarno odvisen od najboljšega fitnesa. Delovanje tipov 5 in 6 je tudi obratno sorazmerno (osebki, ki imajo pri enem večjo možnost izbire v operaciji križanja, so v drugem redkeje izbrani) in razlike, ki nastanejo med njunimi rezultati, so tudi ravno obratne – tip 5 je hitrejši, vendar ne pridobi tako dobrih rezultatov kot konvencionalno delovanje, medtem ko tip 6 tipično najde boljše rezultate, ampak potrebuje za to več generacij. Operaciji mutacije sta si med obema odstotkoma starostne razlike ponovno podobni, z le eno razliko v primarnih parametrih, kar govori o precejšnji konsistentnosti izvajanja.

Tabela 8.7: Rezultati analize za problem nahrbtnika.

	A								B							
	0,15				0,5				0,15				0,5			
	NF	GNF	PF	PNF	NF	GNF	PF	PNF	NF	GNF	PF	PNF	NF	GNF	PF	PNF
1														×		
2			✓				✓		×			×	×		✓	
3			✓					✓								
4					✓	×	×									
5	×		✓	×	×	✓	✓	×					×	✓	✓	×
6			×		✓	×	×				×		✓	×	×	

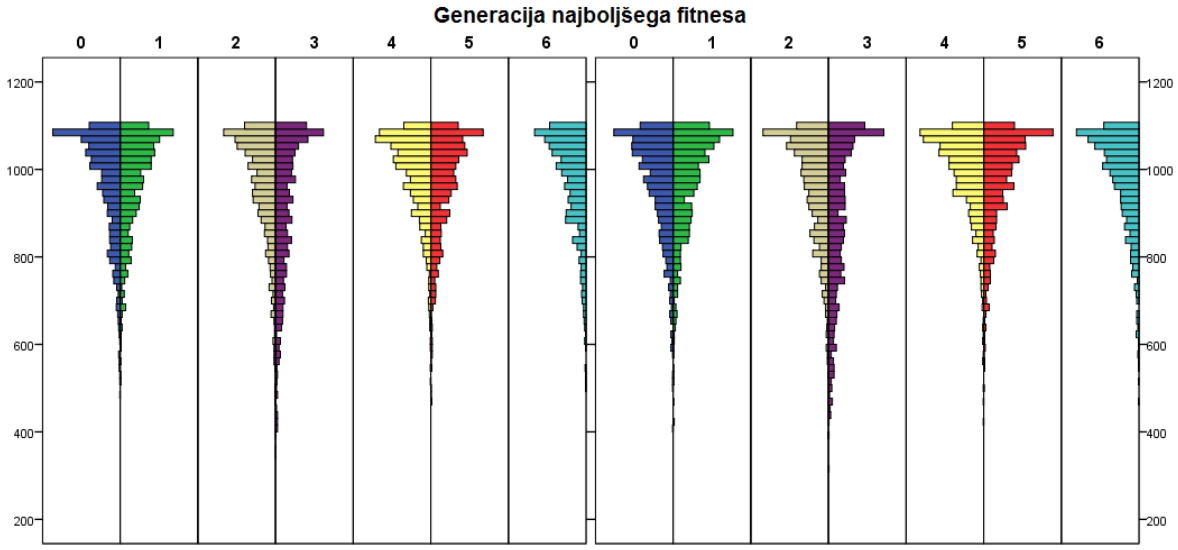
8.4 Funkcija Michalewicz

Algoritem, katerega naloga je najti minimum v funkciji Michalewicz, smo tako kot prejšnje algoritme izvedli ob odstotku starostne razlike 0,15 in 0,5. Rezultati tega GA so v primerjavi s prejšnjimi problemi posebnost, saj so nižje vrednosti fitnesa boljše, kar je potrebno imeti v mislih pri iskanju in interpretiranju razlik med različnimi tipi delovanja. Tako kot vedno začnemo z najboljšim fitnesom (Slika 8.12). Podobno kot pri problemu nahrbtnika A, tudi tu nastaneta dve območji končnih rešitev. Boljši način izvajanja je tisti, ki ima več vrednosti na dnu histograma. Tip izvajanja 4 je zelo drugačen od ostalih in je potrjeno vračal slabše rešitve od konvencionalnega delovanja. V levem grafu sta slabša načina delovanja še 3 in 5, medtem ko je pri odstotku starostne razlike 0,5 slabše rezultate proizvajal tip izvajanja 2.



Slika 8.12: Grafa najboljšega fitnesa za funkcijo Michalewicz.

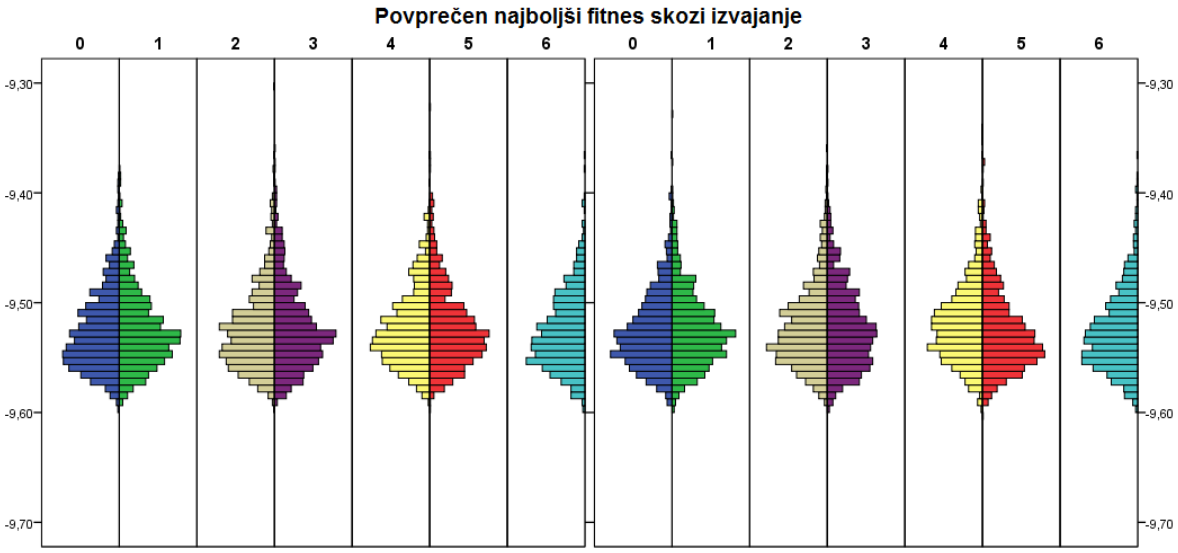
Naslednji diagram predstavlja generacije najboljšega fitnesa (Slika 8.13). Na obeh grafih lahko vidimo, da se histogram 3. tipa izvajanja začne nižje kot ostali in višje vrednosti niso tako pogosto zastopane, kot pri ostalih algoritmih, kar pomeni, da takšno delovanje doseže najboljše rešitve v zgodnejših generacijah. Mann-Whitney U test izpostavi še hitrost tipa 5 na levem grafu in signifikantno počasnejše iskaje tipa 4 v grafu z odstotkom starostne razlike 0,5.



Slika 8.13: Grafa generacij najboljših fitnes vrednosti za funkcijo Michalewicz.

Pri odstotku starostne razlike 0,15 so imeli signifikantno slabši povprečen fitnes skozi izvajanje tipi 1, 4 in 5. Razlik med njimi na histogramih praktično ni mogoče opaziti, saj se povprečna vrednost teh od tipov izvajanja, ki niso imeli pomembno različnih rezultatov proti konvencionalnemu delovanju GA, razlikuje šele na tretjem decimalnem mestu. Ob uporabi vrednosti 0,5 za odstotek starostne razlike, je tip izvajanja 4 edini s signifikantno slabšim rezultatom, medtem ko so vrednosti 3. tipa signifikantno boljše.

V rezultatih parametra povprečnega najboljšega fitnesa skozi izvajanje ni noben tip izvajanja signifikantno presegel tradicionalnega. Na levem grafu so si vsi tipi izvajanja zelo podobni in med njihovimi povprečnimi vrednostmi ni niti pet tisočink razlike. Ne glede na to Mann-Whitney U test najde konsistentno slabše rezultate v tipih 1, 3, 4 in 5. Ob odstotku starostne razlike 0,5 so rezultati malenkostno boljši za starostno odvisne tipe delovanja. Tu je tip delovanja 4 edini signifikantno slabši. Na desnem grafu vidimo, da ima tip 5 pogostejše nižje vrednosti napram tipu 0 in čeprav ima tudi najboljšo povprečno vrednost, med njima ne obstaja pomembna razlika.



Slika 8.14: Grafa povprečnega najboljšega fitnesa skozi izvajanje za funkcijo Michalewicz.

Rezultati pridobljeni za funkcijo Michalewicz so združeni v naslednji tabeli (Tabela 8.8). Iz nje lahko vidimo, da spremembe v delovanju mutacij ponovno niso povzročile velikih razlik v rezultatih GA. Tip izvajanja 6 ni imel povsem nobenega učinka na delovanje. Privilegiranje mlajših genov v operaciji križanja (tip 4) se je izkazalo kot zelo slabo tako za uspešnost kot tudi učinkovitost GA. Tip delovanja 5 je ob določenih pogojih nekoliko hitreje našel optimalno rešitev, vendar za ceno njihove kvalitete. Križanje s prednostnim izborom starejših genov (tip 3) se je izkazalo kot najboljše izmed starostno odvisnih načinov delovanja. Čeprav lahko povzroči slabše rešitve, rezultati kažejo konstantno hitrejše pridobivanje najboljšega osebka izvajanja.

Tabela 8.8: Rezultati analize za funkcijo Michalewicz.

	0,15				0,5			
	NF	GNF	PF	PNF	NF	GNF	PF	PNF
1			×	×				
2					×			
3	×	✓		×		✓	✓	
4	×		×	×	×	×	×	×
5	×	✓	×	×				
6								

9 SKLEP

Genetski algoritmi inteligentno izkoriščajo naključno iskanje za reševanje optimizacijskih problemov. Čeprav delujejo na podlagi naključij, GA niso sami naključni. To dosežejo tako, da uporabijo zgodovinsko znanje za usmerjanje nadaljnjega iskanja na območje, kjer se nahajajo boljše rešitve. Namen te naloge je bil raziskati možnost izboljšanja rezultatov GA z vključitvijo informacije o starosti genov v zgodovinsko znanje. Koristenje novih informacij smo uvedli v tri algoritme s konvencionalnim delovanjem na šest ločenih načinov.

Prvi način vpeljave starostne odvisnosti v delovanje GA je bil s privilegiranjem starejših genov v operaciji mutacije. Rezultati so bili mešani. Za Dawkins Weasel je v zahtevnejšem primeru takšno delovanje sicer zmanjšalo število potrebnih generacij za dosego najboljšega osebka, vendar je bil ta primerljivo slabši od konvencionalnega delovanja. V ostalih dveh algoritmih se takšno delovanje ni bistveno razlikovalo od tradicionalnega. Zatem smo testirali mutacijo s prednostnim izborom mlajših genov. Ta se je pri reševanju Dawkins Weasel problema zelo izkazal. Izboljšali so se tako rezultati uspešnosti najdenih rešitev kot tudi učinkovitost njihovega iskanja. V ostalih problemih ta način delovanja nima učinka ali celo poslabša končne rešitve. Preostale načine implementacije starostne odvisnosti smo uvedli v operacijo križanja. Ponovno začnemo s privilegiranjem starejših genov. Vplivi tega delovanja niso bili opazni v Dawkins Weasel ali problemu nahrbtnika, medtem ko je v primeru funkcije Michalewicz bil to najbolj učinkovit način izvajanja GA, čeprav ob možnosti poslabšanja uspešnosti. Križanje s povečano verjetnostjo izbire mlajših genov je bilo najslabše od vseh. V prvih dveh problemih takšno delovanje ni pomembno odstopalo od konvencionalnega delovanja, vendar se je izkazalo za zelo slabo pri iskanju minimuma funkcije Michalewicz. Peti način vpeljave starostne odvisnosti v delovanje GA je povečal verjetnost izbora povprečno starih genov v operaciji križanja. V Dawkins Weasel problemu ni pokazal velikih razlik z osnovnim delovanjem. V problemu nahrbtnika je ob dovolj velikem učinku starostne odvisnosti za ceno kvalitete rešitev izboljšal učinkovitost, medtem ko je isto dosegel tudi v funkciji Michalewicz, vendar tokrat ob vrednosti odstotka starostne razlike 0,15. Zadnji starostno odvisen način križanja je obratno od prejšnjega pri izbiri dal prednost najmlajšim in najstarejšim genom. Uporaba takšnega delovanja ni imela učinka na problema Dawkins Weasel in funkcijo Michalewicz. Čeprav je v problemu nahrbtnika nosil verjetnost počasnejšega iskanja optimalne rešitve, je v primerjavi s konvencionalnim delovanjem našel boljše rešitve.

Pregled nad vsemi načini starostno odvisnega delovanja GA, ki smo jih v tej nalogi vpeljali vanje, pokaže, da noben način ni univerzalno boljši od standardnega delovanja GA. Pri vsakem problemu se je drugi tip delovanja pokazal kot najboljši. Drugo zanimivo opažanje je ekskluzivnost med operacijama mutacije in križanja. V problemih, kjer je spremenjeno delovanje mutacije vplivalo na rezultate, so bile učinki spremenjene operacije križanja komajda opazni in isto velja tudi v obratni smeri. Razlogi za takšno obnašanje nam trenutno niso povsem znani. Čeprav smo v nalogi pokazali, da starostno odvisna izbira genov lahko izboljša uspešnost in učinkovitost GA, vprašanje kdaj uporabiti kateri način delovanja, ostaja. Zato menimo, da so rezultati te naloge uporabni predvsem ob gradnji GA, ki bodo pogosto izvajani, in kjer se morda izplača predhodno vložiti nekaj časa v raziskavo vplivov, ki jih starostno odvisna izbira genov ima na dan problem ter z implementacijo takšnega delovanja izboljšati končen produkt.

9.1 Omejitve in predlogi za nadaljevanje magistrskega dela

Genetski algoritmi zavzemajo zelo obsežno področje. Delno pri tem govorimo o širokem izboru možnosti za izdelavo algoritma, kot so različni načini selekcije, mutiranja, križanja in tako dalje, delno pa se nanašamo na možen nabor problemov, ki jih lahko z GA rešujemo. Obe lastnosti omejujeta našo raziskavo. GA v tej nalogi so bili izdelani z določenim naborom standardnih tehnik, ki se pogosto uporabljajo, vendar to ni nobeno zagotovilo, da rezultati te naloge držijo tudi za drugačne implementacije algoritmov. Z uporabo treh različnih problemov za testiranje smo v nalogi poizkušali nekoliko bolje pokriti področja delovanja GA in s tem izboljšati zunanjo veljavnost raziskave. Ne glede na to vsestranskost uporabe GA ne omogoča, da bi lahko rezultate te naloge brez pomisleka posploševali na katerikoli problem. Zadnja omejitev, ki jo velja poudariti, so uporabljeni načini vpeljave starostne odvisnosti genov v GA. Ti so bili bolj ali manj izbrani arbitrarno, ob upoštevanju določenih želja o zajetju oziroma uporabi splošnih starostnih skupin. Ker niso bili izbrani po merljivem kriteriju in obstaja mnogo drugih možnosti vpeljave starostne odvisnosti genov, izbrani načini niso predstavljiv vzorec vsega, kar lahko starostna odvisnost genov doprinese v delovanje GA.

Raziskava je pustila mnogo odprtih vprašanj, ki bi bila odlična izhodiščna točka za nadaljnje delo. Rezultati so pokazali, da različni načini vpeljave starostne odvisnosti pri izbiri genov prinesejo različne posledice v reševanju različnih problemov. Če bi lahko našli razloge za posamezne rezultate novih načinov delovanja oziroma povezavo med lastnostmi problema in uspešnimi ter učinkovitimi načini starostno odvisnega delovanja, potem bi dokončno

pokazali uporabnost takšnega pristopa kot tudi olajšali izbiro načina delovanja za posamezne probleme. To je dolgoročen cilj ideje o starostno odvisni izbiri genov v GA. Za njegovo dosego bi zato lahko raje začeli z raziskovanjem novih načinov implementacije starostne odvisnosti genov v algoritme ali že obstoječe načine testirali na novih, še ne preizkušenih problemih. Zanimiva bi bila tudi raziskava pojava, ki smo ga zasledili med našimi rezultati, kjer je na rešitve posameznega problema imela signifikanten vpliv skoraj izključno samo sprememba ene izmed operacij mutacije in križanja, medtem ko so spremembe v drugi komajda povzročile razliko v primerjavi z izvajanjem konvencionalnega genetskega algoritma.

VIRI IN LITERATURA

[1] Y. K. Singh, Fundamental of Research Methodology and Statistics, 2006.

[2] A. Bhattacherjee, Social Science Research: Principles, Methods, and Practices vol. Book 3: Open Access Textbooks, 2012.

[3] W. M. K. Trochim. (2006). Reliability & Validity. Dostopano 9. 5. 2015. Dosegljivo na: http://www.socialresearchmethods.net/kb/relandval.php

[4] L. Cohen, L. Manion in K. Morrison, Research Methods in Education, 6th edition ed., 2007.

[5] Y. Williams. Cause and Effect Relationship: Definition, Examples & Quiz. Dostopano 9. 5. 2015. Dosegljivo na: http://education-portal.com/academy/lesson/cause-and-effect-relationship-definition-examples-quiz.html#lesson

[6] R. L. Haupt in S. E. Haupt, Practical Genetic Algorithms, 2nd Edition ed., 2004.

[7] R. M. Friedberg, "A learning machine: part I," IBM J. Res. Dev., vol. 2, pp. 2-13, 1958.

[8] R. M. Friedberg, B. Dunham in J. H. North, "A learning machine: part II," IBM J. Res. Dev., vol. 3, pp. 282-287, 1959.

[9] A. S. Fraser, "Simulation of Genetic Systems by Automatic Digital Computers I. Introduction," Australian Journal of Biological Sciences, vol. 10, pp. 484-491, 1957.

[10] J. H. Holland, "Nonlinear Environments Permitting Efficient Adaptation," presented at the Computer and Information Sciences II, 1967.

[11] J. H. Holland, Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press, 1975.

[12] A. Marczyk, "Genetic Algorithms and Evolutionary Computation," ed: The TalkOrigins Archive 2004, 2004.

[13] K. De Jong, L. Fogel in H.-P. Schwefel, The Handbook of Evolutionary Computation: Taylor & Francis, 1997.

[14] M. Melanie, An Introduction to Genetic Algorithems, Reprint ed.: A Bradford Book, 1998.

[15] A. Shukla, R. Tiwari in R. Kala, Real Life Applications of Soft Computing: CRC Press, 2010.

[16] A. Abraham, N. Nedjah in L. de Macedo Mourelle, Evolutionary Computation: from Genetic Algorithms to Genetic Programming: Springer Berlin Heidelberg, 2006.

[17] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm", predstavljeno na Proceedings of the Second International Conference on Genetic

Algorithms on Genetic algorithms and their application, Cambridge, Massachusetts, USA, 1987.

[18] K. A. D. Jong, Evolutionary Computation, 1st ed.: A Bradford Book, 2002

[19] R. Dawkins, The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design: Norton, 1986.

[20] D. Cornforth, D. G. Green, in J. Awburn, "Weasel World: a simple artificial environment for investigating open-ended evolution," v 8th Asia Pacific Symposium on Intelligent Evolutionary Systems (IES 2004)(D. Green, B. McKay in A. Namatame, 6 december 2004 do 7 december 2004), 2004, pp. 40-49.

[21] T. Hudson. (2010). Methinks it is like an incestuous weasel. Dostopano 1. 6. 2015. Dosegljivo na: http://tomnomnom.com/posts/methinks-it-is-like-an-incestuous-weasel

[22] S. Martello in P. Toth, Knapsack problems: algorithms and computer implementations: John Wiley; Sons, Inc., 1990.

[23] D. Pisinger, "Algorithms for Knapsack Problems," Dept. of Computer Science, University of Copenhagen, Copenhagen, Denmark, 1995.

[24] H. Pohlheim, "Examples of Objective Functions," GEATbx Examples, 2005.

[25] S. Surjanovic and D. Bingham. Michalewicz Function. Dostopano 1. 6. 2015. Dosegljivo na: http://www.sfu.ca/~ssurjano/michal.html

[26] E. W. Weisstein. Gaussian Function. Dostopano 6. 6. 2015. Dosegljivo na: http://mathworld.wolfram.com/GaussianFunction.html

[27] W. R. Leo. The Gaussian or Normal Distribution. Dostopano 6. 6. 2015. Dosegljivo na: http://ned.ipac.caltech.edu/level5/Leo/Stats2_3.html

[28] Faltung einer Gauss-Funktion mit sich selbst. Dostopano 5. 6. 2015. Dosegljivo na: http://me-lrt.de/u-05-2-faltung-einer-gauss-funktion-mit-sich-selbst

[29] Mann-Whitney U Test using SPSS. Dostopano 10. 7. 2015. Dosegljivo na: https://statistics.laerd.com/spss-tutorials/mann-whitney-u-test-using-spss-statistics.php

[30] Assumptions of the Mann-Whitney U test. Dostopano 10. 7. 2015. Dosegljivo na: https://statistics.laerd.com/premium-sample/mwut/mann-whitney-test-in-spss-2.php

PRILOGE

Priloga A

Aplikacija za izvajanje genetskih algoritmov

- Priložena zgoščenka

Priloga B

SPSS Statistics zbirka podatkov in rezultati statistične analize

- Priložena zgoščenka

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

IZJAVA O AVTORSTVU

Spodaj podpisani/-a Marko Kompara

z vpisno številko E5012752

sem avtor/-ica magistrskega dela z naslovom: Vpeljava verjetnosti izbire genov na osnovi starosti pri genetskih algoritmih

(naslov magistrskega dela)

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)
  red. prof. dr. Vili Podgorelec
  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela.
- soglašam z javno objavo elektronske oblike magistrskega dela v DKUM.

V Mariboru, dne 24. 8. 2015

Podpis avtorja/-ice:

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek avtorja-ice: Marko Kompara

Vpisna številka: E501 2752

Študijski program: Informatika in tehnologije komuniciranja (2. stopnja)

Naslov zaključnega dela: Vpeljava verjetnosti izbire genov na osnovi starosti pri genetskih algoritmih

Mentor: rod. prof. dr. Vili Podgorelec

Somentor:

Podpisani-a Marko Kompara izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna z elektronsko verzijo elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____ ne sme biti javno dostopno do _____(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj: 24. 8. 2015, Maribor Podpis avtorja-ice: _____

Podpis mentorja: _____
(samo v primeru, če delo ne sme biti javno dostopno)

Podpis odgovorne osebe naročnika in žig:
(samo v primeru, če delo ne sme biti javno dostopno) _____

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor :

red. prof. dr. Vili Podgorelec

(ime in priimek mentorja)

in somentor (eden ali več, če obstajata):

/

(ime in priimek somentorja)

Izjavljam (-va), da je študent

Ime in priimek: MARKO KOMPARA

Št. indeksa: E5012752

Na programu: Informatika in tehnologije komuniciranja (2. stopnja)

izdelal zaključno delo z naslovom:

Vpeljava verjetnosti izbire genov na osnovi starosti pri genetskih algoritmih

(naslov zaključnega dela v slovenskem in angleškem jeziku)

The introduction of age based gene selection probability in genetic algorithms

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in mojimi (najinimi oziroma našimi) navodili.

Preveril (-a, -i) in pregledal (-a, -i) sem (sva, smo) poročilo o plagiatorstvu.

Datum in kraj:
24.8.2015

Podpis mentorja:

Datum in kraj:

Podpis somentorja (če obstaja):