SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 657

Optimizacija raspoređivanja u okruženju nesrodnih strojeva

Marko Đurasević

Zagreb, lipanj 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 13. ožujka 2014.

DIPLOMSKI ZADATAK br. 657

Pristupnik: **Marko Đurasević**
Studij: Računarstvo
Profil: Programsko inženjerstvo i informacijski sustavi

Zadatak: **Optimizacija raspoređivanja u okruženju nesrodnih strojeva**

Opis zadatka:

Opisati problem raspoređivanja u okruženju nesrodnih strojeva i primjenu ovog modela raspoređivanja. Istražiti postojeće postupke raspoređivanja s obzirom na uvjete uporabe i načine izgradnje rasporeda. Ostvariti sustav raspoređivanja u okruženju nesrodnih strojeva prioritetnim raspoređivanjem uz pomoć genetskog programiranja. Ocijeniti učinkovitost inačica genetskog programiranja s obzirom na uvjete uporabe i dostupne podatke. Predložiti strukturu i preporučene elemente raspoređivača na temelju svojstava okoline raspoređivanja. Usporediti učinkovitost ostvarenih postupaka s postojećim postupcima raspoređivanja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 30. lipnja 2014.

Mentor:

Izv.prof.dr.sc. Domagoj Jakobović

Predsjednik odbora za diplomski rad profila:

Prof.dr.sc. Krešimir Fertalj

Djelovođa:

Doc.dr.sc. Igor Mekterović

Zahvaljujem se svojim roditeljima na svemu što su mi pružili tijekom života i studija te djedu i baki na predivnom djetinjstvu i uspomenama koje nikad neću zaboraviti.

Također, zahvaljujem mentoru, prof. dr. sc. Domagoju Jakobovića, na svim savjetima, konzultacijama i na cjelokupnoj pruženoj pomoći i podršci tijekom izrade ovog rada, kao i tijekom ostatka studija.

SADRŽAJ

1. Uvod

Raspoređivanje je proces u kojemu se određen skup sredstava dodjeljuje određenom skupu aktivnosti. Rješenje problema raspoređivanja jest raspored koji određuje koje će aktivnosti biti raspoređene na koja sredstva i kojim redoslijedom. Proces raspoređivanja može se prepoznati u mnogim stvarnim problemima kao što su izrada rasporeda sati, raspoređivanje poslova u *cluster* okruženju, raspoređivanje operacija po medicinskim salama i slično.

Cilj problema raspoređivanja jest izgraditi raspored koji optimira određene uvjete, odnosno kriterije postavljene na sam raspored (primjerice smanjenje ukupne duljine rasporeda, smanjenje ukupnog kašnjenje poslova i slično). Nažalost, izrada rasporeda koji optimira pojedini kriterij veoma često nije jednostavan zadatak jer problemi raspoređivanja najčešće spadaju u klasu NP teških problema. Probleme koji spadaju u navedenu klasu nije moguće riješiti metodama iscrpne pretrage jer vrijeme koje bi bilo potrebno za pronalazak optimalnog rješenja je neprihvatljivo. Iz navedenog razloga se za rješavanje ovog problema najčešće koriste određene heurističke metode koje ne garantiraju da će pronađeno rješenje biti optimalno, ali navedeni postupci uspijevaju pronaći dovoljno dobra rješenja u prihvatljivom vremenskom periodu.

U ovom radu pobliže će se proučiti primjena genetskog programiranja na problem izrade rasporeda. Genetsko programiranje jest metaheuristički postupak koji simuliranjem prirodne evolucije nastoji pronaći program ili funkciju koja predstavlja rješenje za neki dani problem. Ovaj algoritam koristi genetske operatore selekcije, mutacije i križanja kako bi kroz iteracije od "lošijih" rješenja uspio izgraditi "bolja" rješenja.

U drugom poglavlju formalno je opisan problem raspoređivanja, konkretno za okolinu nesrodnih strojeva. Prikazana su i pojašnjena svojstva poslova i strojeva koja su bitna za izradu samog rasporeda. Također su opisani i kriteriji koji se najčešće koriste za ocjenu rasporeda. Nakon toga navedeni su i pojašnjeni različiti uvjeti raspoređivanja ovisno o raspoloživosti parametara, pouzdanosti parametara i načinu izrade rasporeda. Konačno, na kraju poglavlja opisano je nekoliko poznatijih heurističkih postupaka razvijenih upravo za rješavanje problema raspoređivanja.

U trećem poglavlju detaljno je opisan algoritam genetskog programiranja. Opisan je način na koji su rješenja predstavljena u samom algoritmu te na koji način se stvara inicijalni skup rješenja. Nakon toga opisani su genetski operatori selekcije, križanja i mutacije te su za svaki od navedenih operatora opisani najčešći načini provođenja istih. Na kraju poglavlja opisani su kriteriji zaustavljanja algoritma, kao i neki od parametara koji su važni za rad algoritma.

U četvrtom poglavlju opisano je raspoređivanje zasnovano na prilagodljivim pravilima. Konkretno, radi se o genetskom programiranju koje je prilagođeno za izradu prioritetnih funkcija za problem raspoređivanja u okolini nesrodnih strojeva. Kroz poglavlje opisani su dijelovi algoritma genetskog programiranja koji su prilagođeni za primjenu na raspoređivanje u okolini nesrodnih strojeva, kao primjerice skup korištenih terminala, načini na koje se ocjenjuju rješenja i slično.

U petom poglavlju opisane su optimizacije koje su provedene nad postupkom raspoređivanja zasnovanog na prilagodljivim pravilima s ciljem dobivanja boljih rješenja. Za početak je opisan postupak pronalaženja optimalnih parametara genetskog programiranja. Nakon toga opisani su novi funkcijski čvorovi koji su uvedeni u ovaj postupak s ciljem povećanja ekspresivnosti algoritma. Nadalje opisana je varijanta genetskog programiranja, pod nazivom semantičko genetsko programiranje, koja jamči da će izrazi dobiveni algoritmom biti semantički ispravni te je opisano kako je naveden algoritam prilagođen za problem raspoređivanja na nesrodnim strojevima. Prikazana je i inačica algoritma koja za postizanje boljih rezultata koristi i iterativan postupak raspoređivanja koji konačan raspored gradi kroz nekoliko iteracija. Konačno, opisan je i postupak, pod nazivom GEP, koji koristi ponešto drugačiji zapis jedinki.

U šestom poglavlju opisani su rezultati koji su dobiveni za različite postupke optimizacija koji su opisani u petom poglavlju te su dodatno komentirani dobiveni rezultati. Na kraju poglavlja rezultati dobiveni kroz ove optimizacije su uspoređeni s rezultatima koji su dobiveni korištenjem drugih postupaka za izradu rasporeda.

U sedmom poglavlju dan je kratak zaključak ovog rada.

2. Raspoređivanje na nesrodnim strojevima

U stvarnom svijetu svakodnevno se susrećemo s mnogo primjera postupaka raspoređivanja, kao na primjer raspoređivanje medicinskih operacija po salama, raspoređivanje aviona po pistama, raspoređivanje zadataka u *cluster* okruženjima i slično. Samo raspoređivanje moguće je definirati kao proces dodjele ograničenih sredstava određenom skupu aktivnosti s ciljem optimiranja mjerila vrednovanja te raspodjele. Sredstva koja se dodjeljuju mogu predstavljati različite stvari - strojeve u nekom proizvodnom pogonu ili procesore na računalima. S druge strane, aktivnosti predstavljaju razne operacije koje se izvršavaju na tim strojevima, a to mogu biti primjerice neke operacije u proizvodnji ili računalni programi. U literaturi se aktivnosti još veoma često nazivaju zadacima ili poslovima, dok se sredstva nazivaju strojevima.

Jedan od glavnih ciljeva proučavanja problema raspoređivanja jest smanjenje troškova uporabe sredstava. Nažalost, pokazano je kako je veliki broj ovakvih problema veoma težak za rješavanje. Iako su ljudima danas na raspolaganju veoma brza i moćna računala, problemi koji se nastoje riješiti su jednako tako postali mnogo složeniji i zahtjevniji. Za veliku većinu takvih problema nije moguće definirati egzaktni algoritam kojim bi se mogla dobiti optimalna rješenja, već je za dobivanje rješenja potrebno upotrijebiti različite heurističke postupke.

Nadalje, nužno je napomenuti da iako postoje različite varijante raspoređivanja s obzirom na vrstu strojeva, kao što su paralelni identični strojevi, jednoliki strojevi, nesrodni strojevi i slično, u ovom radu fokus će biti stavljen samo na jednu vrstu raspoređivanja i to konkretno na problem raspoređivanja u okruženju nesrodnih strojeva.

Okruženje nesrodnih strojeva jest okruženje u kojem svaki stroj obrađuje svaki pojedini posao proizvoljno definiranom brzinom. To znači da je za svaki par stroj-posao definirana brzina kojom se zadani posao izvršava na zadanom stroju. Iz tog razloga može se doći do zaključka da su strojevi nezavisni, odnosno nesrodni, jer čak i ako je poznata brzina izvršavanja određenih poslova na strojevima, ta informacija se

ne može iskoristiti kako bi se odredili odnosi među strojevima (primjerice, čak i ako je poznato da se svi promatrani poslovi izvršavaju dvostruko brže na jednom stroju od nekog drugog, ipak nije moguće zaključiti kako je prvi stroj dvostruko brži od drugog, zbog činjenice da se može pojaviti posao koji će se izvršavati jednakom brzinom na oba stroja ili čak brže na drugom). Intuitivno je jasno kako je broj kombinacija na koje je moguće raspodijeliti ove poslove velik te da za veće instance problema jednostavno nije moguće pronaći optimalno rješenje u nekom razumnom vremenu. Iz tog razloga za rješavanje ovog problema koriste se različite heurističke i metaheurističke metode kojima se nastoji ostvariti dovoljno dobra rješenja u prihvatljivom vremenu.

2.1. Svojstva poslova

Posao, odnosno zadatak, jest određena aktivnost koja se postupkom raspoređivanja želi dodijeliti nekom stroju kako bi se na njemu izvršavala. Ovisno o okruženju u kojem se obavlja raspoređivanje, poslovi se mogu sastojati od više aktivnosti ili se pak posao može sastojati od samo jedne nedjeljive aktivnosti. U nastavku poglavlja navest će se najvažnija svojstva kojima su opisani poslovi i koji predstavljaju temelj za razumijevanje postupaka raspoređivanja. Sva ova svojstva, njihovi nazivi, kao i njihove oznake preuzeti su iz [7].

Skup svih poslova označavat će se s J, dok će se pojedini poslovi označavati s J_j, pri čemu indeks j predstavlja redni broj posla. Najvažnija svojstva koja su, ovisno o okolini, pridružena svakom poslu su:

- **Trajanje izvođenja posla** p_{ij} (engl. *processing time*) - označava trajanje koje je potrebno da se posao J_j obavi na stroju i.
- **Vrijeme pripravnosti posla** r_j (engl. *ready time, release time*) - označava vrijeme, odnosno trenutak u kojemu posao postaje raspoloživ za raspoređivanje. Prije svog vremena raspoloživosti niti jedan posao se ne može rasporediti niti izvoditi na nekom stroju. S druge strane, posao može čekati na početak izvođenja neodređeno dugo nakon svog vremena pripravnosti, što može biti posljedica zauzetosti svih strojeva ili neispunjavanja nekog drugog uvjeta.
- **Vrijeme željenog završetka posla** d_j (engl. *due date*) - označava vrijeme do kojeg je poželjno, odnosno do kojeg se očekuje da će pojedini posao završiti. Naravno, vrijeme završetka niti u kojem slučaju ne osigurava da će se

pojedini posao stvarno završiti prije tog vremena, ali time je ipak osigurano da se, ako posao završi nakon tog vremena, stvori određeni trošak.

- **Vrijeme nužnog završetka posla** $\bar{d}_j$ (engl. *deadline, drop dead time*) - označava strogo vremensko ograničenje do kojega posao nužno mora završiti. Ovo se ograničenje najčešće javlja u sustavima za rad u stvarnom vremenu.
- **Težina posla** w_j (engl. *weight*) - označava prioritet nekog zadatka u sustavu. Najčešće se koristi pri određivanju troškova kod ocjenjivanja rasporeda, gdje ona predstavlja neku stvarnu mjeru kvalitete samog rasporeda. U slučaju da se ocjena rasporeda temelji na više kriterija, poslu se može pridružiti više težina koje onda definiraju prioritete za svaki pojedini kriterij.

2.2. Ocjena kvalitete rasporeda

Svaki raspored koji je generiran pojedinom metodom mora se moći ocijeniti na neki način. Naravno, treba uzeti u obzir da neće u svim primjenama raspoređivanja svi kriteriji biti jednako važni. Stoga je potrebno razviti nekoliko različitih načina ocjenjivanja rasporeda. No prije nego što se kratko opišu ti načini ocjenjivanja, potrebno je definirati neke izlazne veličine sustava, koje će onda kasnije poslužiti za ocjenjivanje samog rasporeda. Sve oznake i nazivi izlaznih veličina preuzeti su iz [7].

- **Vrijeme završetka** C_j (engl. *completion time*) - predstavlja trenutak u kojemu je posao s indeksom j završio s izvođenjem.
- **Protjecanje** F_j (engl. *flowtime*) - predstavlja količinu vremena koju je neki posao proveo u sustavu, odnosno količinu vremena od kada je posao postao pripravan za izvođenje do trenutka kada je on završio s izvođenjem:

$$F_j = C_j - r_j$$

- **Kašnjenje** L_j (engl. *lateness*) - predstavlja razliku između vremena završetka posla i vremena željenog završetka posla (potrebno je uočiti da ona može biti pozitivna i negativna):

$$L_i = C_j - d_j$$

- **Zaostajanje** T_j (engl. *tardiness*) - predstavlja pozitivan iznos kašnjenja nekog posla (ako je kašnjenje negativno, ono se postavlja na nulu):

$$T_j = max\{0, L_j\}$$

- **Preuranjenost** E_j (engl. *earliness*) - predstavlja negativan iznos kašnjenja nekog posla, odnosno koliko je taj posao završio ranije od željenog vremena završetka:

$$E_j = max\{0, -L_j\}$$

- **Zakašnjelost** U_j - predstavlja oznaku koja pokazuje je li neka aktivnost prekoračila željeno vrijeme završetka:

$$U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}$$

Nakon što su sada opisane sve potrebne izlazne veličine sustava, mogu se opisati i pojedini kriteriji vrednovanja dobivenih rasporeda.

- **Ukupna duljina rasporeda** C_{max} (engl. *makespan*) - predstavlja posljednje vrijeme završetka svih poslova u sustavu:

$$C_{max} = max\{C_j\}$$

- **Najveće kašnjenje** L_{max} (engl. *maximum lateness*) - predstavlja najveće vrijeme kašnjenja nekog posla u sustavu:

$$L_{max} = max\{L_j\}$$

- **Težinsko protjecanje** F_w (engl. *weighted flowtime*) - definira se kao suma težinskog protjecanja svih poslova u sustavu:

$$F_w = \sum_j w_j F_j$$

- **Težinsko zaostajanje** T_w (engl. *weighted tardiness*) - definira se kao težinska suma zaostajanja svih poslova:

$$T_w = \sum_j w_j T_j$$

- **Težinski broj zaostalih poslova ili težinska zakašnjelost** U_w (engl. *weighted number of tardy jobs*) - definirana je kao težinska suma svih zaostalih poslova:

$$U_w = \sum_j w_j U_j$$

- **Težinska preuranjenost i težinsko zaostajanje** ET_w (engl. *weighted earliness and weighted tardiness*) - definirano je kao zbroj težinske preuranjenosti i težinskog zaostajanja, s time da su težinski faktori posebno definirani za obje navedene vrijednosti:

$$ET_w = \sum_j (w_{E_j} E_j + w_{T_j} T_j)$$

Naravno, logično je za pretpostaviti kako prethodno navedeni načini ocjene nisu jedini koji se koriste. Međutim, iako postoji mnogo načina ocjenjivanja rasporeda, a moguće je čak i izraditi specifične ocjene rasporeda, upravo ovi načini predstavljaju skup najčešće korištenih načina ocjenjivanja rasporeda.

Koji će se od navedenih kriterija za ocjenjivanje rasporeda koristiti ovisi o tome koji su prioriteti postavljeni na sam raspored. Tako je u pojedinim slučajevima važnije da svi poslovi što prije završe, dok je pak za druge važnije da je vrijeme kašnjenja pojedinačnih poslova što manje, pri čemu nije toliko bitno hoće li ukupno trajanje poslova biti najmanje moguće. Osim toga, pojedinim poslovima moguće je dati i veći prioritet, što će uzrokovati da ukupna ocjena više ovisi o tome kako su raspoređeni ti poslovi. Također, raspored je moguće ocijeniti i s nekoliko kriterija odjednom, pri čemu se odabire onaj raspored koji najbolje zadovoljava odabrane kriterije.

Vidljivo je kako su mogućnosti ocjenjivanja rasporeda mnogobrojne i kako odabir načina ocjenjivanja rasporeda ovisi o zahtjevima i prioritetima koji su postavljeni na raspored.

2.3. Uvjeti raspoređivanja

Uvjeti raspoređivanja odnose se na različite mogućnosti izrade rasporeda te na vremenski odnos između same izrade rasporeda i njegovog izvođenja u nekom stvarnom sustavu. Postupci raspoređivanja koji se koriste za izradu rasporeda ovise o upravo o spomenutim uvjetima raspoređivanja. Tako se može dogoditi da su pojedini postupci raspoređivanja pod pojedinim uvjetima neprimjenjivi, primjerice zbog neraspoloživosti pojedinih podataka. Većina postupaka s kojima se moguće susresti u teoriji raspoređivanja zahtijevaju raspoloživost i nepromjenjivost svih potrebnih podataka, dok to primjerice u stvarnim uvjetima ne mora biti ispunjeno. Nadalje, za izradu rasporeda ne mora uvijek biti na raspolaganju neograničena količina vremena, već vrijeme za donošenje sljedeće odluke može biti ograničeno određenim parametrom. Posljedično, uvjeti raspoređivanja mogu se podijeliti po nekoliko kriterija koji su navedeni u nastavku.

2.3.1. Raspoloživost parametara

Svi algoritmi koji se koriste za izradu rasporeda kao svoje ulazne veličine koriste određene parametre sustava, kao što su primjerice broj poslova i strojeva, vrijeme pripravnosti poslova, težine poslova i slično. Iz tog razloga, odabir algoritma usko je vezan uz skup raspoloživih parametara u sustavu. U slučaju da svi parametri sustava koji su potrebni za rad određenog algoritma nisu dostupni, očito je da se taj algoritam tada ne može primijeniti za izradu rasporeda već će se morati odabrati drugi algoritam. S obzirom na raspoloživost parametara, sustave za raspoređivanje možemo podijeliti u dvije skupine:

- **Predodređeno raspoređivanje** (engl. *offline scheduling*) - ovaj oblik raspoređivanja pretpostavlja da su vrijednosti svih potrebnih parametara sustava poznate prije izrade rasporeda i prije rada samog sustava. To znači da je, na primjer, unaprijed poznat ukupni broj poslova, kao i sve njihove značajke koje su bitne za izradu rasporeda. Osim toga, u svakom trenutku rada poznata je i potpuna budućnost sustava, što može značiti da su unaprijed poznati dolasci svih poslova i slično. Nadalje, u ovom obliku raspoređivanja podrazumijeva se da su poznate vrijednosti parametara konstantne kroz cijeli rad sustava. Treba primijetiti da je ovo donekle idealistički pogled na problem raspoređivanja, jer u stvarnom svijetu obično nisu na raspolaganju sve takve informacije i ne postoji garancija da se veličine tijekom rada sustava neće promijeniti.
- **Raspoređivanje na zahtjev** (engl. *online scheduling*) - za razliku od predodređenog raspoređivanja, kod ove vrste raspoređivanja nisu poznati svi parametri sustava. Kod raspoređivanja na zahtjev proces raspoređivanja obavlja se samo na temelju trenutno dostupnih informacija, bilo kakve informacije o budućnosti sustava su nedostupne i ne mogu se koristiti (primjerice, značajke poslova su poznate tek u onom trenutku kada posao postane raspoloživ). Osim toga, u ovom uvjetu raspoređivanja također je moguća promjena pojedinih poznatih veličina (npr. kod kvara nekog od strojeva i slično). U ovakvom okruženju postavlja se ograničenje na uporabu algoritama koji mogu relativno brzo reagirati na promjene u sustavu, no jednako tako nije potrebno niti definirati raspored za sve preostale poslove, već samo odrediti sljedeće stanje sustava.

2.3.2. Pouzdanost parametara

Neovisno o tome o kojoj se raspoloživosti radi, okolina raspoređivanja može se još dodatno kategorizirati i po kriteriju koji određuje pouzdanost parametara. Pouzdanost parametara definirana je preciznošću kojom su određeni sami parametri. Razlikujemo dvije skupine pouzdanosti:

- **Determinističko raspoređivanje** (engl. *deterministic scheduling*) - u ovom obliku okruženja pretpostavlja se da su vrijednosti svih parametara sustava određene s nekom točno definiranom preciznošću.
- **Stohastičko raspoređivanje** (engl. *stochastic scheduling*) - kod ovog oblika okruženja pretpostavlja se da ne postoje pouzdane procjene vrijednosti parametara sustava, već je stvarne vrijednosti moguće očitati tek na kraju rada određenog dijela sustava. S druge strane, vrijednosti parametara nisu u potpunosti nepoznate, već su definirane funkcijama koje se ponašaju po određenim vjerojatnosnim raspodjelama.

2.3.3. Način izrade rasporeda

Ovisno o dostupnim podacima i uvjetima koji su postavljeni izradu rasporeda možemo podijeliti na dva različita pristupa:

- **Statički pristup** - u ovom pristupu potrebno je već prije samog početka rada sustava u potpunosti izgraditi raspored. U ovakvom postupku podrazumijeva se korištenje predodređenog raspoređivanja jer su za izgradnju kompletnog rasporeda unaprijed potrebne sve informacije o relevantnim parametrima sustava (npr. ukupan broj poslova, vrijeme pripravnosti svih poslova i slično). Ovaj pristup omogućuje korištenje postupaka koji su računalno i vremenski veoma zahtjevni i najčešće nisu upotrebljivi u uvjetima za rad u stvarnom vremenu.
- **Dinamički pristup** - ovaj pristup, za razliku od prethodnog, ne izgrađuje raspored odjednom, već se raspored gradi kroz više iteracija. Izgradnja rasporeda događa se paralelno s radom sustava kojeg se raspoređuje. Ovi postupci najčešće se pozivaju samo onda kada u sustavu nastane određena promjena, pri čemu oni određuju novo stanje u koje sustav prelazi. Dobra strana ovih postupaka jest ta što se oni mogu koristiti i u slučaju predodređenog raspoređivanja (primjerice u sustavima za rad u stvarnom vremenu,

kada nam je bitno što prije krenuti s izvođenjem sustava i ne možemo si priuštiti izgradnju čitavog rasporeda unaprijed), kao i u slučaju raspoređivanja na zahtjev (kada uopće niti ne postoji mogućnost izrade cijelog rasporeda unaprijed).

2.4. Tehnike raspoređivanja na nesrodnim strojevima

Kao što je već ranije napomenuto, a što je vjerojatno i intuitivno jasno iz prethodnih poglavlja, raspoređivanje na nesrodnim strojevima predstavlja veoma težak i zahtjevan problem za koji ne postoji efikasan algoritam pomoću kojeg bi se moglo dobiti optimalno rješenje. Iz tog razloga, za rješavanje ovog problema koriste se mnoge heurističke metode koje ne garantiraju da će dobiveno rješenje biti optimalno, ali mogu izgraditi dovoljno dobro rješenje u prihvatljivom vremenu. Heurističke metode koje se koriste za rješavanje ovog problema mogu se podijeliti u dvije skupine: algoritme koji pretražuju prostor rješenja i algoritme koji izravno izgrađuju rješenje. U nastavku bit će opisane obje vrste ovih heuristika.

2.4.1. Tehnike pretraživanja prostora stanja

Kao što se i mnogi drugi NP-teški problemi rješavaju uz pomoć raznih evolucijskih i populacijskih algoritama, tako se i problem izrade rasporeda na nesrodnim strojevima može napasti ovim algoritmima. Zbog same prirode algoritama ovog tipa, veoma je teško dobiti optimalna rješenja. Međutim, oni često pronalaze dovoljno dobra rješenja koja zadovoljavaju određene uvjete. Primjeri algoritama koji spadaju u ovu skupinu su: genetski algoritmi, simulirano kaljenje, optimizacija rojem čestica, genetsko simulirano kaljenje, tabu pretraživanje i mnogi drugi.

Generiranje rasporeda korištenjem ovih algoritama pokazalo se kao dobra praksa. Ako je algoritam dobro dizajniran i implementiran, on može dati veoma dobra rješenja za zadani problem. Međutim, kako se ipak radi o nedeterminističkim algoritmima čija krajnja rješenja uvelike ovise o početnim rješenjima koja se obično generiraju potpuno nasumično, često je potrebno po nekoliko puta izvesti dani algoritam kako bi se dobila što bolja krajnja rješenja. Tom problemu može se također doskočiti izbjegavanjem potpuno nasumičnog stvaranja populacije pri čemu se dio početne populacije generira pomoću heuristika namijenjenih upravo za izradu rasporeda (neke takve heuristike bit će opisane u idućem poglavlju).

Nažalost, ovi algoritmi imaju i određeni skup nedostataka koji rezultira time da

nisu uvijek primjenjivi na rješavanje problema raspoređivanja. Jedan od njih je taj da su ovakvi algoritmi veoma spori i računalno zahtjevni. To nije toliki problem ako je dostupno dovoljno vremena za izradu rasporeda, no ako je raspored potrebno izraditi u veoma kratkom vremenskom periodu, onda ovakav tip algoritama nije primjenjiv. Nadalje, ovi algoritmi primjenjivi su uglavnom u statičkom predodređenom raspoređivanju jer nemaju mogućnost brze reakcije na promjene koje se mogu dogoditi u sustavu (primjerice, dolazak novih poslova u sustav).

Kao što je vidljivo, ovakvi načini za izradu rasporeda predstavljaju dobro rješenje u slučajevima predodređenog statičkog raspoređivanja bez strogog uvjeta na vrijeme izrade samog rasporeda. U suprotnome, ovakve metode nije moguće efikasno primijeniti, već je potrebno iskoristiti neke druge postupke za izradu rasporeda.

2.4.2. Tehnike izravne izgradnje rješenja

U slučajevima kada je bitna brzina i u uvjetima kada je potrebno brzo reagirati na promjene u stanju sustava, najčešće se koriste upravo postupci koji rješenje grade izravno. U literaturi se za ove postupke veoma često koriste nazivi *pravila raspoređivanja* (engl. *dispatching rules, scheduling rules*) ili čak *heuristike* (u užem smislu). Princip rada ovih heuristika svodi se na uvođenje određene metrike koja se koristi za ocjenu elemenata sustava te na temelju koje se odabiru elementi (oni s najvećom vrijednosti te metrike). Primjerice, svakom poslu koji je potrebno rasporediti se za svaki stroj na koji se može rasporediti pridijeli određena vrijednost spomenute metrike. Zatim se od svih poslova odabere onaj koji ima minimalnu vrijednost te metrike, te se promatrani posao rasporedi na stroj za koji je postignuta upravo ta minimalna vrijednost. Ove heuristike primjenjuju se uzastopno, najčešće paralelno s radom samog sustava, sve dok se ne izradi kompletan raspored, ili se sustav ne dovede u stanje mirovanja.

Ovakvi postupci nude nekoliko prednosti. Brzina izvođenja gotovo je zanemariva, posebice u usporedbi s postupcima koji pretražuju prostor stanja. Nadalje, za razliku od tehnika pretraživanja prostora stanja, ovi postupci se mogu primijeniti i u dinamičkim okruženjima, kao i kod raspoređivanja na zahtjev, jer ovi postupci određuju samo iduće stanje u koje sustav prelazi, te su za njihov rad dovoljni samo trenutni podaci o sustavu (npr. podaci o poslovima koji još nisu ušli u sustav nisu potrebni za određivanje idućeg stanja). Ovakvi algoritmi mogu veoma brzo reagirati na promjene koje se pojave u sustavu. Konačno, ovi algoritmi su konceptualno veoma jednostavni i razumljivi.

S druge strane, ove heuristike ne dolaze i bez određenih nedostataka. Jedan od najvećih nedostataka jest taj da su rezultati dobiveni ovim postupcima ponešto lošiji od

rezultata dobivenih postupcima pretraživanja prostora stanja, što je i očekivano. Nadalje, postoji jako mnogo ovih heuristika i često nije jasno koja je od tih heuristika idealna za izradu rasporeda nad konkretnim problemom. To je slučaj jer njihove performanse uvelike ovise o karakteristikama samog problema. Iz tog razloga, heuristike koje inače daju lošije rezultate mogu na nekim konkretnim instancama problema dati bolje rezultate od nekih generalno uspješnijih heuristika.

U nastavku bit će prikazan kraći pregled nekih od heuristika koje se koriste za izradu rasporeda.

2.4.2.1. OLB

OLB (engl. *Opportunistic load balancing*) [11] jednostavna je heuristika koja svaki posao nasumično dodjeljuje na idući slobodni stroj ne uzimajući pri tome u obzir trajanje izvođenja tog posla na pojedinim strojevima.

2.4.2.2. MET

MET (engl. *Minimum execution time*) [11][3] heuristika je heuristika koja svaki posao pridjeljuje onom stroju koji ima najmanje očekivano trajanje izvođenja. U ovoj heuristici se ne uzima se u obzir dostupnost, kao niti opterećenost konkretnog stroja.

2.4.2.3. MCT

MCT (engl. *Minimum completion time*) [11][3] heuristika je heuristika koja za svaki posao računa najmanje očekivano trajanje izvođenja u slučaju pridjeljivanja tog posla nekom konkretnom stroju. Odabire se onaj stroj koji je polučio najmanjom vrijednošću očekivanog trajanja izvođenje. Također je važno za napomenuti kako ova heuristika uzima u obzir samo jedan posao u svakom trenutku.

2.4.2.4. Min-min

Min-min [5][3] jedna je od najpopularnijih i najčešće korištenih heuristika za izradu rasporeda u okruženju nesrodnih strojeva. Ovaj algoritam postiže veoma dobre rezultate unatoč činjenici da je vrlo jednostavan.

Postupak počinje sa skupom svih neraspoređenih poslova. U prvoj fazi postupka, za svaki neraspoređeni posao određuje se stroj koji daje i najmanje očekivano vrijeme završetka za taj posao. Nakon toga, u drugoj fazi, postupak raspoređuje onaj posao koji postiže najmanje očekivano vrijeme završetka i to na stroj na kojem se to vrijeme može

ostvariti. Nakon što je taj korak obavljen potrebno je ponovo izračunati očekivano vrijeme završetka svih raspoređenih poslova na svakom od strojeva. Opisani postupak ponavlja se sve dok konačno nisu raspoređeni svi poslovi.

Iz opisa algoritma vidljivo je kako on kao funkciju odlučivanja koristi upravo očekivano vrijeme završetka te da ovaj postupak radi upravo tako da se odabire onaj posao za koji će, nakon raspoređivanja, očekivano vrijeme završetka biti najmanje moguće. Složenost ovog algoritma jest $O(ms^2)$ gdje m predstavlja broj strojeva dok s predstavlja broj poslova.

2.4.2.5. Max-min

Max-min [5] [3] druga je veoma popularna i često korištena heuristika koja se koristi za raspoređivanje u okolini nesrodnih strojeva. Sama heuristika veoma je slična Min-min heuristici što će se bolje vidjeti iz opisa rada heuristike.

Postupak također počinje sa skupom svih neraspoređenih poslova. Prva faza je u potpunosti jednaka prvoj fazi Min-min algoritma - za svaki posao traži se stroj koji će za taj posao dati najmanje očekivano vrijeme završetka. U drugoj fazi se, za razliku od Min-min algoritma, ne raspoređuje onaj posao koji daje minimalno očekivano vrijeme završetka, već onaj koji daje maksimalno vrijeme završetka među svim poslovima. Ostatak postupka je identičan kao i kod Min-min algoritma - za svaki stroj se izračuna njegovo novo vrijeme završetka, nakon čega se cijeli postupak ponavlja dok svi ostali poslovi nisu raspoređeni.

Kao što se i iz opisa vidi, algoritam Max-min samo je druga inačica Min-min algoritma pri čemu se ta razlika očituje se u kriteriju raspoređivanja poslova. Naime, ovaj postupak nastoji što ranije izvršiti poslove koji se dulje izvode. U slučajevima kada postoje poslovi koji se izvode mnogo dulje od ostalih poslova, ovaj postupak može dati bolja rješenja od Min-min algoritma. Složenost postupka je jednaka kao i kod Min-min algoritma $O(ms^2)$.

2.4.2.6. Duplex

Duplex algoritam [11] jednostavna je kombinacija algoritama Min-min i Max-min. Oba navedena algoritma izvode se međusobno nezavisno te se na kraju odabere ono rješenje koje daje bolji rezultat. Kako su troškovi ovakvih algoritama veoma maleni, gotovo zanemarivi, čak ni izvođenje dva ovakva algoritma ne predstavlja veliki trošak. No ovaj algoritam može se koristiti samo u predodređenom raspoređivanju, jer tek nakon što su oba rasporeda napravljena može se ocijeniti njihova kvaliteta i odabrati

onaj bolji.

Postoji heuristika koja također kombinira Min-min i Max-min heuristiku, no na ponešto drugačiji način [3]. Ova inačica heuristike u svakom koraku odabire hoće li za raspoređivanje koristiti Min-min ili Max-min algoritam. Odluka o tome koji će se algoritama primijeniti donosi se na temelju trenutnog stanja sustava.

2.4.2.7. Sufferage

Ideja ovog algoritma jest da se na određeni stroj rasporedi točno onaj posao koji bi najviše "propatio" (engl. *to suffer*) ako se ne bi rasporedio točno na taj stroj [3][5].

Za svaki posao se, osim minimalnog očekivanog vremena završetka posla, traži i drugo najmanje vrijeme završetka tog posla na nekom stroju. Razlika između ta dva vremena definira se kao "patnja". Nakon toga odabere se posao koji ima najveću vrijednost "patnje" i rasporedi na stroj na kojem postiže najmanje očekivano vrijeme završetka. Na taj način daje se prioritet onim poslovima koji bi se puno dulje izvodili od drugih, u slučaju da nisu raspoređeni na onaj stroj za kojeg postižu najmanje očekivano vrijeme završetka.

2.4.2.8. LJRF-SJRF

LJRF-SJRF (engl. *Longest job to fastest resource - shortest job to fastest resource*) [5] heuristika započinje također sa skupom neraspoređenih poslova. Kao i kod Min-min algoritma, najprije se za svaki posao odredi minimalno očekivano vrijeme završetka. Nakon toga se posao s najmanjom vrijednošću očekivanog vremena završetka postavi kao najkraći posao (SJFR), a posao s najvećom vrijednošću očekivanog vremena završetka kao najdulji posao (LJRF). Na početku algoritam dodijeli m najduljih poslova na m dostupnih strojeva, a nakon toga se alternira između odabira najkraćeg i najduljeg posla kojeg će se rasporediti na određeni stroj.

2.4.2.9. Min-max

Min-max nešto je novija heuristika prvi put predložena u [5]. Ona se također sastoji od dvije faze za dodjeljivanje poslova na strojeve. U prvoj fazi algoritma se kao metrika koristi očekivano trajanje završetka, dok se u drugoj fazi koristi najmanje trajanje izvršavanje poslova. Prvi korak započinje sa skupom svih neraspoređenih poslova. Kao i kod Min-min algoritma, za svaki posao odredi se stroj za koji taj posao postiže najmanje očekivano vrijeme završetka. U drugoj fazi raspoređuje se onaj posao

koji ima najveći omjer između minimalnog vremena izvođenja i vremena izvođenja na odabranom stroju (iz prve faze).

Ideja ovog algoritma jest da se odabere onaj par posao-stroj iz prve faze, koji će odabrani posao izvršiti što je efikasnije moguće u usporedbi s ostalim strojevima na kojima bi se taj posao mogao izvršiti.

2.4.2.10. Min-mean

Min-mean [11] heuristika funkcionira tako da u prvom koraku generira raspored koristeći Min-min heuristiku. U drugom koraku računa se srednja vrijednost od trenutaka dovršetka rada (trenutak kada su se svi poslovi koji su bili raspoređeni na taj stroj izvršili) svih strojeva. U idućoj fazi odabiru se samo oni strojevi koji imaju očekivano vrijeme dovršetka poslova veće od izračunate srednje vrijednosti. Za svaki takav stroj provodi se ponovno raspoređivanje svih onih poslova koji se još ne izvode na strojeve koji imaju vrijeme dovršetka manje od prosjeka. Na taj način nastoji se ujednačiti opterećenost svih strojeva.

Postoji i nešto novija inačica ovog postupka koja uvođenjem jedne male promjene postiže ponešto bolje rezultate [12]. Naime, autori ove metode utvrdili su da je moguće dobiti bolje rezultate ako se srednjoj vrijednosti trenutaka završetka rada svih strojeva nadoda određena konstantna vrijednost. Kroz eksperimente je pokazano kako idealan iznos te konstantne vrijednosti iznosi oko 7% dobivene srednje vrijednosti. Ostatak algoritma je u potpunosti analogan.

3. Genetsko programiranje

Genetsko programiranje široko je korištena metaheuristika koja se koristi za rješavanje teških problema za koje ne postoje egzaktni algoritmi, ili bi rješavanje nekim od poznatih algoritama jednostavno bilo presporo. Ono spada u grupu optimizacijskih tehnika koje nazivamo evolucijskim računanjem (engl. *Evolutionary Computation*). Po svojim karakteristikama najsličnije je genetskom algoritmu, od kojeg se razlikuje samo po obliku u kojem su zapisane jedinke. Dok genetski algoritam, a i slični pristupi, traže rješenje za neki dani problem, genetsko programiranje zapravo traži "program" čijim će se izvođenjem dobiti rješenje određenog problema. Algoritam genetskog programiranja primjenjuje se za rješavanje širokog spektra problema, od kojih su neki: problemi raspoređivanja, problemi regresije i klasifikacijski problemi [2] [6].

Sam algoritam genetskog programiranja u suštini je veoma jednostavan te se sastoji od samo nekoliko koraka. Najprije je potrebno generirati početnu populaciju jedinki koje predstavljaju potencijalna rješenja danog problema. Nakon toga, svakoj jedinki odredi se dobrota koja određuje koliko je to rješenje zapravo "dobro". Tada se nad populacijom rješenja provode genetski operatori križanja (stvaranje nove jedinke kombiniranjem drugih jedinki), mutacije (uvođenjem nasumičnih promjena u jedinku) i selekcije kako bi se dobila nova, po mogućnosti bolja, rješenja. Genetski operatori provode se dok nije zadovoljen jedan od uvjeta zaustavljanja. Kao krajnje rješenje problema uzima se jedinka s najboljom dobrotom. U idućim poglavljima bit će detaljnije opisani svi elementi genetskog programiranja.

3.1. Prikaz jedinki

U genetskom programiranju jedinke su prikazane u obliku sintaksnog stabla pri čemu one najčešće predstavljaju program ili funkciju koja se onda dalje koristi za rješavanje konkretnog problema. Što se tiče same strukture sintaksnog stabla, ono se sastoji od čvorova koje je moguće podijeliti na dva disjunktna skupa, a to su terminalni i funkcijski čvorovi. Terminalni čvorovi predstavljaju konstantne veličine (npr. 3.14, 33, 7, 42 i

slično), kao i ulazne veličine u sam program, odnosno varijable (npr. x, y, z). Čvorovi ovog tipa mogu isključivo biti listovi stabla. Drugu grupu predstavljaju funkcijski čvorovi koji mogu predstavljati aritmetičke funkcije (npr. +, -, *, /), logičke funkcije (npr. i, ili, isključivo-ili), ili neke druge korisnički definirane funkcije. Funkcijski čvorovi se mogu nalaziti samo u unutarnjim čvorovima stabla, a nikako u listovima (iznimka su funkcijski čvorovi koji nemaju operanada nego direktno vračaju neku vrijednost, npr. slučajno generiranje brojeva, no one se zapravo tretiraju kao terminalni čvorovi). Prethodno navedena dva skupa zajedno nazivamo skupom primitiva genetskog programiranja.

Osim što je za prikaz jedinki bitan skup primitiva od kojih su sačinjeni čvorovi stabla, za jedinke su također veoma bitne i veličine koje određuju minimalnu i maksimalnu dubinu stabla koje sama jedinka predstavlja. Maksimalna dubina stabla izrazito je važna veličina jer se njome nastoji spriječiti pojava prekomjernog rasta stabla u dubinu (engl. *bloat*), što je nažalost veoma česta pojava u genetskom programiranju. Naime, tijekom evolucije stabla rastu sve više i više u dubinu bez ikakvog značajnog poboljšanja što se tiče dobrote jedinki. To je veoma nezahvalno svojstvo jer se time povećava složenost stabla, što posljedično uzrokuje njihovu dugotrajniju evaluaciju, ali i težu interpretaciju. Iz tog razloga javlja se zahtjev da se rast stabla ograniči, što se obično postiže zadavanjem maksimalne dubine stabla unaprijed, ili korištenjem regularizacije. Regularizacija modelira utjecaj složenosti stabla na samu funkciju dobrote, tako da što više kažnjava funkciju dobrote što je stablo "veće" [21].

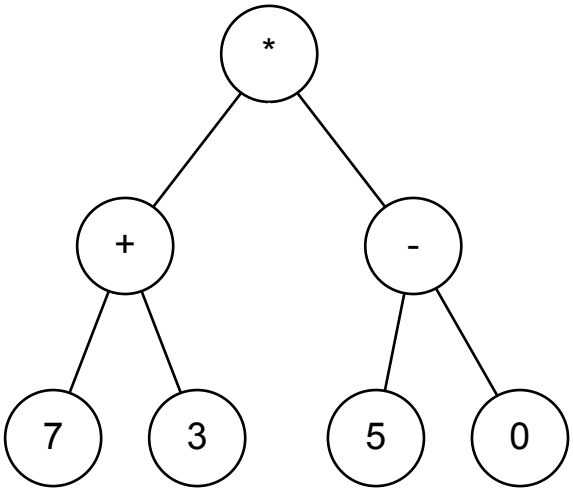
Svakoj jedinki je, osim stabla koje predstavlja rješenje, pridružena i dobrota. Dobrota je numerička vrijednost koja određuje koliko neka jedinka predstavlja dobro rješenje. Ona omogućuje efektivnu usporedbu dvije jedinke te odabir one bolje (boljeg rješenja). Dobrota jedinke najčešće se računa uprosječivanjem nekolicine instanci konkretnog problema čije rješenje sama jedinka predstavlja.

3.2. Inicijalizacija populacije

Genetski algoritam spada u populacijske algoritme, što znači da algoritam ne radi samo nad jednim rješenjem, već na mnogo njih. Na taj način želi se pokriti što veće područje prostora pretraživanja kako bi se smanjila mogućnost zapinjanja algoritma u lokalnom optimumu. Upravo iz tog razloga poželjno je da se početna populacija sastoji od što više različitih jedinki. Kako bi se osiguralo to svojstvo, nužno je definirati strategije inicijalizacije početne populacije jedinki. Najčešće se koriste tri načina inicijalizacije populacije, a to su: *full*, *grow* i *ramped half-and-half*. U nastavku će svaka od tih

metoda biti opisana detaljnije.

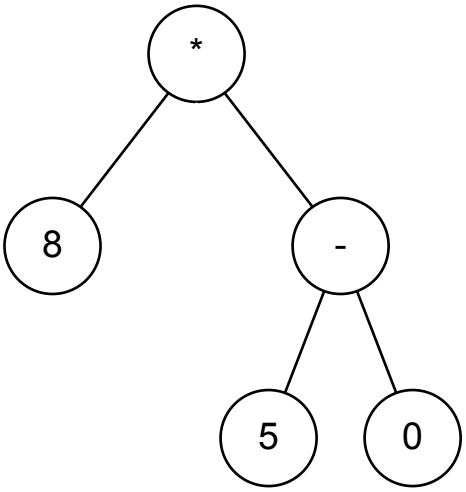
Full metoda stvara stabla koja će biti potpuna, odnosno u kojima će svi listovi imati istu dubinu (duljinu puta od korijena do nekog čvora). Sam postupak je jednostavan - ako je trenutna dubina čvora manja od maksimalne, čvor se nasumično odabire iz skupa funkcijskih čvorova. S druge strane, ako je dubina jednaka maksimalnoj dozvoljenoj dubini, tada se čvor nasumično odabire iz skupa terminalnih čvorova. Nadalje, treba napomenuti da iako ova metoda stvara stabla kod kojih listovi imaju jednaku dubinu, to ne znači da će broj čvorova koje stabla sadrže biti jednaki. Razlog tome je taj da ne moraju svi funkcijski čvorovi imati isti broj operanada. No bez obzira na to, broj različitih struktura stabla koje ova metoda može izgraditi je dosta ograničen. Slika 3.1 prikazuje primjer stabla maksimalne dubine dva generiranog *full* metodom.



Slika 3.1: Stablo generirano *full* metodom

Grow metoda ne zahtijeva da svi listovi imaju jednaku dubinu te na taj način omogućuje izgradnju stabala s mnogo raznovrsnijim strukturama. U ovom postupku čvor se stvara nasumičnim odabirom jednog čvora iz skupa primitiva. Jedino u slučaju kada se radi o čvoru koji ima dubinu jednaku maksimalnoj dubini potrebno je odabrati čvor iz skupa terminalnih čvorova (kako se ne bi narušilo ograničenje o maksimalnoj dubini stabla). Slika 3.2 prikazuje primjer stabla maksimalne dubine dva generiranog *grow* metodom.

Niti jedna od ovih dviju metoda sama po sebi ne pruža veliku raznolikost u strukturi generiranih stabala. Zbog tog razloga predložena je nova metoda koja se naziva *ramped half-and-half* koja je trebala doskočiti upravo tom problemu [21]. Ova metoda inicijalizira populaciju na način da prvu polovicu populacije inicijalizira *grow* metodom, a drugu polovicu *full* metodom. Osim toga, ova metoda također koristi ograničenja



Slika 3.2: Stablo generirano *grow* metodom

na dubinu jedinki, tako da su jedinke kreirane s različitim maksimalnim dubinama (no nikada većom od one originalne koja je zadana) čime nastoji postići veću raznolikost u strukturi populacije. Ova metoda jedna je od najčešće korištenih metoda inicijalizacije populacije.

3.3. Selekcija

Kao i kod ostalih evolucijskih algoritama, genetski operatori (križanje i mutacija) se obavljaju nad jedinkama koje su probabilistički odabrane na temelju svoje dobrote. Drugim riječima, to znači da će bolje jedinke imati veću vjerojatnost preživljavanja te prijenosa svog genetskog materijala i obrnuto. Na taj način želi se simulirati postupak prirodne selekcije. Postoji mnogo postupaka na koje se selekcija može provesti, a najpoznatiji su svakako turnirska selekcija i proporcionalna selekcija.

Turnirska selekcija najčešće je korištena selekcija u praksi. Ona počinje nasumičnim odabirom k jedinki iz populacije, pri čemu je k parametar turnirske selekcije (k se obično postavlja barem na vrijednost tri). Iz tog nasumično odabranog skupa jedinki odabiru se dvije jedinke koje će se križati te jedinka koja će biti obrisana iz populacije. Na mjesto obrisane jedinke postavit će se dijete koje je nastalo upravo tim križanjem. Parametar k veoma je važan iz razloga što o njemu ovisi koliko jedinki nikada neće dobiti priliku da budu odabrane za križanje. Primjerice, kod 7-turnirske selekcije, pet najgorih jedinki nikako neće moći biti odabrano za križanje jer će uvijek postojati barem 2 jedinke koje su bolje. Iz tog razloga njihov genetski materijal nikada neće biti iskorišten. Spomenuta situacija jest razlog zbog kojeg se najčešće koristi parametar k

jednak tri. Općenito, turnirska selekcija ima niz prednosti zbog kojih se često koristi, a neke od njih su jednostavnost i mogućnost paralelizacije.

Proporcionalna selekcija, s druge strane, svakoj jedinki dodjeljuje određenu vjerojatnost odabira. Vjerojatnost odabira pojedine jedinke određuje se kroz iznos dobrote te jedinke, što se najčešće postiže normalizacijom dobrote određene jedinke (dobrota određene jedinke podijeli se s ukupnom dobrotom svih jedinki u populaciji). Nakon toga jedinke se nasumično odabiru iz populacije, pritom uzimajući u obzir prethodno izračunatu vjerojatnost odabira. Međutim, treba napomenuti kako je ova metoda nešto zahtjevnija od turnirske selekcije jer je pri svakoj promjeni populacije potrebno ponovo izračunati vjerojatnosti odabira za svaku pojedinu jedinku.

Selekcijski postupci dodatno se mogu podijeliti prema načinu prenošenja genetskog materijala boljih jedinki u sljedeću iteraciju. Po ovom kriteriju selekcijski postupci mogu se podijeliti na eliminacijske i generacijske postupke [22].

Kod generacijskih selekcija odabiru se bolje jedinke koje će sudjelovati u reprodukciji. Prvo se bolje jedinke iz originalne populacije kopiraju u novu populaciju jedinki (koja se naziva međupopulacijom). Nad tom novom populacijom se obavljaju operatori križanja i mutacije čime se stvara iduća populacija jedinki. Iz ovog opisa već su očiti određeni problemi ove vrste selekcije. Prvi problem koji se javlja jest potreba za pohranom dvije populacije u memoriji računala (u isto vrijeme). Nadalje, broj jedinki koji preživi selekciju obično je manji od originalnog broja jedinki pa se prazna mjesta popunjavaju s duplikatima preživjelih jedinki, što je nepoželjno jer duplikati smanjuju genetsku raznolikost. Konačno, preživljavanje najbolje jedinke (elitizam) nije inherentno osigurano ovim postupkom, već se mora dodatno ugraditi.

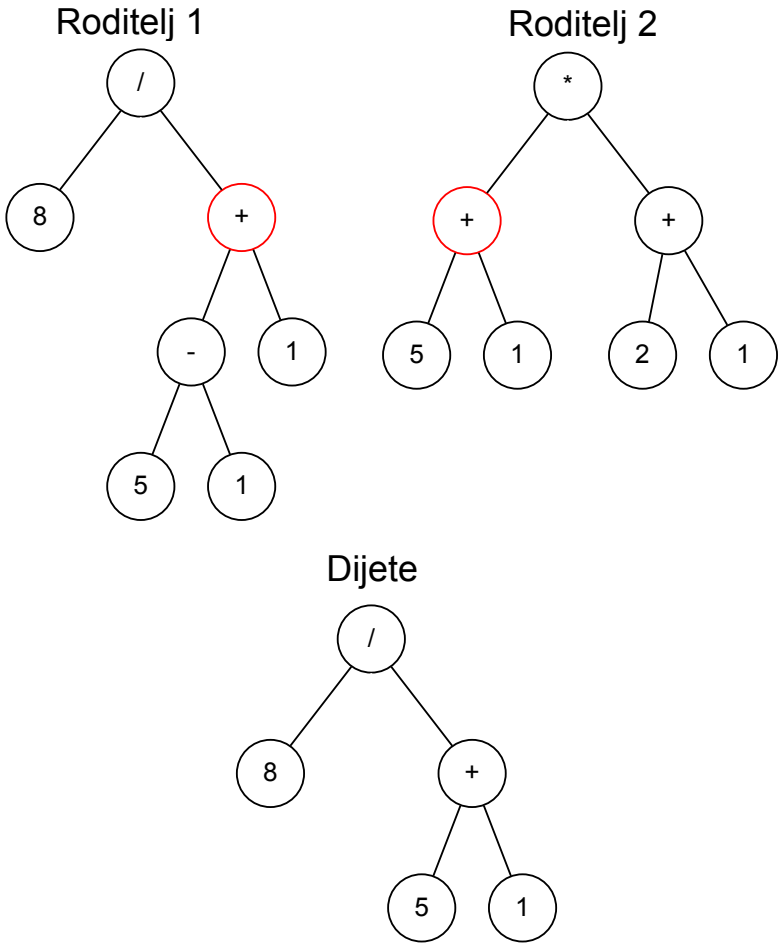
Druga vrsta selekcijskog postupka jest eliminacijska selekcija. Kod ove selekcije bolje jedinke preživljavaju, dok lošije bivaju nadomještene jedinkama koje su nastale križanjem i mutacijom iz boljih jedinki. Prethodna činjenica ističe jednu od prednosti eliminacijske selekcije. Naime, najbolje jedinke uvijek će preživjeti što znači da je u ovu selekciju inherentno ugrađen elitizam. Nadalje, u ovoj vrsti selekcije nije potrebno pohraniti dvije populacije odjednom, već se jedinke brišu i dodaju u populaciju jedna po jedna. Također je poželjno spomenuti kako ova selekcija otklanja i nedostatak kopiranja jedinki. Iz navedenih razloga, eliminacijska selekcija se puno češće koristi od generacijske selekcije.

3.4. Križanje

Križanje je genetski operator koji kombinira dvije jedinke kako bi stvorio novu jedinku koja sadrži kombinaciju genetskog materijala izvornih jedinki. Izvorne jedinke nazivaju se roditeljima, dok se novonastalu jedinku naziva djetetom. Cilj ove kombinacije je stvoriti dijete koje će biti bolje od svojih roditelja. Naravno, to nikako nije zajamčeno jer se dijete stvara stohastički pa postoji mogućnost da će dijete biti i lošije. Također je bitno za napomenuti kako će se dijete nakon križanja sastojati isključivo od genetskog materijala roditelja. To povlači činjenicu da, u slučaju da roditelj ne sadrži određeni element iz skupa primitiva, neće ga sadržavati niti dijete. Križanje kod genetskog programiranja mora biti prilagođeno stablastoj strukturi jedinki. Zbog toga se ovi operatori ponešto razlikuju od svojih inačica u drugim evolucijskim algoritmima, no princip ostaje uglavnom isti. U nastavku je objašnjeno nekoliko varijanti operatora križanja koje se koriste u genetskom programiranju.

3.4.1. Križanje podstabla

Križanje podstabla (engl. *Subtree Crossover*) [21] je varijanta križanja kod kojeg se nasumično i nezavisno odaberu točke križanja (odnosno čvorovi) u svakom od roditelja. Dijete se stvara na način da se podstablo koje ima korijen u točki križanja prvog stabla zamijeni podstablom drugog roditelja koje ima korijen u njegovoj točki križanja. U ovoj varijanti, točke križanja obično nisu odabrane s uniformnom vjerojatnošću. Razlog tome je taj što su čvorovi u stablu većinom terminali pa ako bi svi čvorovi imali uniformnu vjerojatnost da budu odabrani za točke križanja, došlo bi do toga da se veoma često razmijene jako malene količine genetskog materijala među roditeljima. Kako bi se spomenuti problem izbjegao preporučuje se da se unutarnji čvorovi stabla odabiru u 90% slučajeva, dok se listovi odabiru samo u 10% slučajeva. Slika 3.3 prikazuje primjer ove vrste križanja. Crvenom bojom označeni su čvorovi koji predstavljaju točke križanja kod svakog od roditelja.

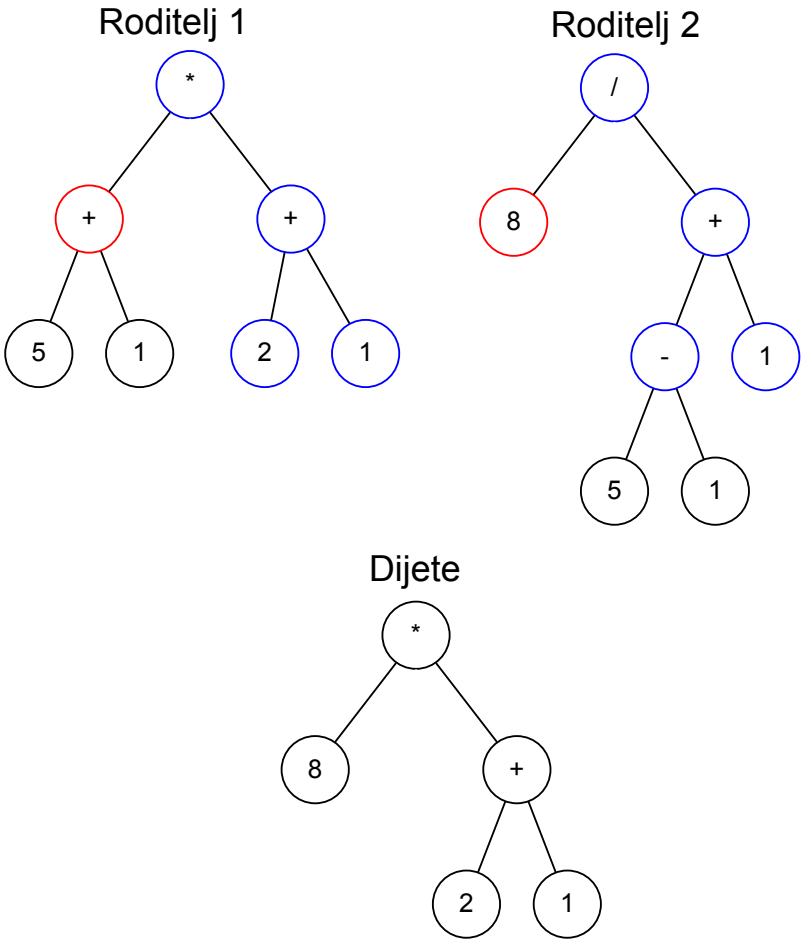


Slika 3.3: Primjer križanja podstabla

3.4.2. Križanje s jednom točkom

Križanje s jednom točkom (engl. *one-point crossover*) [21] spada u posebnu skupinu križanja koja se nazivaju homolognim (engl. *homologous*) križanjima. Homologna križanja nastoje očuvati poziciju na kojima se genetski materijal nalazi. Konkretno, ovo križanje nastoji križati čvorove koji imaju istu dubinu. Za potpuno definiranje ovog križanja, također je nužno uvesti pojam zajedničke regije (engl. *common region*). Zajednička regija područje je u jedinkama koje obuhvaća sve čvorove za koje vrijedi da svi odgovarajući parovi čvorova između roditelja imaju jednak broj operanada. Ako to za trenutni čvor ne vrijedi, taj čvor se bez obzira na to uključuje u zajedničku regiju, no njegova djeca se preskaču. Točka križanja se onda odabire nasumično iz te zajedničke regije, ali ne i nezavisno za svakog od roditelja, već je odabrana točka križanja istovjetna za oba roditelja. Postupak je nadalje isti kao i kod križanja podstabla, to jest podstablo koje ima korijen u točki križanja jednog roditelja zamjenjuju se s pod-

stablom drugog roditelja koje ima korijen u točki križanja. Slika 3.4 prikazuje primjer ove vrste križanja. Plavom bojom označeni su čvorovi koji pripadaju u zajedničku regiju, dok su crvenom bojom označeni oni čvorovi iz zajedničke regije koji odabrani kao točke križanja.

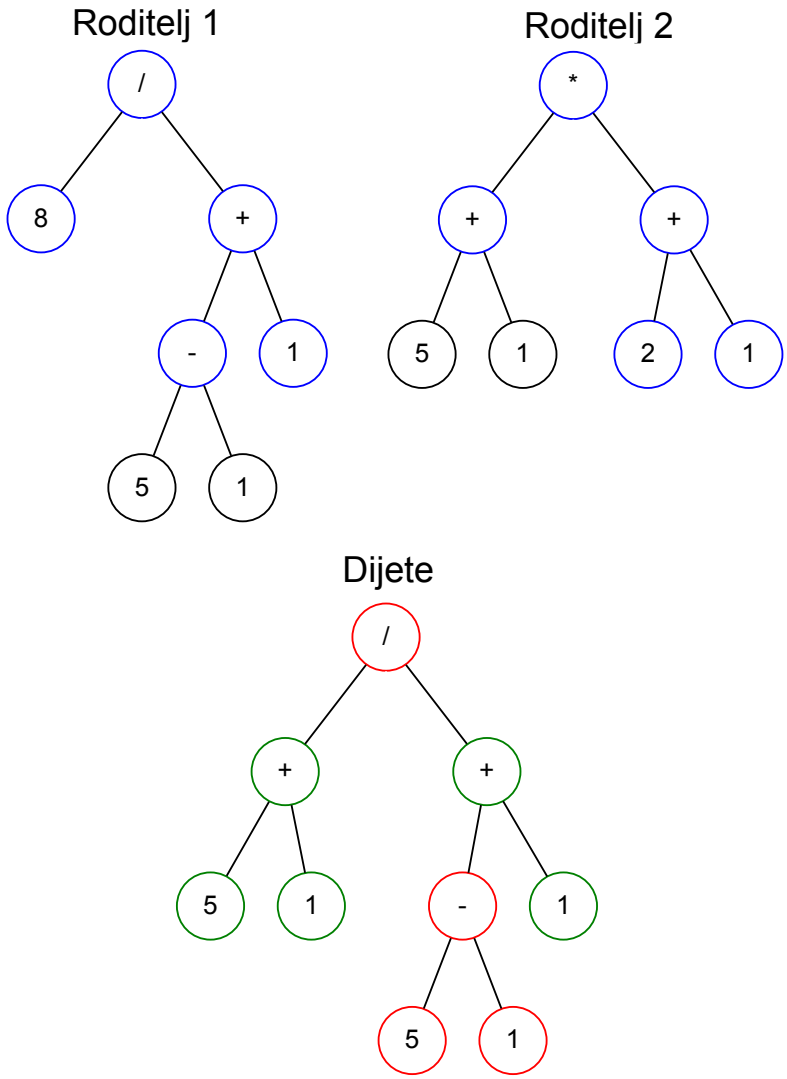


Slika 3.4: Primjer križanja s jednom točkom

3.4.3. Uniformno križanje

Uniformno križanje (engl. *Uniform Crossover*) [21] također najprije određuje zajedničku regiju između jedinki. Nakon toga se za svaki čvor u zajedničkoj regiji nasumično odabire hoće li se on preuzeti od prvog ili od drugog roditelja. U slučaju da se čvor nalazi na granici zajedničke regije i čvor spada u skup funkcijskih čvorova, tada se osim tog čvora preuzimaju i sva podstabla kojima je ovaj čvor korijen. Prednost ovog križanja je ta što ono dozvoljava puno više miješanja genetskog materijala između oba roditelja nego dosad spomenute varijante križanja. Slika 3.5 prikazuje primjer ove vrste križanja. Plavom bojom su kod roditelja označeni čvorovi koji pripadaju

u zajedničku regiju. Kod djeteta, crvenom bojom su označeni čvorovi koji su preuzeti iz prvog roditelja, dok su zelenom bojom označeni čvorovi koji su preuzeti od drugog roditelja.

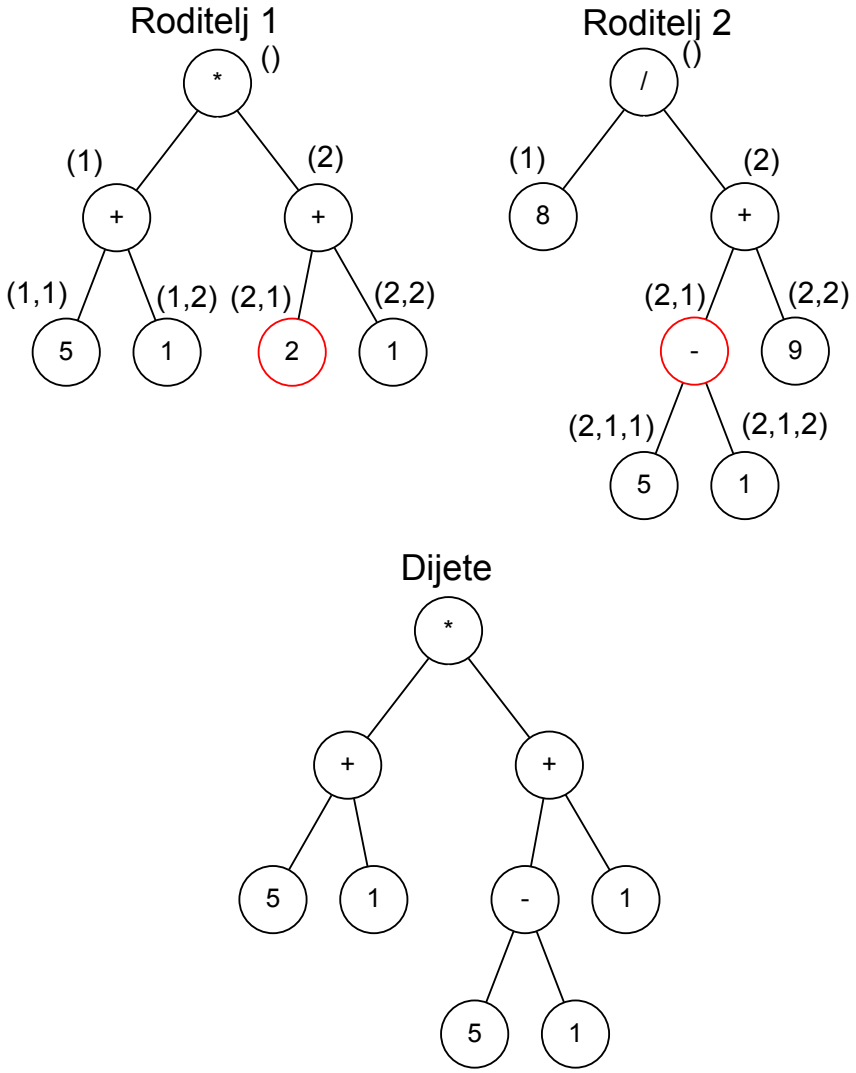


Slika 3.5: Primjer uniformnog križanja

3.4.4. Kontekstno očuvajuće križanje

Kontekstno očuvajuće križanje (engl. *context-preserving crossover*) [1] za razliku od ranije navedenog križanja ne zahtijeva određivanje zajedničke regije, već uvodi novi pojam, a to su koordinate čvorova. Svaki čvor jedinstveno je određen putom od korijena stabla do samog tog čvora. Koordinate čvora se stoga mogu definirati kao n-torka $T = (b_1, b_2, ..., b_n)$ pri čemu je n dubina čvora, a b_i određuje koja grana je odabrana na dubini i (grane se broje s lijeva na desno te obično započinju s brojem jedan). Ko-

rijen stabla obično se označava s praznom n-torkom. Kao točke križanja se mogu odabrati samo oni čvorovi koji imaju jednake koordinate. Kada su odabrani čvorovi u oba roditelja, podstablo prvog roditelja, koje ima korijen u čvoru križanja, zamjenjuje se s podstablom drugog roditelja koje ima korijen u vlastitoj točki križanja. Slika 3.6 prikazuje primjer ove vrste križanja. Kod roditelja, svaki od čvorova ima pridruženu koordinatu, a čvorovi koji predstavljaju točke križanja su označeni crvenom bojom.

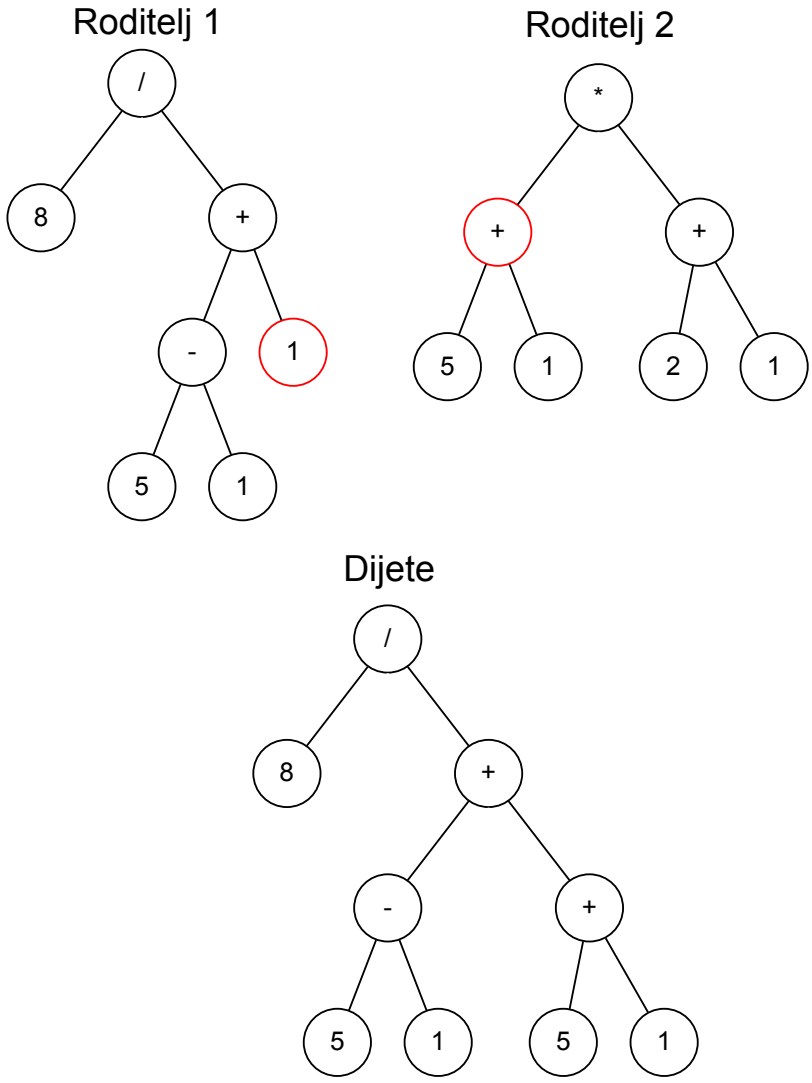


Slika 3.6: Primjer kontekstno očuvajućeg križanja

3.4.5. Veličinski pravedno križanje

Veličinski pravedno križanje (engl. *size-fair crossover*) [15] ne koristi niti zajedničku regiju, niti koordinate čvorova. Kod ovog križanja točka križanja se kod prvog roditelja odabire u potpunosti nasumično kao kod standardnog križanja. Nakon toga se izračuna

veličina podstabla koje će se obrisati iz prvog roditelja. Ta informacija se tada koristi prilikom određivanja točke križanja kod drugog podstabla. Uvjet koji se postavlja jest da veličina podstabla koje ima korijen u točki križanja drugog roditelja ima veličinu koja je manja od $1 + 2 * s$, pri čemu je s veličina podstabla koje će biti obrisano iz prve jedinke. Ovim uvjetom zajamčeno je da podstablo koje će se preuzeti iz drugog roditelja neće biti preveliko u odnosu na originalno podstablo koje je obrisano iz prvog roditelja. Slika 3.7 prikazuje primjer ove vrste križanja. Čvorovi koji predstavljaju točke križanja označeni su crvenom bojom. Kao točka križanja drugog roditelja mogao je biti odabran bilo koji čvor koji zadovoljava uvjet da mu je veličina manja od tri (jer veličina podstabla koje će biti zamijenjeno iz prvog roditelja je jedan, a najveća dopuštena veličina podstabla od drugog roditelja računa se kao $1+2*s$, gdje je s upravo veličina zamijenjenog podstabla, odnosno jedan). Iz toga proizlazi kako je jedini čvor koji nije mogao biti odabran iz drugog roditelja upravo korijenski čvor jer on jedini ima veličinu koja je veća od tri (konkretno, veličina njegovog podstabla je sedam).



Slika 3.7: Primjer veličinski pravednog križanja

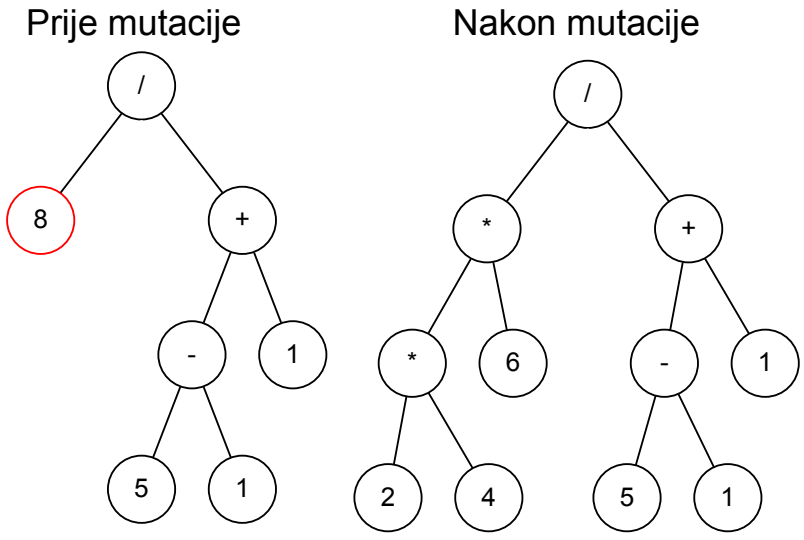
3.5. Mutacija

Mutacija je genetski operator koji nastoji unijeti stohastičke promjene u jednu jedinku, koje bi mogle rezultirati poboljšanjem jedinke. Glavna odlika mutacije jest da ona ima mogućnost da u jedinku uvede genetski materijal koji do tada nije bio prisutan u toj jedinki i time poveća raznovrsnost genetskog materijala. Kako je ranije napomenuto, križanjem nastaju jedinke koje sadrže jednak genetski materijal onom kojeg su imali i roditelji te jedinke. Mutacija može uvesti i onaj genetski materijal koji se nije pojavio u roditeljima, što može biti jako bitno ako je neki genetski materijal jako slabo zastupljen u populaciji jer se time zapravo sprječava njegovo izumiranje i nestajanje. Povećanje genetske raznovrsnosti populacije uvođenjem slučajnih promjena u

jedinke ima vrlo bitnu ulogu u kasnijim fazama genetskog programiranja kada ono već teži nekom rješenju. Naime, spomenuto rješenje često predstavlja lokalni minimum te upravo stohastička svojstva mutacije imaju mogućnost izbaciti algoritam iz istog. Također vrijedi napomenuti kako mutacije, osim što nastoje uvesti novi genetski materijal u jedinke, vrlo često pokušavaju i smanjiti veličinu stabla koje predstavlja jedinka. U nastavku će biti opisano nekoliko različitih operatora mutacije.

3.5.1. Mutacija podstabla

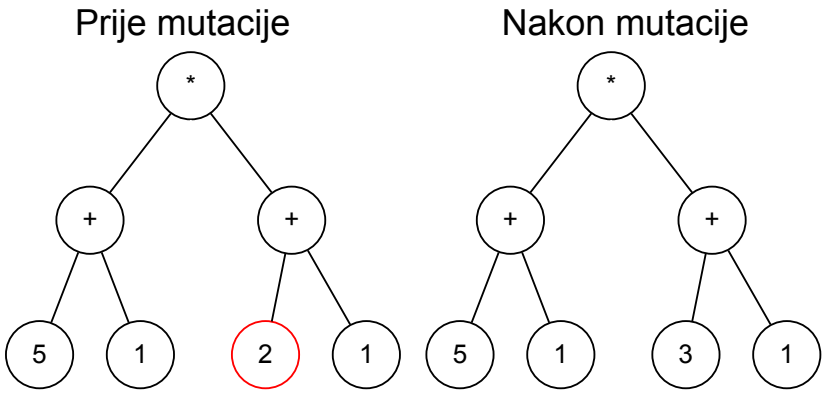
Mutacija podstabla (engl. *subtree mutation*) [21] jedna je od najčešće korištenih mutacija u genetskom programiranju. Ova varijanta mutacije nasumično odabire čvor u stablu i zamjenjuje podstablo koje ima korijen u odabranom čvoru s novim slučajno generiranim podstablom. Ova mutacija se veoma često implementira kao križanje između odabrane jedinke i jedne druge nasumično generirane jedinke, zbog čega se ovaj postupak često naziva križanjem "bezglave kokoši" (engl. *"headless chicken" crossover*). Naravno, pri obavljanju mutacije potrebno je paziti da se ne naruši ograničenje na maksimalnu dubinu stabla. Postoji i inačica mutacije koja sprječava stvaranje novog podstabla ako ono ima više od 15% veću dubinu od podstabla kojeg zamjenjuje. Primjer ove mutacije prikazan je na slici 3.8. Crveno je označen čvor koji predstavlja korijen podstabla koje će mutirati.



Slika 3.8: Primjer mutacije podstabla

3.5.2. Gaussova mutacija

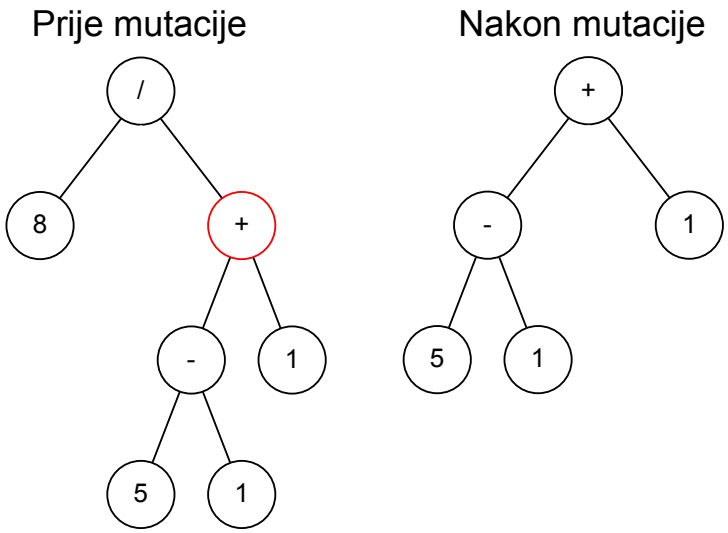
Gaussova mutacija (engl. *Gauss mutation*) [21] vrsta je mutacije koja se može primijeniti samo na terminale koji predstavljaju realne konstante. Ova mutacija na čvorove koji predstavljaju konstante dodaje slučajno generirani šum koji je uzorkovan iz Gaussove normalne razdiobe. Pri tome ova mutacija ima dva parametara koji ujedno predstavljaju i parametre Gaussove distribucije iz koje je taj šum uzorkovan. Prvi parametar predstavlja srednju vrijednost distribucije, dok drugi parametar predstavlja varijancu koja određuje koliko su primjeri zgusnuti oko srednje vrijednosti. Šum se najčešće generira iz distribucije $\mathcal{N}(0, 1)$. Također, ovisno o veličini konstanti koje se koriste u konkretnom problemu dobiveni šum će biti potrebno skalirati na odgovarajuću vrijednost. Slika 3.9 prikazuje primjer ove mutacije. Crveno je označen čvor kojem će biti nadodana slučajno generirana vrijednost (u ovom primjeru pretpostavljeno je da je to vrijednost jedan).



Slika 3.9: Primjer Gaussove mutacije

3.5.3. Podizajuća mutacija

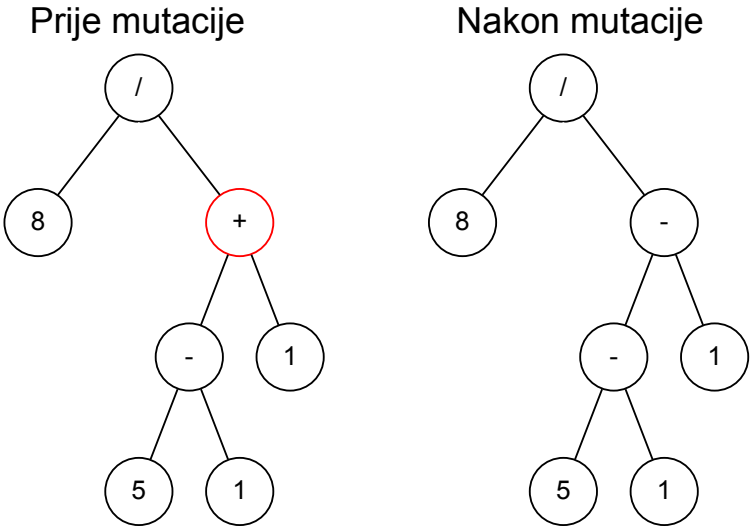
Podizajuća mutacija (engl. *hoist mutation*) [21] vrsta je mutacije kod koje se u jedinku ne uvodi dodatni genetski materijal ako nije prethodno postojao u odabranoj jedinki. Naime ova mutacija jednostavno odabere jedno podstablo iz te jedinke te zamijeni čitavu jedinku tim odabranim podstablom. Iako se možda na prvu ruku ova varijanta mutacije ne čini kao previše korisna jer nema mogućnost uvođenja novog genetskog materijala u jedinku, ona je itekako korisna jer generira jedinke koje su manje od originalne jedinke, i na taj način pokušava doći do jednostavnijih i interpretabilnijih rješenja. Primjer ove mutacije prikazan je na slici 3.10. Crveno je označen čvor koji je odabran da postane novi korijen stabla.



Slika 3.10: Primjer podizajuće mutacije

3.5.4. Komplementirajuća mutacija

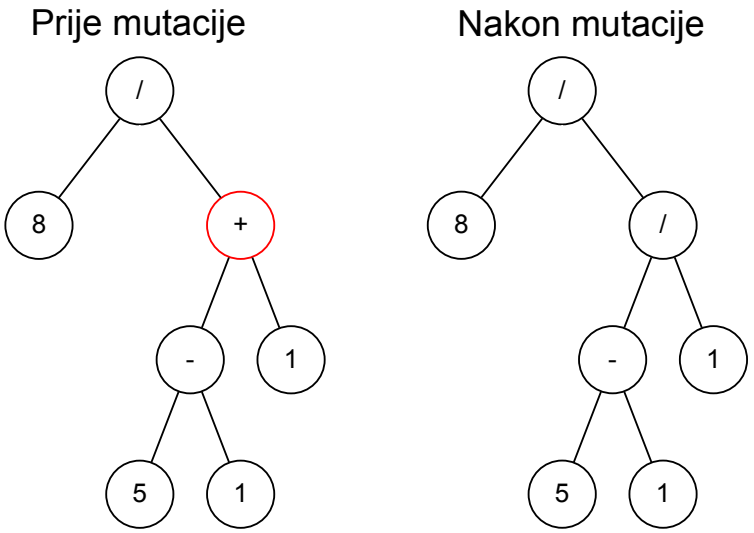
Komplemetirajuća mutacija (engl. *Node complement mutation*) vrsta je mutacije kod koje se odabrani čvor zamjenjuje sa svojim komplementarnim čvorom. Ova mutacija je stoga primjenjiva samo na one čvorove koji imaju definirane komplementarne čvorove. Primjeri nekih komplementarnih funkcija su množenje i dijeljenje, zbrajanje i oduzimanje, minimum i maksimum, i slično. Koje su funkcije i koji terminali će biti međusobno komplementarni može se proizvoljno definirati, no one obavezno moraju imati jednak broj argumenata kako bi se čvorovi mogli međusobno zamijeniti (za terminale je ovo svojstvo inherentno zadovoljeno jer oni nikad nemaju djecu). Slika 3.10 prikazuje primjer ove mutacije. U ovom primjeru pretpostavljeno je da je čvoru koji obavlja zbrajanje komplementarni čvor onaj koji obavlja oduzimanje.



Slika 3.11: Primjer komplementirajuće mutacije

3.5.5. Nadomještajuća mutacija

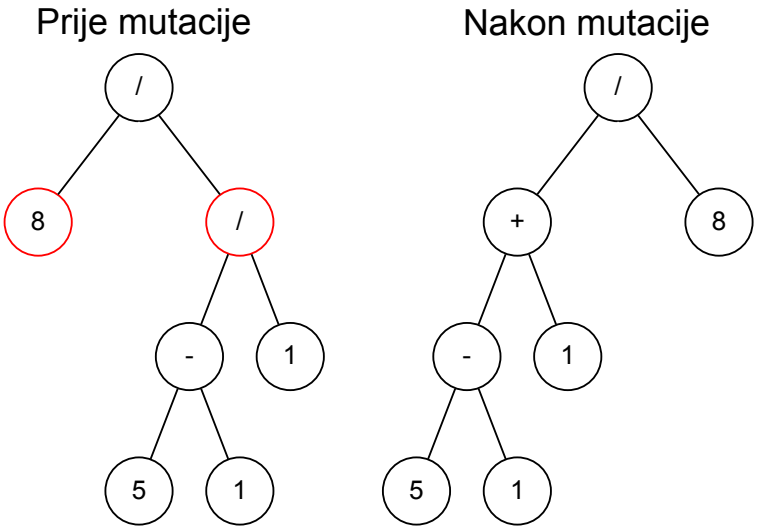
Nadomještajuća mutacija (engl. *node replacement mutation, point mutation*) [21] može se shvatiti kao poopćenje komplementirajuće mutacije. Za razliku od komplementirajuće mutacije gdje je za određene čvorove postojao komplementirajući čvor, ovdje to nije slučaj. Naime, kod ovog tipa mutacije svaki se čvor može zamijeniti za drugim čvorom dok god oba čvora imaju jednak broj djece. Slika 3.12 prikazuje primjer ove mutacije. Na primjeru se može vidjeti kako je slučajno odabrani čvor zamijenjen nekim drugim, slučajno generiranim čvorom koji ima jednak broj operanada.



Slika 3.12: Primjer nadomještajuće mutacije

3.5.6. Permutirajuća mutacija

Permutirajuća mutacija (engl. *permutation mutation*) [21] još je jedna mutacija koja ne uvodi novi genetski materijal u samu jedinku. Ova mutacija nasumično odabire jedan čvor koji sadrži dvoje ili više djece, iz čega slijedi da ova mutacija dakle nije primjenjiva na listove ili čvorove koji sadrže funkcije sa samo jednim argumentom. Nakon što je odabran čvor, on će se mutirati na način da se njegova djeca ispremiješaju. U slučaju čvora sa samo dva djeteta, ta će djeca jednostavno zamijeniti mjesta. Postoji i varijanta ove mutacije koja se naziva zamjenjujuća mutacija (engl. *swap mutation*) koja je primjenjiva samo na binarne nekomutativne funkcijske čvorove. Primjer permutirajuće mutacije može se vidjeti na slici 3.13. Crveno su označeni čvorovi čija će mjesta biti zamijenjena. Treba se uočiti da nakon djelovanje ove mutacije stablo može generirati potpuno isti izlaz ako su zamijenjeni operandi čvora koji predstavlja komutativnu funkciju.

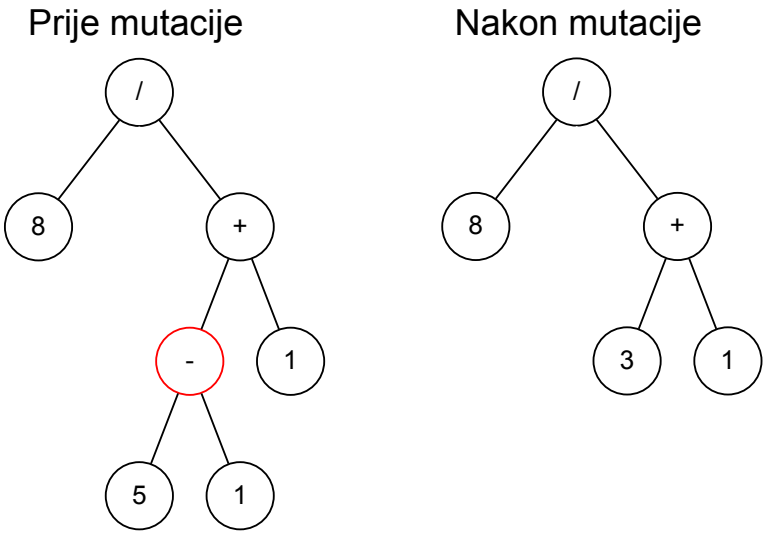


Slika 3.13: Primjer permutirajuće mutacije

3.5.7. Smanjujuća mutacija

Smanjujuća mutacija (engl. *shrink mutation*) [21] vrsta je mutacije koja, osim što uvodi novi genetski materijal u jedinku, nastoji i samu jedinku smanjiti i pojednostaviti. Ideja ove mutacije jest da se nasumično odabere jedan čvor stabla te da se podstablo koje ima korijen u tom čvoru zamijeni sa slučajno odabranim terminalom. Kao što je vidljivo, ova mutacija može djelovati jako destruktivno na stabla, jer ako je odabran neki čvor koji se nalazi pri vrhu stabla, može doći do brisanja većeg dijela stabla. Primjer ove mutacije prikazan je na slici 3.14. Crveno je označen čvor koji predstavlja korijen

podstabla koje će biti zamijenjeno jednim čvorom iz skupa terminalnih čvorova.



Slika 3.14: Primjer smanjujuće mutacije

3.6. Kriterij zaustavljanja

Kako je genetsko programiranje stohastički algoritam koji ne jamči optimalno rješenje, na njega se ne može primijeniti klasičan kriterij zaustavljanja algoritma koji zaustavlja algoritam kada je pronađeno optimalno rješenje. Iz tog razloga razvijeni su posebni kriteriji po kojima se određuje kada je potrebno zaustaviti algoritam. U nastavku poglavlja dat će se kraći pregled najčešće korištenih kriterija zaustavljanja.

Možda najčešće korišten kriterij zaustavljanja jest taj da se algoritam zaustavi kada dostigne određeni broj generacija (ako se radi o generacijskom genetskom programiranju) odnosno iteracija (ako se radi o eliminacijskom genetskom programiranju). Iako je ovaj kriterij veoma jednostavan za implementaciju, on ima jedan veoma bitan nedostatak, a to je da optimalan broj iteracija nije unaprijed poznat. Na prvu ruku se može činiti da je iz tog razloga bolje provesti što veći broj iteracija, odnosno generacija prije zaustavljanja, ali to se u praksi nije pokazalo kao dobro rješenje. Prvi razlog je taj što je izvođenje algoritma genetskog programiranja veoma računski zahtjevno i dugotrajno. Zbog toga se želi izbjeći nepotrebno izvođenje algoritma ako ono ne donosi nikakva značajnija poboljšanja u smislu kvalitete rješenja. Drugi problem koji se može pojaviti kod pretjerano velikog broja generacija jest taj da će algoritam prenaučiti rješenja. Drugim riječima, to znači da će se rješenja prilagoditi onim instancama problema koja su korištena za učenje samog algoritma te će za njih postizati dobre rezultate, dok će se za nove instance problema ponašati dosta lošije (kažemo da rješenja

imaju slabo svojstvo generalizacije). Upravo je iz tog razloga veoma bitno odrediti optimalnu vrijednost ovog kriterija kako dobivena rješenja ne bi bila niti prejednostavna (odnosno podnaučena) niti previše prilagođena instancama koje su se koristile za učenje (odnosno prenučena). Nažalost, ne postoji efikasni način na koji bi se vrijednost ovog parametra mogla odrediti, osim naravno eksperimentiranjem.

Od ostalih kriterija zaustavljanja bitno je još spomenuti i kriterij koji zaustavlja algoritam genetskog programiranja kada u zadanom broju iteracija, odnosno generacija, nije postignuto poboljšanje u dobroti. Smisao ovog kriterija zaustavljanja je sam po sebi jasan - ako kroz određeni broj iteracija nije pronađeno bolje rješenje, tada je teško za očekivati da će algoritam uspjeti pronaći bolje rješenje u bližoj budućnosti (za jako velik broj iteracija će vjerojatno uspjet). Problem je opet jednak, a to je da je eksperimentalno potrebno odrediti optimalan broj iteracija za koje će se algoritam prekinut.

3.7. Parametri algoritma

Nakon što su obrađeni svi osnovni dijelovi genetskog programiranja, vidljivo je kako se ono sastoji od mnogo povezanih elemenata. To također rezultira i činjenicom da genetski algoritam ima mnogo parametara koje je potrebno odrediti. Brojnost, a i priroda samih parametara rezultira odsutnošću jednostavnog načina određivanja/procjene vrijednosti tih parametara unaprijed, bez pokretanja samog algoritma.

Jedan od najintuitivnijih, ali i najbitnijih parametara genetskog programiranja jest veličina populacije. Logično, ona određuje koliko će se jedinki nalaziti u populaciji. Na prvu ruku bi se moglo učiniti kako je bolje imati što veći broj jedinki u populaciji, ali to nije nužno točno. Naime, veći broj jedinki povlači i puno sporiju konvergenciju algoritma, što u konačnici može rezultirati značajnim usporenjem algoritma. S druge strane, algoritam s premalom populacijom sklon je zaglavljivanju u nekom od lokalnih optimuma, zato što početna populacija ne sadrži dovoljno različitog genetskog materijala.

Skup operatora križanja i mutacija koji će genetsko programiranje koristiti također se može smatrati jednim od parametara genetskog programiranja. Naime, pojedini operatori mogu se pokazati mnogo uspješnijim u rješavanju određenih problema od drugih. Stoga je također potrebno odrediti optimalan skup operatora koji postižu najbolje rješenje za dani problem.

Vjerojatnost mutacije jest parametar koji određuje koliko će se često mutacije primjenjivati. Ovo je isto jedan veoma bitan parametar zato što je mutacija jako važan

genetski operator koji uvodi dodatne promjene u jedinke i čuva raznolikost genetskog materijala. Ako se mutacija događa prerijetko, imat će slab utjecaj na algoritam i te u slučaju zapinjanja algoritma u jednom od lokalnih optimuma, teško će se moći izvući iz njega. S druge strane, povećavanjem vjerojatnosti mutacije algoritam se polako pretvara u nasumičnu pretragu s obzirom na to da se u gotovo sva rješenja uvodi dodatna stohastička promjena.

Od ostalih parametara vrijedi spomenuti još veličinu turnira te kriterij zaustavljanja (spomenuti u ranijim poglavljima). Kada se oni svi pribroje (a to nipošto nisu svi parametri algoritma) i kada se uzme u obzir da zapravo ne postoji način kako bi se ti parametri mogli jednostavno odrediti, trivijalno je doći do zaključka kako je zapravo i samo određivanje tih parametara jedan zaseban optimizacijski problem. Kako bi iscrpno pretraživanje svih ovih parametara trajalo jednostavno predugo, prilikom njihovog određivanja najčešće se koriste određene heuristike. One najčešće traže optimalnu vrijednost jednog parametra uz uvjet da su svi ostali parametri fiksirani.

4. Raspoređivanje zasnovano na prilagodljivim pravilima

Raspoređivanje korištenjem prilagodljivih pravila detaljno je opisano u [7] i [8]. Ideja ovog postupka jest da se za raspoređivanje koristi algoritam koji krajnji raspored izgrađuje izravno. No, za razliku od postupaka koji rješenja grade izravno i koji su bili opisani u drugom poglavlju, ovaj postupak ne koristi jednu jedinstvenu funkciju koja određuje prioritete pojedinih poslova. Cilj ovog postupka jest da se, ovisno o kriteriju ocjene rasporeda koji se optimira, koristi različita funkcija za određivanje prioriteta pojedinih poslova. Logično pitanje koje se postavlja jest kako odrediti funkcije koje će se koristiti za pojedine kriterije. Najjednostavnije rješenje jest pokušati intuitivno modelirati takve funkcije, te ih isprobati na određenim ispitnim primjerima. Međutim, takav pristup očito nije veoma poželjan jer je dugotrajan, a i postoje mnogo napredniji, a time i bolji načini generiranja spomenutih funkcija. Do jednog pristupa moguće je doći ako se shvati kako su funkcije koje dodjeljuju prioritet poslovima obične matematičke funkcije. Ako se tako modelira i prioritetna funkcija, onda je jasno kako je za generiranje takvih funkcija moguće koristiti algoritam genetskog programiranja. Genetskim programiranjem se tada za svaki traženi kriterij ocjene rasporeda generira posebna prioritetna funkcija koja će se koristiti prilikom izrade rasporeda koji treba minimizirati dani kriterij. Na taj način dobit će se funkcije koje su usko specijalizirane za izradu rasporeda s obzirom na dani kriterij i tako će se postići bolji rezultat za dani kriterij nego što bi se dobio korištenjem klasičnih postupaka izgradnje rješenja izravno [9] [10] [18] [20].

Generiranje funkcije koja će se koristiti za određivanje prioriteta za strojeve i poslove sama po sebi nije dovoljna za proces raspoređivanja. Osim navedene funkcije potreban je još i određeni meta-algoritam koji će korištenjem generirane funkcije provesti raspoređivanje poslova na strojeve. Algoritam 1 prikazuje pseudokod meta-algoritma koji se koristi u trenutnoj inačici raspoređivanja temeljenog na prilagodljivim pravilima.

Algoritam 1 Meta-algoritam za raspoređivanje zasnovano na prilagodljivim pravilima

```
1: while postoje neraspoređeni poslovi do
2:     čekaj dok posao ne postane dostupan ili se posao završi;
3:     for sve dostupne poslove i sve strojeve do
4:         izračunaj prioritet π_ij posla j na stroju i;
5:     end for
6:     for sve dostupne poslove do
7:         odredi najbolji stroj (onaj za koji se postiže najbolji iznos prioriteta π_ij);
8:     end for
9:     while postoje poslovi čiji je najbolji stroj dostupan do
10:        odredi najbolji prioritet od svih takvih poslova;
11:        rasporedi posao s najboljim prioritetom;
12:    end while
13: end while
```

Kako se optimizacije koje će biti opisane u idućem poglavlju temelje upravo na postupku raspoređivanja korištenjem prilagodljivih pravila, u nastavku će se ukratko opisati neki osnovni elementi ovog postupka.

4.1. Parametri algoritma genetskog programiranja

Određivanje parametara genetskog algoritma obavljeno je u sklopu ovog rada. Međutim, radi preglednosti, ovdje će biti nabrojani samo oni parametri koji se nisu optimirali, već su bili fiksirani. Konkretne vrijednosti parametara koji su bili optimirani bit će prikazani u idućem poglavlju. Tablica 4.1 prikazuje navedene parametre algoritma.

Tablica 4.1: Parametri za algoritam genetskog programiranja

parametar	vrijednost
način odabira	eliminacijski
operator odabira	turnirski odabir
veličina turnira	3
uvjet zaustavljanja	maksimalni broj generacija
inicijalizacija	*ramped half-and-half*

4.2. Funkcije cilja

Kako bi se mogla ocijeniti učinkovitost razvijenih pravila, potrebno je definirati funkciju cilja koja će svakom pravilu, odnosno jedinki pridružiti odgovarajuću dobrotu. Oblik funkcije cilja ovisit će dakako o konkretnom kriteriju koji se optimira. Iako se na prvi pogled može činiti kako iznos funkcije cilja može biti jednak vrijednosti kriterija kojeg se optimira, to nažalost u praksi nije poželjno. Razlog tome je što se ukupna dobrota jedinke određuje na temelju funkcija ciljeva svih korištenih ispitnih primjera. Također treba napomenuti da se javlja još jedan problem. Naime, unatoč tome što je poželjno imati što raznovrsnije ispitne primjere, takvi primjeri mogu utjecati i na povećanje razlike iznosa kriterija ocjene koja će u ovom slučaju biti veoma velika. Drugim riječima, to znači da na konačni rezultat pojedini ispitni primjeri utječu više od drugih. To svojstvo nije poželjno, već bi htjeli da svaki primjer jednakim intenzitetom utječe na konačnu dobrotu jedinke. To se može postići tako da se vrijednosti kriterija ocjene normiraju, tako da svi utječu na iznos konačne dobrote jednakim intenzitetom.

Prije nego što se opišu konkretne funkcije cilja, potrebno je uvesti dodatne veličine koje će se koristiti za normiranje. Tako je broj poslova u primjeru predstavljen s n, srednja vrijednost težina svih poslova s $\bar{w}$, a srednje trajanje poslova s $\bar{p}$. Za ocjenu kvalitete jedinki koristit će se sljedeće četiri funkcije cilja:

- za kriterij težinskog zaostajanja:
$$f_i = \frac{\sum_{j=1}^{n} w_j T_j}{n * \bar{w} * \bar{p}}$$
- za kriterij težinskog broja zakašnjelih poslova (odnosno težinske zakašnjelosti):
$$f_i = \frac{\sum_{j=1}^{n} w_j U_j}{n * \bar{w}}$$
- za kriterij težinskog protjecanja:
$$f_i = \frac{\sum_{j=1}^{n} w_j F_j}{n * \bar{w} * \bar{p}}$$
- za kriterij ukupne duljine rasporeda:
$$f_i = \frac{max\{C_j\}}{n * \bar{p}}$$

Ukupna funkcija dobrote se tada definira kao zbroj iznosa funkcije cilja za svaki pojedini ispitni primjer:

$$F = \sum_i f_i.$$

4.3. Skup primitiva

U ovom poglavlju dat će se kraći opis primitiva koji će se koristiti u genetskom programiranju za izradu rješenja. Ovaj skup primitiva će se dodatno proširivati u pojedinim optimizacijama, no skup primitiva koji će se ovdje opisati predstavlja osnovni skup kojeg će koristiti sve optimizacije.

4.3.1. Skup funkcijskih čvorova

Osnovna inačica algoritma raspoređivanja na temelju prilagodljivih pravila koristi funkcijske čvorove koji su navedeni u tablici 4.2. Iako se radi o malom skupu funkcijskih čvorova, nužno je istaknuti kako se radi o dovoljno ekspresivnom skupu. Svi čvorovi su manje-više intuitivno jasni, dok je jedino operator dijeljenja pomalo specifičan. Ova vrsta dijeljenja koja se koristi se često naziva zaštićeno dijeljenje. Naime, problem kod regularnog dijeljenja jest da ono nije definirano kada je djelitelj jednak nuli. Kako bi se tome doskočilo, moguće je operaciju dijeljenja prilagoditi na način da se u situaciji dijeljenja s nulom vrati neka unaprijed određena vrijednost. Ovakva definicija operacije dijeljenja često nije poželjna jer uvodi prekide u funkciju. Međutim, dobra strana ovog pristupa jest ta da je implementacija ove operacije vrlo jednostavna. Postoje i bolja rješenja ovog problema (primjerice korištenje intervalne aritmetike [13]), ali ona se ovdje neće razmatrati.

Tablica 4.2: Funkcijski čvorovi

Oznaka čvora	Opis
ADD	binarni operator zbrajanja
SUB	binarni operator oduzimanja
MUL	binarni operator množenja
DIV	binarni operator dijeljenja: $DIV(a,b) = \begin{cases} 1, \text{ ako } \lvert b \rvert < 0.000001 \\ \frac{a}{b}, \text{inače} \end{cases}$
POS	unarni operator: $POS(a) = max\{a, 0\}$

4.3.2. Skup terminalnih čvorova

Terminalni čvorovi, kao što je već ranije navedeno, predstavljaju određene ulazne varijable u prioritetnu funkciju koja se evoluira. U ovom slučaju te varijable predstavljaju stanje sustava u određenom vremenskom trenutku. Tablica 4.3 predstavlja popis svih

osnovnih terminala koji se koriste. U opisu pojedinih terminala može se uočiti varijabla $time$ koja predstavlja trenutno vrijeme u kojem se sustav nalazi (koja povlači da su ti terminali vremenski ovisni). Međutim, treba napomenuti da nije nužno uvijek koristiti sve terminale prilikom evolucije. Primjerice, funkcije cilja koja optimira težinsko protjecanje ili ukupnu duljinu rasporeda ne zahtijevaju terminale dd, SL niti w. S druge strane, kod funkcije cilja koja optimira težinsko zaostajanje i funkcije cilja koja optimira težinsku zakašnjelost koriste se svi navedeni terminali.

Tablica 4.3: Terminalni čvorovi

Oznaka čvora	Opis
pt	trajanje izvođenja posla na promatranome stroju (p_{ij})
dd	željeno vrijeme završetka (d_j)
w	težina (w_j)
SL	dopuštena odgoda uz brzinu dotičnog stroja: $max\{d_j - p_{ij} - time, 0\}$
pmin	najkraće trajanje izvođenja posla (za sve postojeće strojeve)
pavg	srednje trajanje izvođenja posla
PAT	strpljenje (engl. *patience*) - količina vremena za koju će stroj koji za promatrani posao daje najkraće trajanje izvođenja biti raspoloživ
MR	količina vremena do raspoloživosti promatranog stroja (engl. *machine ready*)
age	vrijeme koje je posao proveo u sustavu: $time - r_j$

5. Optimizacije

U ovom poglavlju opisat će se različiti postupci koji su bili korišteni za poboljšanje postupka raspoređivanja zasnovanim na prilagodljivim pravilima, opisanog u prošlom poglavlju. U nastavku će biti opisane samo ideje pojedinih postupaka optimizacije, dok će dobiveni rezultati biti detaljno prikazani u idućem poglavlju.

5.1. Optimizacija parametara

Prvi, i možda najvažniji korak u optimizaciji algoritma genetskog programiranja jest pronaći što bolje parametre algoritma. Kako efikasnost algoritma genetskog programiranja uvelike ovisi o parametrima s kojima je algoritam pokrenut, važno je odrediti optimalan skup parametara. U sklopu ovog rada optimirani su sljedeći parametri genetskog programiranja: veličina populacije, broj generacija, dubina stabla, vjerojatnost mutacije, skup genetskih operatora križanja i mutacija. Kao što je vidljivo radi se o dosta širokom skupu parametara i iscrpno ispitivanje svih mogućih vrijednosti i kombinacija ovih parametara naprosto je nemoguće. Iz tog razloga, optimiran je parametar po parametar na način da kada se određuje optimalna vrijednost za jedan parametar, vrijednosti svih ostalih parametara se fiksiraju. Naravno da ovaj način određivanja parametara ne garantira da će biti pronađene optimalne vrijednosti parametara, ali pomoću ovog načina moguće je odrediti vrijednosti parametara koje daju veoma dobre rezultate.

Prvi parametri koji su bili određeni su veličina populacije i broj generacija. Iznosi ova dva parametra određeni su istodobno na način da je pokrenuto više eksperimenata koji su se izvodili veći broj generacija i koji su se razlikovali samo po broju jedinki u populaciji. Svakih nekoliko generacija za svaki eksperiment je uzorkovana vrijednost dobrote najbolje jedinke u populaciji. Na taj način dobiveno je kretanje dobrote najbolje jedinke za populacije s različitim brojem jedinki. Ta informacija je onda iskorištena kako bi se odredili optimalni iznosi navedenih parametara. Nakon analize eksperimenata ustanovljeno je kako je veličina populacije od 1000 jedinki najisplativija. Iako se

s više jedinki postižu ponešto bolji rezultati, razlike nisu toliko značajne da bi korištenje veće populacije bilo isplativo. S druge strane, parametar koji određuje maksimalan broj generacija postavljen je na iznos od 80 generacija. Iako se za veći broj generacija u pojedinim slučajevima mogu postići bolji rezultati, to poboljšanje je gotovo zanemarivo. Iz tog razloga bolje je uzeti manju populacije i izvesti manji broj generacija te dobiti rješenje u puno kraćem vremenskom periodu. Ta vremenska komponenta više dolazi do izražaja jer je algoritam, zbog svoje stohastičke prirode, potrebno pokrenuti nekoliko puta kako bi se našla što bolja rješenja.

Idući parametar koji je bio optimiziran jest maksimalna dubina stabla. Ovo je veoma važan parametar o kojem ovise i brzina konvergencije algoritma, dobrota jedinki, kao i složenost, odnosno interpretabilnost rješenja. Na prvu pomisao može se činiti kako će stabla koja imaju veću dubinu zapravo predstavljati i bolja rješenja. Navedena pretpostavka se kroz eksperimente pokazala netočnom. Naime, pokazano je kako stabla manje dubine postižu mnogo bolje rezultate. Iako je ovo svojstvo pomalo iznenađujuće, ono je veoma poželjno jer su manja stabla iz mnogobrojnih razloga bolja. Konačna maksimalna dubina koja je odabrana jest pet. Stabla koja su bila veća od ove dubine u prosjeku su davala dosta lošija rješenja, a kako veća dubina stabala ne nudi nikakve prednosti, već samo nedostatke, nije bilo razloga koji bi pogodovali izboru većeg iznosa za maksimalnu dubinu stabla.

Vjerojatnost mutacije jest idući parametar koji je bio optimiran. Važno je dobro podesiti ovaj parametar jer će se za preveliku vrijednost vjerojatnosti mutacije uvoditi prevelike promjene u populaciju, dok za premalenu vrijednost promjene će se događati prerijetko da bi značajno utjecale na konačno rješenje. Kroz eksperimente se kao optimalna vrijednost pokazao iznos vjerojatnosti mutacije od 0.3.

Konačno, bilo je potrebno odrediti i optimalan skup genetskih operatora križanja i mutacija. Dostupni su svi operatori križanja i mutacije koji su bili opisani u poglavlju o genetskom programiranju. Intuitivno je jasno da je nemoguće isprobati sve moguće kombinacije ovih operatora iz razloga što bi to bilo vremenski prezahtjevno. Iz tog razloga bilo je potrebno upotrijebiti dva heuristička postupka kako bi se odredili ti optimalni skupovi. U tu svrhu korištene su takozvane "izgrađujuće" i "razgrađujuće" heuristike. One nisu ograničene samo na određivanje optimalnog skupa genetskih operatora, već se mogu koristiti za određivanje optimalnog skupa i drugih vrsta parametara. U sljedeća dva paragrafa opisat će se ideja ovih dviju heuristika (konkretno za određivanje optimalnog skupa operatora).

"Izgrađujuće" heuristike, kao što im i samo ime kaže, pokušavaju izgraditi optimalan skup operatora. Ovaj tip heuristike koristi dva skupa, skup korištenih operatora i

skup nekorištenih operatora. Skup korištenih operatora predstavlja one operatore koje algoritam tijekom izvođenja koristi, dok skup nekorištenih operatora predstavlja dakako one operatore koje algoritam trenutno ne koristi. Općenito ova heuristika kreće s praznim skupom korištenih operatora, no može krenuti i s nekim predefiniranim skupom. Ideja heuristike je da se jedan operator iz skupa nekorištenih operatora doda u skup korištenih operatora. Za ovakav uvećani skup tada se izračuna prosječna dobrota algoritma, te se ona pohrani. Nakon toga se taj isti operator ponovo vrati u skup nekorištenih operatora. Ovaj postupak se ponovi za svaki pojedini operator u skupu nekorištenih operatora. Kada je ovaj korak obavljen, odredi se za koji je operator postignuta najbolja prosječna dobrota algoritma. Ovaj operator postaje kandidat za ubacivanje u skup korištenih operatora. No prije nego što bude ubačen u taj skup mora zadovoljiti još jedan kriterij, koji zahtijeva da algoritam koristeći skup operatora u koje je uključen odabrani operator postiže bolju prosječnu dobrotu od skupa kada taj odabrani operator nije uključen. Na taj način se operator dodaje u skup ako i samo ako on u konačnici poboljšava rad algoritma. Postupak se ponavlja sve dok u jednoj iteraciji nije dodan niti jedan operator u skup korištenih operatora, bilo iz razloga što su već dodani svi operatori ili što operator nije zadovoljio potrebni uvjet za dodavanje.

"Razgrađujuće" heuristike djeluju upravo suprotno od "izgrađujućih" heuristika, odnosno nastoje "razgraditi" neki veliki skup na mnogo jednostavniji skup. Ova heuristika također koristi skup korištenih i skup nekorištenih operatora. Za razliku od "izgrađujuće" heuristike, ideja ove heuristike jest da su u početku algoritma svi operatori uključeni u skup korištenih operatora, dok je skup nekorištenih operatora prazan. Zatim se odabire jedan operator iz skupa korištenih operatora te se stavlja u skup nekorištenih operatora. Za tako dobiveni, "umanjeni", skup korištenih operatora izračuna se srednja vrijednost dobrote dobivenih rješenja te se taj iznos pohrani. Nakon toga se operator koji je bio premješten iz skupa korištenih operatora u skup nekorištenih operatora ponovo vrati u skup korištenih operatora. Ovaj postupak se ponovi za svaki operator u skupu korištenih operatora. Može biti postavljeno i ograničenje na skup korištenih operatora na način da se određeni operatori nikada ne smiju izbaciti iz tog skupa, što povlači da se onda oni jednostavno ignoriraju od strane algoritma. Nakon što je ovaj korak obavljen za sve operatore odredi se onaj operator čije je izbacivanje iz skupa korištenih operatora rezultiralo najvećom prosječnom dobrotom. Ovaj operator postavlja se kao kandidat za izbacivanje. No prije nego što se navedeni operator može izbaciti mora se dodatno provjerit da li algoritam za skup operatora bez tog čvora postiže bolju prosječnu dobrotu od algoritma koji koristi skup operatora u koji je uključen navedeni čvor. Ovime se sprječava izbacivanje operatora ukoliko to neće rezultirati

poboljšanjem u radu algoritma. Postupak se ponavlja sve dok u jednoj iteraciji nije izbačen niti jedan operator iz skupa korištenih operatora, bilo iz razloga što su već izbačeni svi operatori ili što operator nije zadovoljio potrebni uvjet za izbacivanje.

Iz prethodno navedenih opisa ovih heuristika vidljivo je kako se radi o takozvanim pohlepnim (engl. *greedy*) postupcima. Iz tog razloga rješenje dobiveno ovim postupcima ne mora nužno biti i optimalno rješenje. No bez obzira na navedeno ograničenje, ove heuristike mogu odrediti dosta dobar skup operatora koje bi algoritam trebao koristiti, i to u prihvatljivom vremenu. Prilikom određivanja optimalnog skupa operatora (a i ostalih parametara u nastavku) korištene su obje heuristike kako bi se povećala mogućnost pronalaženja što boljeg skupa operatora. Popis genetskih operatora križanja i mutacije koji su uključeni u skup korištenih operatora nabrojani su u tablici 5.1 gdje su prikazani i ostali operatori.

Konačni skup parametara koji je bio određen i korišten za sve eksperimente, ako to nije drugačije napomenuto, jest onaj prikazan u tablici 5.1 (uključivo s parametrima koji nisu bili optimirani).

Tablica 5.1: Parametri za algoritam genetskog programiranja

Parametar	Vrijednost
veličina populacije	1000
uvjet zaustavljanja	maksimalni broj generacija
maksimalni broj generacija	80
način odabira	eliminacijski
operator odabira	turnirski odabir
veličina turnira	3
inicijalizacija	*ramped half-and-half*
vjerojatnost mutacije	0.3
maksimalna dubina stabla	5
skup operatora križanja	križanje podstable, uniformno križanje, kontekstno očuvajuće križanje, veličinski pravedno križanje
skup operatora mutacije	mutacija podstabla, gaussova mutacija, podizajuća mutacija, komplementirajuća mutacija, nadomještajuća mutacija, permutirajuća mutacija, smanjujuća mutacija

5.2. Proširenje skupa funkcijskih čvorova

Kao što je ranije napomenuto, trenutna inačica raspoređivanja temeljenog na prilagodljivim pravilima koristi jako malen skup osnovnih matematičkih operatora. Pitanje koje se logično nameće jest da li se proširivanjem skupa funkcijskih čvorova mogu postići bolji rezultati. Dodavanjem novih funkcijskih čvorova povećava se ekspresivnost rješenja koja se generiraju algoritmom genetskog programiranja. Nažalost, dodavanje novih funkcijskih čvorova ne dolazi bez posljedica. Glavni nedostatak jest da se povećava prostor koji algoritam pretražuje. To posljedično može uzrokovati sporiju konvergenciju algoritma ili povećati vjerojatnost zapinjanja u nekom lokalnom optimumu. Iz tog razloga nije svako dodavanje novih funkcijskih čvorova nužno dobro i ne mora nužno rezultirati poboljšanjem rezultata. Iz tog razloga potrebno je, nakon što su novi funkcijski čvorovi uključeni u sustav, odrediti i optimalan skup čvorova za koje se postižu najbolji rezultati. U nastavku će se za početak opisati novi čvorovi koji su dodani u sustav te će se nakon toga opisati postupak kako je od skupa svih funkcijskih čvorova određen optimalan skup čvorova.

Tablica 5.2 prikazuje skup novododanih čvorova. Iako je iz tablice jasno kako funkcioniraju pojedini čvorovi, u nastavku će se dati kratki opis svakog pojedinog čvora te kratka argumentacija zašto je odlučeno uvesti upravo te čvorove.

Tablica 5.2: Novododani funkcijski čvorovi

Oznaka čvora	Opis
IFGT	kvartarni operator grananja: $IFGT(a,b,c,d) = \begin{cases} c, \text{ ako } a > b \\ d, \text{inače} \end{cases}$
IFLT	kvartarni operator grananja: $IFLT(a,b,c,d) = \begin{cases} c, \text{ ako } a < b \\ d, \text{inače} \end{cases}$
MAX	binarni operator maksimalne vrijednosti: $MAX(a,b) = \begin{cases} a, \text{ ako } a > b \\ b, \text{inače} \end{cases}$
MIN	binarni operator minimalne vrijednosti: $MIN(a,b) = \begin{cases} a, \text{ ako } a < b \\ b, \text{inače} \end{cases}$
SQRT	unarni operator korjenovanja: $SQRT(a) = \begin{cases} 1, \text{ ako } \lvert a \rvert < 0 \\ \sqrt{a}, \text{inače} \end{cases}$
AVG	bininarni operator srednje vrijednosti: $AVG(a,b) = (a+b)/2$
ABS	unarni operator apsolutne vrijednosti: $ABS(a) = \lvert a \rvert$

Prve dvije funkcije koje su prikazane u tablici su funkcije $IFGT$ i $IFLT$. One

zapravo predstavljaju pokušaj modeliranja $if...else$ strukture grananja u obliku stabla. To je ostvareno na način da prva dva podstabla zapravo predstavljaju argumente uvjeta koji se ispituje, dok druga dva podstabla predstavljaju povratnu vrijednost. Koji od ta dva čvora će zapravo biti odabran za povratnu vrijednost ovisi dakako o tome je li uvjet ispunjen ili nije. Ako je uvjet ispunjen vraća se prvo od ta dva podstabla (odnosno treće u ukupnom poretku), u suprotnom vraća se drugo (odnosno četvrto u ukupnom poretku). Kod $IFGT$ funkcije uvjet je zadovoljen ako je vrijednost koju vrati prvo podstablo veće od vrijednosti koju vrati drugo podstablo. S druge strane kod $IFLT$ funkcije uvjet je zadovoljen ako je vrijednost koju vrati prvo podstablo manja od vrijednosti koju vrati drugo podstablo. Motivacija koja stoji iza uvođenja ovih čvorova jest ta da oni omogućuju izvođenje određenog podstabla ovisno o određenom uvjetu, što posljedično omogućuje da se pojedini dijelovi stabla specijaliziraju za različite slučajeve. Na taj način se uvelike povećava ekspresivnost rješenja. Nažalost, navedena dva funkcijska čvora dolaze s jednim velikim nedostatkom, a to je broj djece koji oni imaju. Njihov poveći broj djece uzrokuje povećanje veličine stabla odnosno broj čvorova koje stablo sadrži. To nije nimalo poželjno svojstvo jer uzrokuje sporiju konvergenciju algoritma i smanjuje interpretabilnost rješenja.

MAX i MIN funkcijski čvorovi predstavljaju istoimene matematičke funkcije max i min. Ovi čvorovi se zapravo mogu shvatiti kao pojednostavljene verzije $IFGT$ i $IFLT$ čvorova koji vraćaju najveći, odnosno najmanji argument. Navedeni čvorovi namjerno su ograničeni na samo dva djeteta, upravo iz razloga kako bi se spriječio problem koji je opisan u prethodnom paragrafu kod $IFGT$ i $IFLT$ čvorova. Naravno, broj djece ovih čvorova nije ograničen, jer semantika čvorova ostaje ista bez obzira na broj djece koji imaju.

SQR je funkcijski čvor koji predstavlja operator zaštićenog korjenovanja. Korjenovanje je naime operator koji nije definiran za negativne brojeve. Iz tog razloga potrebno je osigurati da se vrati određena predefinirana vrijednost u slučaju da je argument ovog operatora negativan broj. Upravo iz tog razloga se ovaj operator i naziva zaštićenim operatorom korjenovanja. Naravno da ovaj postupak ne dolazi bez određenih nedostataka. Prethodno opisanim postupkom se zapravo uvode skokovi i prekidi u konačnu funkciju. Oba navedena svojstva su veoma nepoželjna, jer otežavaju rad algoritma, stoga je preporučljivo izbjegavati ovakve metode, te koristiti sofisticiranije načine određivanja jesu li rješenja u potpunosti definirana za ulazni interval vrijednosti.

AVG funkcija jednostavno određuje srednju vrijednost između svojih argumenata. Opet zbog prethodnih razloga i ovaj operator je ograničen na samo dva argumenta, iako je primjenjiv na neodređen broj argumenata. Na prvi pogled se zapravo čini kako

se radi o suvišnoj funkciji jer algoritmu genetskog programiranja su na raspolaganju osnovni matematički operatori pomoću kojih se može izračunati srednja vrijednost. No motivacija uvođenja ovog čvora nalazi se upravo u tome da se algoritmu olakša izgradnja funkcije srednje vrijednosti, odnosno da za to koristi gotov funkcijski čvor, umjesto da potroši određen broj iteracija kako bi izgradio tu funkciju. Čak i ako algoritam uspije izgraditi tu funkciju, ne postoji nikakva garancija da genetski operatori neće promijeniti jedinku upravo na tom mjestu gdje je izgradila taj operator. Iz upravo navedenih razloga može se vidjeti da definicija srednje vrijednosti kao atomarne jedinice u obliku funkcijskog čvora ima smisla.

ABS je funkcijski čvor koji računa apsolutnu vrijednost argumenta. Sličan čvor, pod nazivom POS već postoji ugrađen u sustav. Doduše taj čvor za negativne vrijednosti vraća isključivo 0. Upravo zbog toga je taj čvor veoma destruktivan, jer u potpunosti zanemaruje negativne vrijednosti. ABS je po tom pitanju nešto slobodniji. On zadržava vrijednost negativnog argumenta i samo mu promijeni predznak. U ovom trenutku teško je pretpostaviti koji će od ova dva čvora više koristiti algoritmu genetskog programiranja, već će se to pokazati u nastavku tijekom testiranja.

Nažalost, ne postoji garancija da će se uključivanjem svih ovih funkcijskih čvorova u sustav stvarno dobiti poboljšanje algoritma. Iako se dodavanjem svakog novog čvora u sustav povećava njegova ekspresivnost, također se povećava i prostor pretraživanja algoritma. Iz tog razloga uvođenje novih čvorova u sustav ne mora uvijek rezultirati dobivanjem boljih rješenja. Zato je potrebno napraviti odabir čvorova za koje se postižu najbolji rezultati. Iscrpno pretraživanje svih mogućih kombinacija čvorova dakako nije primjenjivo jer je broj kombinacija koje bi bilo potrebno isprobati prevelik već i za ovako malen broj čvorova. Iz tog razloga, za određivanje optimalnog skupa funkcijskih čvorova, korištene su heurističke metode analogne metodama koje su bile korištene za određivanje optimalnog skupa genetskih operatora. Iz tog razloga se ovdje neće ponovo detaljno opisivati navedeni postupci, već će se samo ukratko istaknuti pojedine specifičnosti njihove primjene za pronalazak optimalnog skupa funkcijskih čvorova.

"Izgrađujuća" heuristika pri izgradnji optimalnog skupa funkcijskih čvorova ne kreće od praznog skupa korištenih čvorova, već se kreće sa skupom koji sadrži osnovne matematičke operatore: zbrajanje, oduzimanje, dijeljenje i množenje. Razlog tome je što nema smisla krenuti od praznog skupa funkcijskih čvorova jer bi se tada jedinke sastojale samo od jednog terminalnog čvora, što je svakako besmisleno. Slična ideja je ugrađena i u "razgrađujuću" heuristiku. Ona može izbaciti bilo koji čvor osim ranije navedenih matematičkih operatori. Oni u ovoj heuristici također čine temeljni dio skupa koji se ne provjerava.

Konačni skup operatora je određen na način da su iskorištene obje navedene heuristike te je od oba dobivena skupa odabran onaj bolji.

5.3. Semantičko genetsko programiranje

Semantičko genetsko programiranje [14] je podvrsta genetskog programiranja koje, osim što zahtijeva da su stabla sintaksno ispravna, zahtijeva i semantičku ispravnost stabla.

Sintaksna ispravnost stabla garantira da stablo ima ispravnu strukturu i da se može prevesti u ispravnu matematičku formulu ili program. Primjerice, to znači da svaki operator ima točno zadan broj operanada i da se ne smije pojaviti situacija da operator ima broj operanada koji je različit od tog broja. Naravno ovo je najjednostavniji primjer, mogu postojati i neki drugi uvjeti koji moraju biti zadovoljeni kako bi stabla bila sintaksno ispravna. Dakle, može se zaključiti da je sintaksna ispravnost stabla uvjet kako bi se jedinke uopće mogle evaluirati. Iz tog razloga uopće nema smisla generirati sintaksno neispravna rješenja jer se njih uopće neće moći evaluirati, stoga je uvjet dobrog funkcioniranja genetskog programiranja da sve jedinke predstavljaju sintaksno ispravna rješenja.

S druge strane, semantička ispravnost stabla zahtijeva da je izraz koje stablo predstavlja ispravan i u samoj domeni problema. Svaki čvor dodatno sadrži i određenu semantičku informaciju, a to je najčešće pripadna mjerna jedinica čvora. No, uvođenje semantičke informacije, odnosno mjernih jedinica, samo po sebi nije dovoljno, već je potrebno uvesti i dodatna ograničenja na operatore kako bi se izgradilo semantički ispravno stablo. Primjer jednog semantičkog pravila jest da operator zbrajanja "+" može obavljati operaciju samo nad čvorovima koji imaju ekvivalentnu semantičku informaciju (odnosno mjernu jedinicu). Na taj način mogu se dobiti semantički ispravna stabla, koja su slična izrazima iz stvarnog svijeta, gdje mjerne jedinice određuju koje se operacije mogu obavljati nad pojedinim izrazima. Semantička ispravnost stabla, za razliku od sintaksne, u genetskom programiranju nije uvjet za ispravno funkcioniranje samog algoritma.

Motivacija koja stoji iza semantičkog programiranja jest ta da klasično genetsko programiranje generira potpuno nasumična stabla koja mogu predstavljati fizikalno neispravne izraze. Nasuprot tome, semantičko programiranje garantira da će konačni izraz predstavljati potpuno ispravni fizikalni izraz. Izrazi dobiveni semantičkim genetskim programiranjem mogu iz tog razloga biti puno interpretabilniji nego izrazi dobiveni običnim genetskim programiranjem.

Zahtjev za semantičkom ispravnosti stabla zahtijevat će dodatne promjene u samom algoritmu genetskog programa. Potrebno je osigurati da su sva stabla semantički ispravna, a to se najjednostavnije može postići na način da se prilikom inicijalizacije samih stabala odmah generiraju semantički ispravna stabla. No to samo po sebi nije dovoljno, jer operatori mutacije i križanja veoma lako mogu narušiti semantičku ispravnost stabla. Iz tog razloga potrebno je i te operatore prilagoditi kako bi se osigurala semantička ispravnost stabla. Ostali dijelovi algoritma genetskog programiranja (selekcija, evaluacija) ne zahtijevaju nikakve promjene.

U nastavku biti će opisani prilagođeni operatori inicijalizacije, križanja i mutacije za semantičko genetsko programiranje.

5.3.1. Izgradnja stabla

Inicijalizacija stabla je vjerojatno najvažniji dio semantičkog genetskog programiranja, budući da je njen zadatak da osim sintaksne ispravnosti, garantira i semantičku ispravnost rješenja. Ovo je posebno važno još i zbog razloga što se operatori mutacije i križanja oslanjaju na to da su sva stabla semantički ispravna, i ako ta pretpostavka ne vrijedi, ti operatori također neće raditi ispravno. Kako bi se pojednostavio opis algoritma, u nastavku će se prikazati njegova primjena na konkretan problem raspoređivanja na nesrodnim strojevima, no algoritam je općenit i može se poopćiti i na druge domene.

Prvo je potrebno prikazati koje su mjerne jedinice pridružene terminalima u problemu raspoređivanja na nesrodnim strojevima. Kao što se iz tablice 5.3 može vidjeti, uglavnom svi osnovni terminali imaju istu mjernu jedinicu. Jedina iznimka je težina w koja zapravo nema mjernu jedinicu. Mjerna jedinice težine može se zapravo shvatiti kao sekunda na nultu potenciju. Ovakvom interpretacijom težine ne narušava se semantička ispravnost stabala, a postupak njihove izgradnje se uvelike pojednostavljuje. Obavljanjem aritmetičkih operacija dobit će se uvijek jedna te ista mjerna jedinica, samo će imati drugačiju potenciju.

Tablica 5.4 prikazuje osnovni skup operatora koji se koristi u genetskom programiranju. Ranije je napomenuto kako efektivno postoji samo jedna mjerna jedinica (sekunda) u ovom problemu. Iz tog razloga dovoljno je promatrati samo potencije koje čvorovi imaju te nije potrebno brinuti o konkretnim mjernim jedinicama. Kao što se iz tablice može vidjeti, funkcije zbrajanja i oduzimanja zahtijevaju da operandi (odnosno lijevo i desno dijete tog čvora) imaju iste potencije, ali također vračaju rezultat koji ima istu potenciju kao i operandi. S druge strane, operatori množenja i dijeljenja nemaju ni-

Tablica 5.3: Mjerne jedinice za raspoređivanje na nesrodnim strojevima

Terminal	Mjerna jedinica
pt	sekunda
dd	sekunda
SL	sekunda
pmin	sekunda
pavg	sekunda
PAT	sekunda
MR	sekunda
age	sekunda
w	-

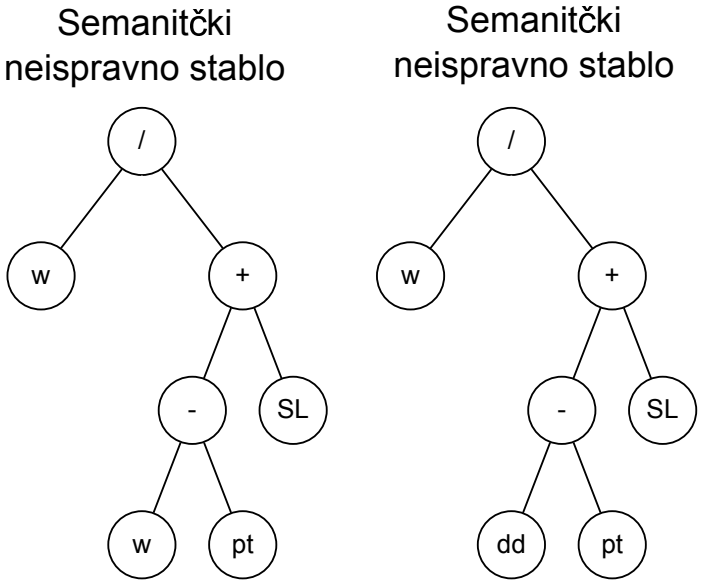
kakva ograničenja nad operandima, ali zato vračaju rezultat koji ima potenciju koja je zbroj potencija operanada (kod množenja), odnosno razlika potencija operanada (kod dijeljenja). Operator POS ima samo jedno dijete i ne postavlja nikakva ograničenja na njegovu potenciju, a vraća rezultat koji ima jednaku potenciju kao dijete.

Tablica 5.4: Funkcije koje se koriste u raspoređivanju na nesrodnim strojevima

Funkcija	Ograničenje	Računanje mjerne jedinice
+	lijevo i desno dijete moraju imati iste potencije	potencija lijevog ili desnog djeteta (neovisno kojeg jer su jednake)
-	lijevo i desno dijete moraju imati iste potencije	potencija lijevog ili desnog djeteta (neovisno kojeg jer su jednake
*	nema	zbroj potencija lijevog i desnog djeteta
/	nema	razlika potencija lijevog i desnog djeteta
pos	nema	potencija djeteta

Sada kada su prikazani i pojašnjeni svi čvorovi u okviru semantičkog genetskog programiranja, a prije nego se krene na opis algoritma za inicijalizaciju stabala, pokazat će se primjer jednog semantički ispravnog i neispravnog stabla. Slika 5.1 prikazuje upravo dva takva stabla. Prvo stablo predstavlja matematički izraz: $\frac{w}{w-pt+SL}$. Seman-

tička neispravnost se pojavljuje u dijelu stabla koji predstavlja izraz: $w - pt$. Kao što je ranije spomenuto, operator oduzimanja zahtijeva da operandi imaju iste potencije mjerne jedinice, dok to u ovom slučaju nije zadovoljeno jer w ima potenciju jednaku 0, a pt mjernu potenciju u iznosu od 1. Drugo stablo predstavlja izraz: $\frac{w}{dd-pt+SL}$. Ovaj izraz je u potpunosti semantički ispravan jer zadovoljava sva pravila koja su postavljena od stane operatora.



Slika 5.1: Primjer semantički neispravnog i ispravnog stabla

Algoritam inicijalizacije stabala mora uzeti u obzir sva ova ograničenja i biti u stanju izgraditi semantički ispravno stablo. Kako u literaturi nije pronađen niti jedan algoritam koji bi mogao izgenerirati stablo uz ovakva ograničenja, u sklopu ovog rada osmišljen je algoritam koji je bio korišten prilikom inicijalizacije jedinki. Algoritam je detaljnije opisan u nastavku poglavlja.

Algoritam je, kao i svi klasični algoritmi izgradnje stabla, rekurzivnog karaktera. U svakom rekurzivnom pozivu algoritam obavezno izgradi jedan čvor te ako nije dostigao maksimalnu dubinu, dalje se poziva rekurzivno. Time su određeni prvi parametri algoritma, a to su trenutna dubina i maksimalna dubina stabla. Radi jednostavnosti algoritma, umjesto klasične dubine koja se računa od korijena prema listovima, ovdje se dubina računa obrnuto, od listova do korijena. Ta se notacija koristi samo prilikom izgradnje stabala i nikako ne utječe na ostale dijelove genetskog programiranja. Nadalje, kako bi se izgenerirala semantički ispravna stabla, potrebni su dodatni parametri. Primjerice, ako je korijen stabla funkcija zbrajanja, i ako je izgenerirano lijevo podstablo s potencijom 3, onda je očito da desno podstablo koje je potrebno izgenerirati

također mora imati potenciju 3. Posljedica toga je da je algoritmu potrebna i informacija o tome koliku potenciju mora imati podstablo s korijenom u čvoru kojeg će generirati. Nadalje, da je umjesto funkcije zbrajanja korijen stabla sadržavao funkcija množenja, onda desno podstablo ne mora imati istu potenciju kao i lijevo i stoga se može puno slobodnije inicijalizirati. To ponašanje se može kontrolirati jednom zastavicom koja određuje je li na čvor koji je potrebno generirati postavljen uvjet. Ako uvjet nije postavljen, čvor se može generirati gotovo nasumično. U suprotnom, algoritam mora pratiti specifična pravila kako bi generirao čvor s točno zadanom potencijom. Dakle, potrebni parametri algoritma u svakom rekurzivnom pozivu su: trenutna dubina čvora, maksimalna dubina stabla, vrijednosti eksponenta koji je potrebno izgenerirati i zastavica koja određuje može li se ograničenje na eksponent zanemariti.

Najprije će se opisati najspecifičniji slučaj algoritma, a to je kada je potrebno generirati podstablo dubine 0. To zapravo znači da je u tom slučaju potrebno generirati čvor iz skupa terminalnih čvorova. Nadalje, postavlja se pitanje koji je terminal potrebno generirati, a to ovisi o uvjetu koji je postavljen na ovo podstablo. Ovdje se mogu pojaviti samo dva uvjeta, a to je da je potrebno generirati terminal s eksponentom 0, u tom slučaju će se generirati terminal w, ili da je potrebno generirati terminal s eksponentom 1, u tom slučaju će se generirati bilo koji drugi terminal. Također može postojati i slučaj da je potpuno nebitno koji se terminal generira, tada se odabere jedan terminal nasumično iz skupa svih terminala.

U općenitom slučaju, kada je potrebno izgenerirati podstablo veće dubine, algoritam razlikuje dva različita slučaja. U prvom slučaju ne postoji ograničenje koje podstablo mora zadovoljiti, dok u drugom slučaju mora se moći generirati podstablo koje predstavlja izraz sa specifičnom vrijednosti eksponenta. U nastavku će se opisati oba navedena slučaja.

Jednostavniji slučaj je kada nije postavljen uvjet na to koliki eksponent izgenerirano podstablo mora imati. Tada se može nasumično odabrati funkciju za trenutni čvor. Neovisno o tome koja je funkcija odabrana, lijevo podstablo se generira rekurzivnim pozivom algoritma, s dubinom smanjenom za jedan i bez ikakvih uvjeta. Nakon što je generirano lijevo podstablo, o funkciji trenutnog čvora ovisi kako će se generirati desno podstablo. Ako se radi o funkciji koja ne traži da operandi imaju isti eksponent, desno podstablo se generira na potpuno isti način kao i lijevo podstablo. Ako se radi o funkciji koja zahtijeva da operandi imaju isti eksponent, onda se algoritam poziva rekurzivno uz uvjet da je potrebno generirati podstablo sa zadanim eksponentom. Time su pokriveni svi slučajevi kada je potrebno generirati podstablo bez zahtjeva za određenim eksponentom.

Drugi, mnogo kompliciraniji slučaj je kada je potrebno izgenerirati podstablo koje mora imati točno određeni eksponent. U tom slučaju algoritam više ne može odabirati proizvoljne funkcije, već mora poštivati točno određena pravila kako bi se u svakom trenutku moglo generirati semantički ispravno stablo. Algoritam najprije određuje koji su maksimalni i minimalni eksponenti podstabala djece koja se mogu generirati iz ovog čvora. Ako je eksponent koji je potrebno generirati unutar tog intervala, onda se funkcija u trenutnom čvoru može odabrati nasumično. To je iz razloga što u tom slučaju postoji garancija da će se moći izgraditi lijevo i desno podstablo sa zadanim eksponentom. Ako se generira funkcija koja zahtijeva da su operandi jednaki, jednostavno se za svako podstablo pozove algoritam s dubinom umanjenom za jedan, ali s uvjetom da generiraju podstabla koja imaju eksponent koji je jednak eksponentu koji je postavljen na trenutni čvor. Ako se s druge strane radi o funkciji množenja, trenutni traženi eksponent se podijeli s dva i generiraju se podstabla s tom polovičnom vrijednosti eksponenta (ako je broj neparan, jedan eksponent je za jedan veći od drugog). Ako se radi o dijeljenju postupak je veoma sličan kao i kod množenja. Eksponent se podijeli s dva te se lijevo podstablo generira s tim polovičnim iznosom eksponenta, dok je desno podstablo potrebno generirati s negativnom polovičnom vrijednošću (dakle polovična vrijednost eksponenta se dodatno samo pomnoži s -1). To je posljedica toga što se kod dijeljenja novi eksponent rezultata računa kao razlika između eksponenata operanada, pa je zbog toga efektivni eksponent zapravo suprotnog predznaka. S druge strane, ako se traženi eksponent ne nalazi unutar intervala, tada ne postoji mogućnost slobodnog odabira funkcije. U tom slučaju, ako je eksponent veći od maksimalne vrijednosti, onda trenutni čvor obavezno mora sadržavati funkciju množenja kako bi na temelju svojih podstabala mogao izgenerirati traženi eksponent. Postupak je dalje isti kao i što je bio kod množenja kada je eksponent bio unutar intervala. Ako se dogodi da je eksponent manji od donje granice intervala, tada trenutni čvor mora sadržavati funkciju dijeljenja, kako bi se generirao dovoljno malen eksponent. Ostatak postupka je potpuno analogan postupku kada su eksponenti bili unutar intervala.

Prethodno opisani način inicijalizacije jedinke predstavlja zapravo *full* metodu inicijalizacije stabla prilagođenu za generiranje semantički ispravnih stabala. Algoritam se lako može prilagoditi da radi i s *grow* metodom. Jedina razlika koju je u algoritmu potrebno obaviti jest da se pri rekurzivnom pozivu algoritma za idući čvor kao dubina čvora preda neki slučajno generirani broj iz intervala od nula do dubine trenutnog čvora umanjene za jedan. Naravno potrebno je prije toga provjeriti da je s tom dubinom moguće zadovoljiti potrebna semantička pravila. Ako ih nije moguće zadovoljiti, potrebno je ponovo generirati broj dok se ne izgenerira dubina uz koju je moguće za-

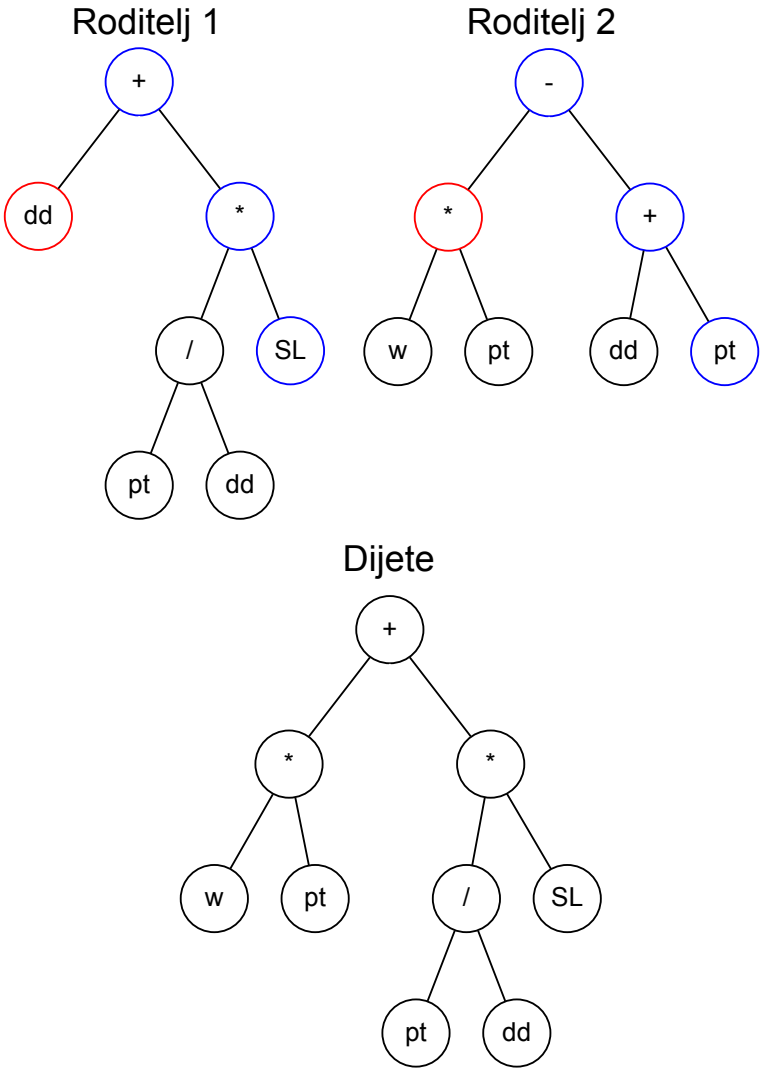
dovoljiti semantička pravila ili kao u *full* metodi jednostavno predati dubinu trenutnog čvora umanjenu za jedan, uz koju će se semantička pravila sigurno moći zadovoljiti.

5.3.2. Operator križanja

Svi operatori križanja koji su primjenjivi na obično genetsko programiranje primjenjivi su i na semantičko genetsko programiranje. Nažalost, jedinke koje nastanu primjenom tih operatora križanja više ne moraju biti semantički ispravne. Iz tog razloga potrebno je ta križanja prilagoditi kako bi generirala semantički ispravna rješenja. To uglavnom znači da se kao točke križanja mogu odabrati samo oni čvorovi koji imaju jednaku semantičku informaciju, odnosno koji u ovom slučaju imaju jednaki iznos eksponenata. Kako fokus ovog rada nije isključivo na semantičkom genetskom programiranju, već je ideja samo usporediti uspješnost semantičkog genetskog programiranja s klasičnim genetskim programiranjem, implementiran je samo jedan operator križanja i to križanje s jednom točkom.

Križanje s jednom točkom već je opisano u ranijem poglavlju i neće se ponovo opisivati čitav algoritam, već samo prilagodba koja ja napravljena kako bi se generirale semantički ispravne jedinke. Ranije je spomenuto kako ovo križanje za određivanje točaka križanja koristi zajedničku regiju. Kod križanja s jednom točkom prilagođenim za semantičko genetsko programiranje, čitav postupak će bit isti kao i kod običnog genetskog programiranja, jedino će se promijeniti način kako se određuje zajednička regija. Naime, da bi se par čvorova dodao u zajedničku regiju osim uvjeta da moraju imati jednak broj operanada, moraju zadovoljiti i dodatni uvjet koji nalaže da taj par čvorova ima istu semantičku informaciju, odnosno ima isti iznos eksponenata. Tim uvjetom se osigurava da se križaju samo oni čvorovi koji imaju isti iznos eksponenata i na taj način se eliminira mogućnost da se prilikom križanja stvore semantički neispravne jedinke. Ostatak postupka križanja je u potpunosti istovjetan kao i kod klasičnog genetskog programiranja.

Slika 5.2 prikazuje primjer križanja s jednom točkom prilagođenog za semantičko genetsko programiranje. Plavom bojom su označeni čvorovi koji pripadaju zajedničkoj regiji, dok su crvenom bojom označeni čvorovi koji predstavljaju točke križanja (ti čvorovi također spadaju u zajedničku regiju). Kao što se iz primjera može vidjeti izgled zajedničke regije malo je drugačiji nego bi to bio slučaj kod običnog genetskog programiranja. Iz primjera se može vidjeti kako oba roditelja predstavljaju semantički ispravne jedinke, i da je dijete koje je nastalo njihovom kombinacijom također semantički ispravno.



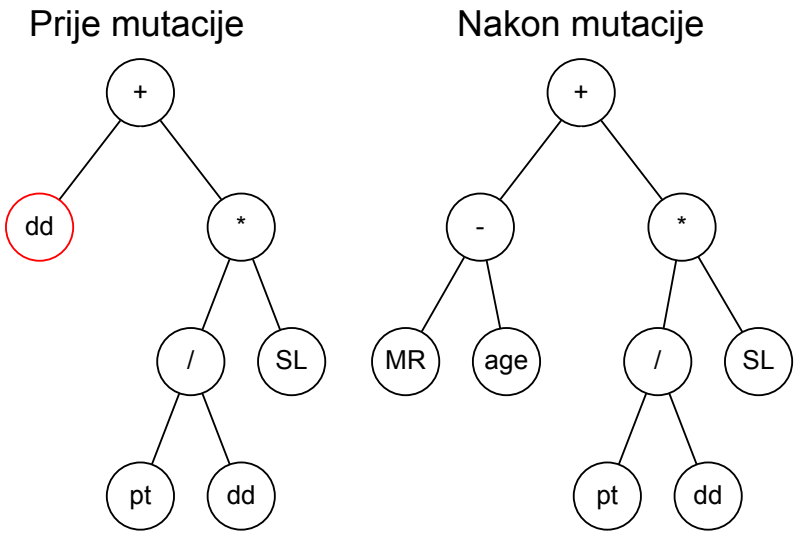
Slika 5.2: Primjer križanja s jednom točkom prilagođenog semantičkom genetskom programiranju

5.3.3. Mutacija

Za operatore mutacije vrijedi ista napomena kao i za operatore križanja. Svi operatori mutacije klasičnog genetskog programiranja su primjenjivi i na semantičko genetsko programiranje, no ne postoji garancija da će se primjenom tih operatora generirati semantički ispravne jedinke. Iz tog razloga operatore mutacija potrebno je dodatno prilagoditi kako se njihovom primjenom ne bi narušila semantička ispravnost stabla. Nažalost, kako operatori mutacije promjene u jedinku uvode na veoma različite načine, svaki od njih mora se prilagoditi na sebi svojstven način. U sklopu ovog rada odabrana je mutacija podstabla, te će se opisati kako je onda prilagođena da rješenja koja su njome dobivena budu semantički ispravna.

Kod mutacije jedinke, jedno konkretno podstablo jedinke se odbacuje te se zamjenjuje nekim novim, slučajno generiranim podstablom. Već iz ovog kratkog opisa trebalo bi biti intuitivno jasno koje je ograničenje potrebno zadovoljiti kako bi novonastala jedinka bila semantički ispravna. Naime, novo podstablo mora imati isti iznos eksponenta koji je imalo i staro podstablo. Ako je to zadovoljeno, nikada nije moguće generirati semantički neispravno stablo. Moguće je dodatno relaksirati ovo pravilo tako da, ako je roditelj čvora koji je korijen podstabla odabranog za brisanje, čvor koji ne zahtijeva da oba operanada imaju isti eksponent, tada je moguće generirati podstablo s proizvoljnom vrijednošću eksponenta. Time se zapravo dopušta puno slobodnije generiranje podstabala kada je to moguće. No, zbog jednostavnosti, ova modifikacija nije implementirana u operatoru mutacije.

Slika 5.3 prikazuje primjer mutacije podstabla prilagođene za semantičko genetsko programiranje. Crvenom bojom je označen čvor koji predstavlja korijen podstabla koje će se zamijeniti. Odabrani čvor ima vrijednost eksponenta od jedan, što znači da će generirano podstablo također morati imati vrijednost eksponenta od jedan. Može se vidjeti kako je to i bio slučaj, jer je navedeno podstablo zamijenjeno podstablom koje ima istu vrijednost eksponenta. Na taj način je očuvana semantička ispravnost mutiranog stabla.



Slika 5.3: Primjer mutacije podstabla prilagođenog semantičkom genetskom programiranju

5.4. Iterativno raspoređivanje

Ideja o uvođenju iterativnih pravila raspoređivanja (engl. *iterative dispatching rules*) u genetsko programiranje predložena je u [17]. U članku je navedena metoda pri-

mijenjena na problem raspoređivanja u okolini proizvoljne obrade (engl. *job shop*). Korištenjem ove metode dobiveni su bolji rezultati od onih koji su dobiveni korištenjem klasičnog algoritma genetskog programiranja. Iz navedenog razloga opravdano je ovu metodu primijeniti i u drugim okruženjima raspoređivanja. U sklopu ovog rada navedena metoda će se prilagoditi i primijeniti u okolini raspoređivanja na nesrodnim strojevima. U nastavku poglavlja biti će opisan postupak iterativnih pravila raspoređivanja te kako je on prilagođen za navedenu okolinu.

U tradicionalnim pravilima raspoređivanja, prioritetna funkcija se definira kao funkcija koja pridjeljuje prioritet pojedinom poslu u odnosu na promatrani stroj. Izlazna vrijednost te prioritetne funkcije se određuje na temelju određenih svojstava poslova i strojeva. Dakle, prioritetna funkcija može se definirati kao $\Delta(J, M)$, gdje je J konkretan posao, a M jedan konkretan stroj. Ograničenost ovakve definicije funkcije prioriteta leži u tome što se prioritet određuje samo na temelju dostupnih informacija o poslovima i strojevima u trenutku kada je odluka donesena. Kako ove informacije dolaze od djelomično generiranog rasporeda, utjecaj koju je ova odluka imala na konačno izgrađeni raspored se ne uzima u obzir. Za razliku od ovakve prioritetne funkcije može se definirati prioritetna funkcija oblika $\Delta^I(J, M, R)$, pri čemu R predstavlja određenu informaciju o rasporedu koja je zabilježena iz prethodne primjene pravila. Na taj način je nakon generiranja cjelovitog rasporeda zabilježena određena informacija o tom rasporedu koja se nastoji iskoristiti za što bolje generiranje rasporeda u budućnosti. Što to točno znači biti će detaljnije pojašnjeno u nastavku.

Algoritam 2 prikazuje pseudokod koji se koristi za određivanje dobrote jedinki kod iterativnih pravila raspoređivanja. Na početku algoritma se u R postavi inicijalna vrijednost zabilježene informacije, koja se odredi ovisno o tome koja se informacija rasporeda koristi. Dobrota trenutnog rješenja se postavi na maksimalnu moguću vrijednost. Nakon toga se korištenjem prioritetne funkcije izgradi raspored. Kada je raspored izgrađen odredi se dobrota takvog rasporeda. Ako je dobrota dobivenog rasporeda bolja od dobrote rasporeda koja je bila dobivena u prethodnoj iteraciji, onda se ta dobrota sprema u varijablu $Fitness^*$, u varijablu R se spremi nova informacija o rasporedu te se algoritam vrati na korak izračunavanja dobrote. Taj postupak se ponavlja sve dok dobrota izračunata u trenutnoj iteraciji nije lošija od dobrote u prethodnoj iteraciji. Potrebno je uočiti da će se uvijek izvesti barem jedna iteracija ovog postupka, jer je vrijednost početne dobrote postavljena na beskonačnost, pa će se u prvoj iteraciji uvijek dobiti neka bolja vrijednost za dobrotu. Za razliku od klasičnih pravila raspoređivanja koja donose odluku samo na temelju djelomično generiranih rasporeda, iterativno pravilo raspoređivanja koristi informacije iz kompletnog rasporeda kako bi

se ispravile pogreške napravljene u prethodnim iteracijama. Jedno bitno pitanje koje se postavlja oko ovog algoritma jest da li on uvijek konvergira. U [17] je pokazano kako ovaj postupak uvijek konvergira, te se neće ulaziti u detaljna pojašnjenja oko toga, no intuitivno se može doći do tog zaključka jer postoje minimalna i maksimalna vrijednost za svaki kriterij, broj stanja (odnosno broj iteracija) je konačan zbog preciznosti računala, a kako bi algoritam nastavio s radom u svakoj iteraciji mora prijeći u stanje s boljom vrijednošću dobrote.

Algoritam 2 Funkcija za računanje dobrote iterativnih pravila raspoređivanja

```
1:  function CalculateFitness
2:      R ← R^0
3:      Fitness* ← +∞
4:      Generiraj raspored korištenjem prioritetne funkcije Δ^I(J, M, R)
5:      Fitness ← izračunaj vrijednost dobrote na temelju generiranog rasporeda
6:      if Fitness* > Fitness then
7:          Fitness* ← Fitness
8:          R ← Iz rasporeda odredi novu vrijednost zabilježene informacije
9:          goto 4
10:     else
11:         return Fitness
12:     end if
13: end function
```

Nakon što je opisan algoritam iterativnih pravila raspoređivanja ostala je još jedna stvar koju je potrebno odrediti da bi se algoritam mogao primijeniti, a to su informacije koje će se koristiti iz generiranih rasporeda. Te informacije će se u jedinke uklopiti u obliku novih funkcijskih i terminalnih čvorova. Ovi čvorovi se ugrubo mogu podijeliti u dvije kategorije. U prvu kategoriju spadaju terminalni čvorovi koji sadrže informaciju koja je jednaka za sve poslove ($NLATE$ i $LATENESS$ čvorovi), dok u drugu kategoriju spadaju čvorovi koji sadrže informaciju čija vrijednost ovisi o trenutnom poslu čiji se prioritet računa ($ISLATE$ i $INDLATE$ čvorovi). U nastavku će se kratko opisati svi navedeni čvorovi. Bitno je za napomenuti kako odabrani čvorovi predstavljaju informacije koje su bitne za Twt kriterij jer je isti odabran kao glavni kriterij koji je minimiziran korištenjem iterativnih pravila raspoređivanja.

Tablica 5.5: Čvorovi za iterativna pravila raspoređivanja

Oznaka čvora	Opis
NLATE	broj zakašnjelih poslova
ISLATE	oznaka je li posao zakasnio
LATENESS	cjelokupna zakašnjelost
INDLATE	zakašnjelost konkretnog posla

$NLATE$ predstavlja broj poslova koji su u prošloj iteraciji algoritma (odnosno u prošlom izrađenom rasporedu) zakasnili. Ovim terminalom nastoji se broj poslova koji zakasne smanjiti iz iteracije u iteraciju. Vrijednost ovog terminala u prvoj iteraciji algoritma postavi se na ukupni broj poslova, odnosno kreće se s pretpostavkom da su svi raspoloživi poslovi zakasnili.

$ISLATE$ je jedini novi funkcijski čvor koji je uveden u iterativno pravilo raspoređivanja. Ovaj čvor sadrži dvoje djece i ovisno o tome je li posao koji se trenutno promatra zakasnio u prošloj iteraciji algoritma izvršava prvo podstablo navedenog čvora, dok se u suprotnom izvršava drugo podstablo čvora. Kao što se može vidjeti zapravo se radi o veoma pojednostavljenoj verziji uvjetnog čvora u kojem je uvjet eksplicitno ugrađen. Ovim čvorom nastoji se izgraditi dva nezavisna stabla od kojih bi jedno trebalo biti prilagođeno situaciji kada je posao u prošloj iteraciji zakasnio, a drugi situaciji kada taj posao nije zakasnio. U prvoj iteraciji se svi poslovi tretiraju kao da su zakasnili.

$LATENESS$ je terminalni čvor koji predstavlja ukupnu zakašnjelost rasporeda koji je generiran u prethodnoj iteraciji. U prvoj iteraciji se vrijednost ovog čvora postavlja na neku veliku vrijednost (obično beskonačno, no može i manje ako je poznata najveća moguća vrijednost zakašnjelosti).

$INDLATE$ je terminalni čvor koji predstavlja zakašnjelost pojedinog posla. Kao i kod $LATENESS$ čvora vrijednost ovog čvora se u prvoj iteraciji mora postaviti na neku veliku vrijednost, veću od maksimalne zakašnjelosti koju pojedini posao može imati.

5.5. GEP

Posljednja isprobana optimizacija zasnovana na uporabi ponešto drugačije reprezentacije rješenja (odnosno jedinki) pod nazivom GEP. GEP (engl. *gene expression programming*) jest vrsta evolucijskog algoritma koji je po svojim karakteristikama naj-

sličniji genetskim algoritmima i genetskom programiranju [4]. Sam algoritam se ne razlikuje previše od genetskog programiranja osim po načinu kako su rješenja predstavljena. Naravno, to rezultira i pojavom novih parametara algoritma, potrebom da se pojedini genetski operatori prilagode novom zapisu jedinki, kao i pojavom mogućnosti za uporabom novih genetskih operatora. U nastavku ovog poglavlja će se ukratko opisati najosnovniji dijelovi ovog algoritma.

Motivacija za isprobavanje ovakvog prikaza jedinki proizlazi iz činjenice da je on već bio primijenjen za izradu pravila raspoređivanja i da je pokazao dosta dobre rezultate [19].

5.5.1. Prikaz jedinki

GEP koristi prikaz jedinki koji se može smatrati svojevrsnom kombinacijom prikaza jedinki koje koriste genetski algoritam i genetsko programiranje. Sličnost s genetskim algoritmom je u tome što GEP koristi jedinke koje su uvijek jednake veličine (to kod genetskog programiranja nije slučaj). S druge strane, sličnost s genetskim programiranjem proizlazi iz činjenice da jedinke kod GEP algoritma također predstavljaju stabla različitih veličina i oblika, koja se interpretiraju kao izrazi. Na taj način GEP postiže prednost pred genetskim programiranjem na način da je određene genetske operatore križanja i mutacije puno lakše primijeniti.

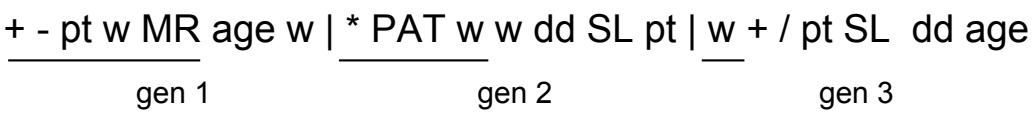
GEP jedinka se sastoji od nekoliko gena (engl. *gene*). Svaki gen se sastoji od niza primitiva koji predstavljaju jedan izraz. Kako bi se osiguralo da geni uvijek predstavlja sintaksno ispravne izraze, uvedeno je nekoliko ograničenja u gene. Svaki gen se može podijeliti na dva dijela: glava gena i rep gena. Glava (engl. *head*) gena je početak gena i ona se može sastojati od bilo kojih primitivnih čvorova. Rep (engl. *tail*) gena predstavlja završetak gena i on se može sastojati samo od terminalnih čvorova. Glava gena može biti proizvoljne veličine, no veličina repa gena ovisi o veličini glave gena. Veličina repa gena se računa po formuli $t = g*(n_{max}-1)+1$, pri čemu je t duljina repa, h duljina glave gena i n_{max} je maksimalni broj djece nekog čvora iz skupa primitiva koje algoritam koristi. Na ovaj način osigurano je da su sve jedinke, ako su generirane zadovoljavajući ova pravila, sintaksno ispravne.

Ranije je napomenuto kako se jedinka sastoji od više gena. Pitanje koje se intuitivno postavlja jest kako te gene, od kojih svaki predstavlja jedan zaseban izraz, spojiti u jedan jedinstveni izraz. Odgovor je jednostavan, geni se spajaju međusobno korištenjem takozvanih čvorova za spajanje (engl. *linking nodes*) koji moraju biti iz skupa funkcijskih čvorova. Na taj način je sve gene moguće spojiti u jedan izraz koji onda

zapravo predstavlja čitavu jedinku.

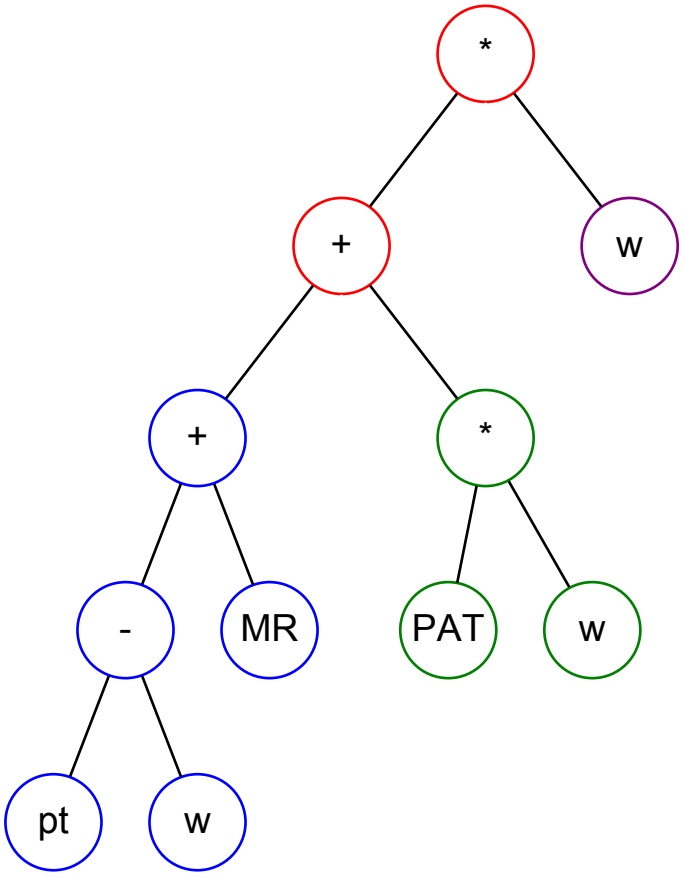
Posljednji koncept koji je potrebno objasniti jest način na koji se jedinka mapira u sam izraz. Postoji nekoliko načina na koje je to moguće napraviti. U ovom tipu algoritma se izrazi najčešće zapisuju korištenjem takozvanih *k-izraza* (engl. *k-expression*). U ovom radu to neće biti slučaj, već će izrazi u jedinci biti zapisani korištenjem prefiksne notacije. Razlog korištenja prefiksne notacije je taj što korišteni sustav evaluira izraze zapisane u prefiksnoj notaciji. Kada bi se koristili *k-izrazi* trebalo bi provesti još jedan međukorak koji bi taj izraz pretvorio u ekvivalentni prefiksni izraz. Zapis GEP jedinki korištenjem prefiksne notacije korišten je i u [16] te je pokazao dosta dobre rezultate.

Slika 5.4 prikazuje primjer jedne GEP jedinke. Prikazana jedinka se sastoji od tri gena (na slici su geni međusobno odvojeni oznakom "|"). Veličina glave gena jest tri. Kako je pretpostavljeno da svi funkcijski čvorovi imaju po dva djeteta, veličina repa gena jest četiri čvora. Iz tog proizlazi da je ukupna duljina jednog gena sedam čvorova. Iz primjera se može vidjeti kako se u repu gena niti u jednom slučaju ne pojavljuje funkcijski čvor, dok se u glavi gena mogu pojaviti i funkcijski i terminalni čvorovi. Na slici su podcrtani čvorovi koji u svakom genu tvore takozvanu kodnu regiju (engl. *coding region*), koja zapravo predstavlja dio gena koji izgrađuje sintaksni ispravan izraz. Ostatak gena prilikom izrade izraza se zanemaruje.



Slika 5.4: Primjer GEP jedinke

Slika 5.5 prikazuje jedinku prikazanu na slici 5.4 zapisanu u obliku stabla koje predstavlja određeni izraz. Kodna regija svakog gena se najprije pretvori stablo. Nakon toga se dobivena stabla povežu čvorovima za spajanje koji mogu biti unaprijed određeni ili se također mogu evaluirati tijekom algoritma. U ovom primjeru pretpostavljeno je da je prvi čvor za spajanje funkcijski čvor koji predstavlja množenje, dok je drugi čvor za spajanje funkcijski čvor koji predstavlja zbrajanje. Oba ta čvora su na slici označena crvenom bojom. Plavom bojom su označeni čvorovi iz kodne regije koja pripada prvom genu, zelenom čvorovi iz kodne regije koja pripada drugom čvoru i ljubičastom čvorovi koji pripadaju kodnoj regiji trećeg gena.



Slika 5.5: Primjer GEP jedinke zapisane u obliku stabla

5.5.2. Genetski operatori

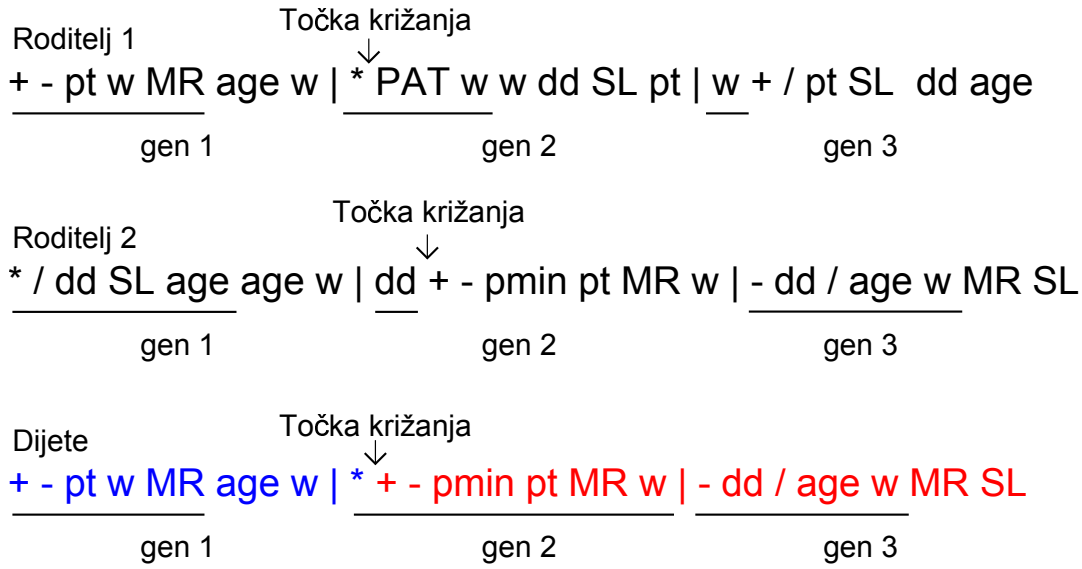
U ovom potpoglavlju bit će napravljen kratak osvrt na genetske operatore za GEP. Kratko će se opisati jedna implementirana mutacija i jedno implementirano križanje te kratko navesti ideja nove vrste operatora koja se ponekad koristi za GEP.

5.5.2.1. Operator križanja

Kao operator križanja implementirano je križanje s jednom točkom. Križanje s jednom točkom se provodi na način da se slučajno odabere jedna točka u jedinki. Dijete se stvori na način da se svi primitivni čvorovi ispred te točke preuzmu od jednog roditelja, a svi primitivni čvorovi nakon te točke od drugog roditelja. Kao što se može vidjeti, ovo križanje je po postupku izvođenja u potpunosti jednako istoimenom križanju koje se koristi kod genetskih algoritama. Upravo u ovom slučaju je vidljiva i prednost GEP jedinke jer za razliku od genetskog programiranja, nije potrebno paziti na strukturnu ispravnost generiranog rješenja, jer je ona implicitno zadovoljena i ne može se ovakvim

tipom križanja narušiti.

Slika 5.6 prikazuje primjer jednog ovakvog križanja. U djetetu plavom bojom su označeni čvorovi koji su preuzeti od prvog roditelja, dok su crvenom bojom označeni čvorovi koji su preuzeti od drugog roditelja. Iz ovog primjera se može vidjeti kako se križanjem mogu promijeniti veličine kodnih regija što će rezultirati time da će dijete imati drugačiju strukturu od oba roditelja.



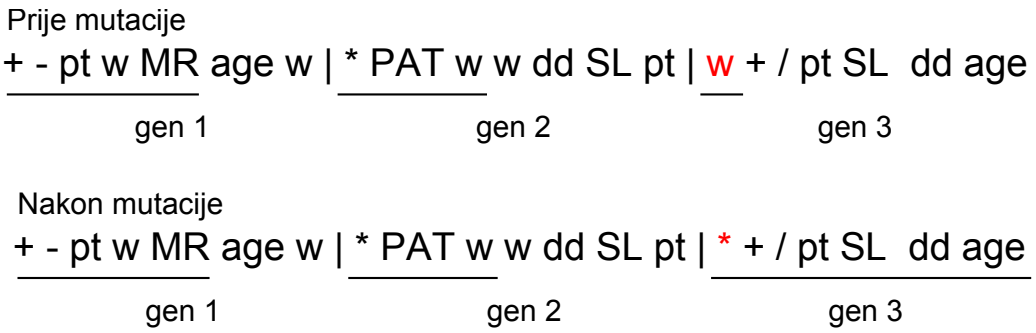
Slika 5.6: Primjer križanja s jednom točkom

5.5.2.2. Operator mutacije

Kao operator mutacije implementirana je nadomještajuća mutacija. Ideja ove mutacije je analogna istoimenoj mutaciji kod genetskog programiranja, slučajno se odabere jedan čvor u jedinki i zamijeni se s nekim slučajno generiranim čvorom. Nažalost, za razliku od prethodno opisanog križanja, mutacija neće moći toliko slobodno djelovati, već će djelovanje mutacije ovisiti o tome u kojem dijelu jedinke se nalazi čvor koji će se mutirati. Ako se čvor nalazi u glavi gena, onda se on može slobodno nadomjestiti bilo kojim drugim čvorom iz skupa svih primitiva. S druge strane, ako se čvor nalazi u repu gena, onda se može zamijeniti samo s terminalnim čvorom. Ovo je potrebno kako bi se sačuvala sintaksna ispravnost jedinke. Bitno je napomenuti da, za razliku od genetskog programiranja, ovdje odabrani čvor nije nužno zamijeniti čvorom koji ima jednak broj djece, već je odabir čvora proizvoljan dok god zadovoljava ranije navedene uvjete. Mogu se također mutirati i čvorovi za spajanje, koji onda moraju biti zamijenjeni nekim drugim funkcijskim čvorom.

Slika 5.7 prikazuje primjer nadomještajuće mutacije. Crvenom bojom je označen

čvor koji će biti nadomješten nekim drugim čvorom. Kako se može vidjeti, jedna ovako jednostavna mutacija koja kod genetsko programiranja ne uvodi strukturne promjene rješenja, ovdje može zapravo jako utjecati na samu strukturu izraza kojeg jedinka predstavlja.

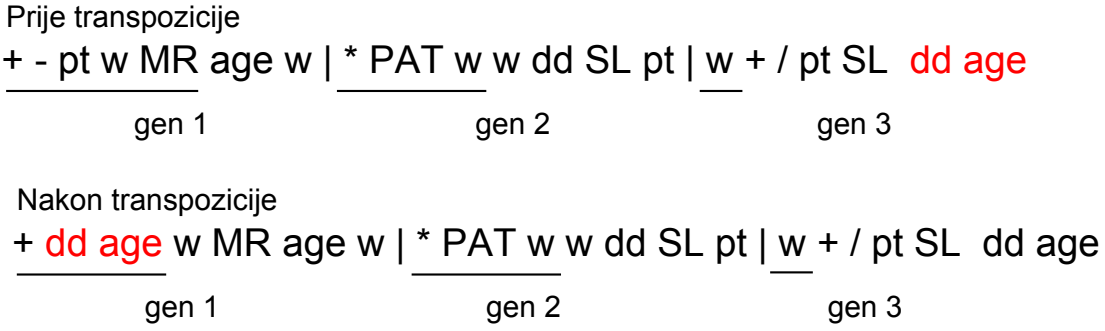


Slika 5.7: Primjer nadomještajuće mutacije

5.5.2.3. Operator transpozicije

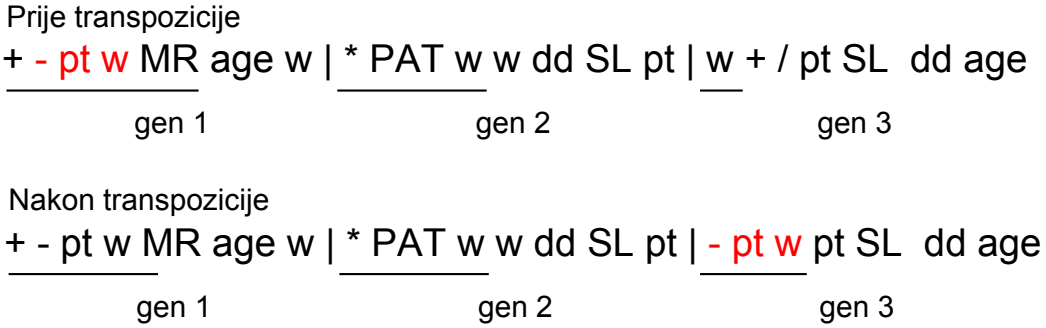
Kod GEP zapisa jedinke postoji jedan novi genetski operator koji se ponekad koristi i koji se naziva transpozicija (engl. *transposition*). Ovaj genetski operator je specifičan po tome što zapravo djeluje nad genima i dijelovima gena, a ne nad cijelom jedinkom. Postoji nekoliko različitih operatora transpozicije i u nastavku će se kratko opisati najčešće korišteni tipovi navedenog operatora.

IS transpozicija nasumično odabire jedan fragment gena koji započinje s terminalnim ili funkcijskim čvorom i transponira ga u glavu gena. Postoji jedno ograničenje koja ova transpozicija postavlja, a to je da fragment koji se transponira nije dozvoljeno smjestiti na sam početak gena. Prilikom ove transpozicije potrebno je paziti da odabrani fragment stane u glavu gena. Slika 5.8 prikazuje primjer IS transpozicije. Fragment čvorova koji se transponira označen je crvenom bojom.



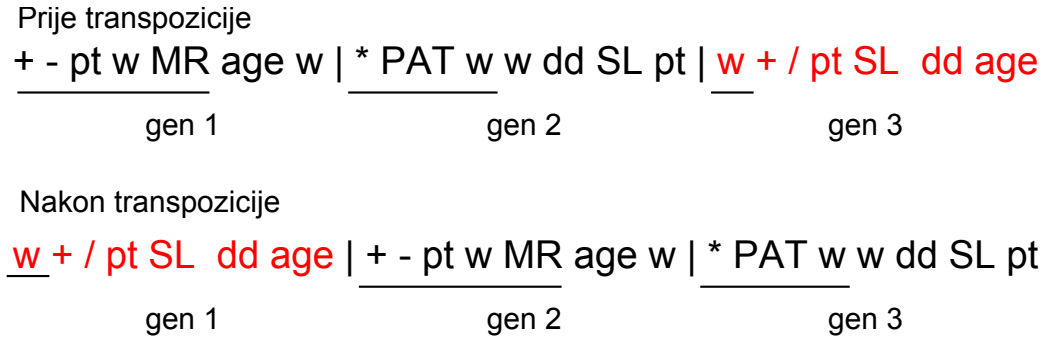
Slika 5.8: Primjer IS transpozicije

RIS transpozicija je veoma slična IS transpoziciji. Razlika je u tome što RIS transpozicija zahtjeva da odabrani fragment koji će se transponirati obavezno započinje s funkcijskim čvorom. Nadalje, prilikom transponiranja ovaj se fragment obavezno mora transponirati na sam početak gena. Slika 5.9 prikazuje primjer RIS transpozicije. Fragment čvorova koji se transponira označen je crvenom bojom.



Slika 5.9: Primjer RIS transpozicije

Transpozicija gena (engl. *gene transposition*), za razliku od prethodne dvije transpozicije koje su djelovale samo nad dijelovima gena, djeluje nad čitavim genima. Ova transpozicija nasumično odabire jedan gen koji se onda postavlja kao prvi gen u jedinci. Svi ostali geni koji su se prije transpozicije nalazili ispred toga gena se pomaknu za jedno mjesto u desno. Slika 5.10 prikazuje primjer transpozicije gena. Crvenom bojom je označen cijeli gen koji će transponirati korištenjem ovog operatora.



Slika 5.10: Primjer transpozicije gena

6. Rezultati

U ovom poglavlju prikazat će se rezultati koji su dobiveni korištenjem optimizacijskih postupaka navedenih u prošlom poglavlju. Svaki eksperiment bio je pokrenut u 50 izvođenja kako bi se dobili što značajniji rezultati. Za svako od tih pokretanja odredit će se konačno najbolje rješenje dobiveno u tom pokretanju. Za svaki eksperiment će se na temelju tih 50 pokretanja (i na temelju 50 dobivenih najboljih jedinki) izračunati pet kriterija: srednja vrijednost dobrote, minimalna vrijednost dobrote, maksimalna vrijednost dobrote, medijan dobrote i standardna devijacija dobrote. Prilikom usporedbe eksperimenata najbolja dobivena vrijednost za svaki od tih eksperimenata biti će podebljana.

Prije nego što se krene na opis rezultata, potrebno je još kratko opisati instance problema nad kojima će se provesti učenje i validacija. Za potrebe testiranja korišteno je 120 instanci problema koje su se sastojale od poslova koje je bilo potrebno rasporediti na određene strojeve. Ovisno o konkretnoj instanci problema, ukupni broj poslova može iznositi 12, 25, 50 ili 100 poslova, dok broj raspoloživih strojeva može iznositi 3, 6 ili 10 strojeva. Potrebno je još spomenuti kako je cjelokupni skup instanci ovih problema podijeljen na dva disjunktna skupa: skup za učenje i skup za validaciju. Skup za učenje bio je korišten za učenje algoritma genetskog programiranja, dok je skup za validaciju bio korišten za ispitivanje učinkovitosti dobivenih rješenja nakon završetka algoritma genetskog programiranja. Oba navedena skupa su sadržavala po 60 instanci problema raspoređivanja. Ovakvom podjelom svih primjera moguće je dobiti bolju ocjenu učinkovitosti razvijenog rješenja, uz neiskorištavanje određenog broja primjera tijekom postupka učenja. Svi rezultati koji će biti prikazani u nastavku ovog poglavlja su izmjereni nad skupom za validaciju.

6.1. Rezultati optimizacije parametara

U ovom poglavlju opisat će se različite optimizacije koje su provedene nad parametrima algoritma genetskog programiranja s ciljem ostvarenja boljih rezultata. U nas-

tavku poglavlja opisat će se provedene optimizacije nad veličinom populacije, maksimalnim brojem generacija, dubinom stabla, vjerojatnosti mutacije i skupom genetskih operatora križanja i mutacije.

6.1.1. Optimizacija veličine populacije i maksimalnog broja generacija

U ovom potpoglavlju biti će prikazani rezultati koji su dobiveni prilikom optimizacije maksimalnog broja generacija i veličine populacije nad kojom algoritam djeluje. Ova dva parametra su međusobno veoma povezana jer za veću populaciju će često biti potreban i veći broj generacija da se postignu bolji rezultati. Upravo iz tog razloga odlučeno je ta dva parametra optimirati istodobno. Tablica 6.1 prikazuje eksperimente obavljene u okviru ove optimizacije. Ponašanje algoritma isprobano je nad četiri veličine populacije. Kako bi se dobili što relevantniji rezultati svi eksperimenti pokrenuti su s maksimalnim brojem generacija od 300. Nakon što su eksperimenti obavljeni, najbolja vrijednost jedinke iz populacije je uzorkovana u različitim intervalima kako bi se odredio najisplativiji broj generacija. Za ove eksperimente kao kriterij optimizacije rasporeda odabrano je isključivo težinsko zaostajanje. Ostali kriteriji nisu bili uzeti u obzir.

Tablica 6.1: Eksperimenti za određivanje parametara veličine populacije i maksimalnog broja generacija

Broj eksperimenta	Veličina populacije	Maksimalni broj generacija
1	200 jedinki	300 generacija
2	500 jedinki	300 generacija
3	1000 jedinki	300 generacija
4	2000 jedinki	300 generacija

Tablica 6.2 prikazuje rezultate koji su dobiveni kroz navedene eksperimente. Stupci tablice prikazuju rezultate za svaki pojedini eksperiment, dok retci tablice određuju za koju su generaciju ti rezultati dobiveni. Jedan unos u tablici prikazuje srednju vrijednost dobrote (izračunatu na temelju dobrote najboljih jedinki, od kojih je dobivena po jedna za svako pokretanje eksperimenta) u trenutnoj generaciji. Slika 6.1 grafički prikazuje navedenu tablicu. Iz rezultata se može vidjeti kako se za populaciju koja se sastoji od samo 200 jedinki postižu značajno lošiji rezultati od ostalih veličina populacija. Za ostale veličine populacija postižu se rezultati koji su više manje sumjerljivi,

odnosno nema toliko velikih razlika između vrijednosti dobivenih rezultata. Razlika postaje očitija tek pri većem broju generacija, kada algoritam s populacijom od 2000 jedinki ipak postiže nešto bolje rezultate od ostalih algoritama. No prije nego što se odabere određena veličina populacije, potrebno je još dodatno pogledati kako se navedeni eksperimenti ponašaju za različiti broj generacija.

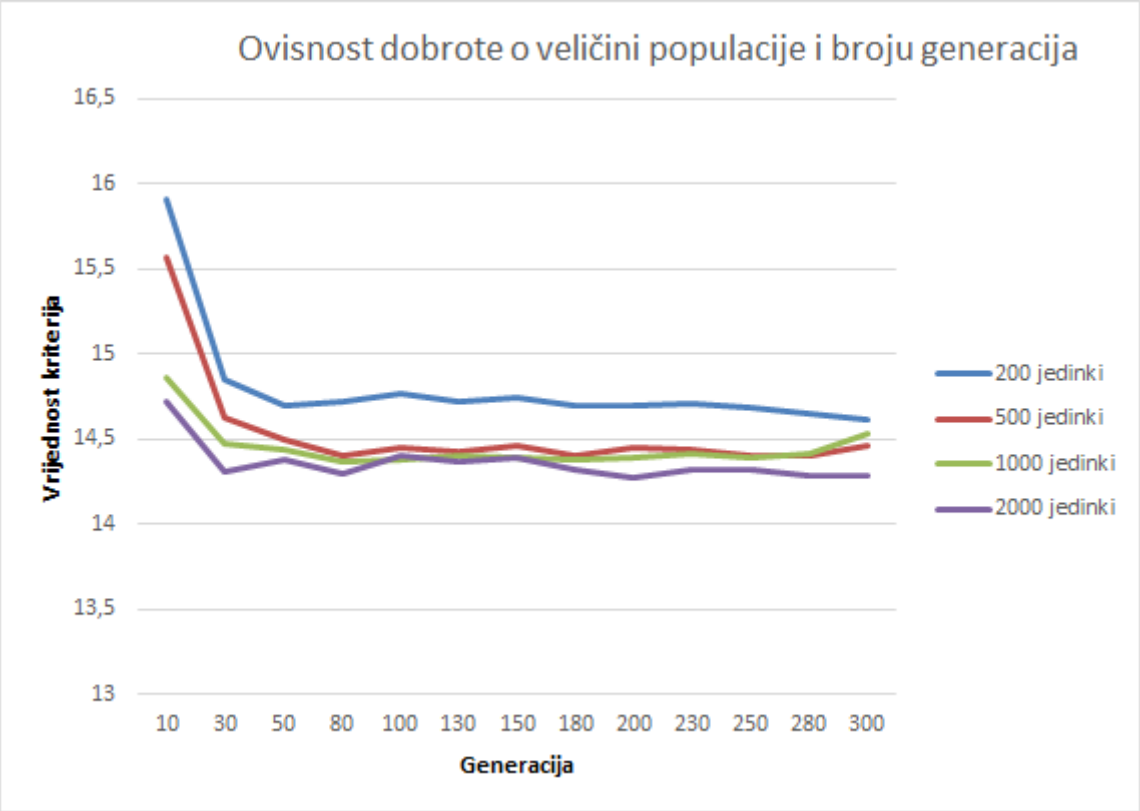
Ako se promotri promjena dobrote po generacijama za sve eksperimente može se uočiti da se dobrota ne smanjuje uvijek s povećanjem broja generacija. Za veći broj generacija može se uočiti kako u pojedinim dijelovima dolazi i do porasta dobrote. Iz tog razloga može se zaključiti kako za taj broj generacija došlo do prenaučenosti, odnosno da se algoritam počeo previše prilagođavati primjerima za učenje. Iz navedenog razloga nije dobro uzeti preveliki broj generacija, već je bolje ograničiti se na neki manji broj generacija. Ako se bolje promotre rezultati može se uočiti kako svi eksperimenti postižu određenu minimalnu vrijednost oko 80. generacije. Upravo iz tog razloga odlučeno je fiksirati broj generacija na 80. Još jedna prednost od korištenja manjeg broja generacija jest da je algoritam manje vremenski zahtjevan, što u pojedinim situacijama može biti veoma značajno.

Tablica 6.2: Rezultati za različite veličine populacija po generacijama

Generacija	Broj eksperimenta			
	1	2	3	4
10	15,907	15,574	14,858	14,723
30	14,855	14,631	14,481	14,312
50	14,702	14,498	14,442	14,380
80	14,727	14,406	14,374	14,299
100	14,766	14,452	14,375	14,399
130	14,726	14,424	14,408	14,373
150	14,741	14,458	14,397	14,387
180	14,695	14,406	14,378	14,316
200	14,694	14,453	14,391	14,280
230	14,715	14,435	14,413	14,317
250	14,691	14,401	14,388	14,317
280	14,646	14,399	14,419	14,281
300	14,622	14,459	14,530	14,291

Nakon što je fiksiran broj generacija koji će se koristiti može se pogledati koliki je bio iznos prosječne dobrote za svaki eksperiment u odabranoj generaciji. Tablica 6.3

prikazuje razne kriterije koji su izmjereni nad populacijama u 80. generaciji. Može se vidjeti kako eksperimenti 2, 3 i 4 postižu dosta slične rezultate, no algoritam za najveću veličinu populacije postiže najbolje rezultate za većinu navedenih kriterija. Nažalost, odabir veće populacije ima za posljedicu povećanje vremenske kompleksnosti algoritma. Iz tog razloga za veličinu populacije će se ipak odabrati samo 1000 jedinki jer poboljšanje koje se dobiva korištenjem populacije od 2000 jedinki nije dovoljno isplativo da opravda potrebu za dvostruko većom populacijom.



Slika 6.1: Graf koji prikazuje kretanje dobrote za različite veličine populacija u ovisnosti o broju generacija

Tablica 6.3: Usporedba rezultata za različite veličine populacije u 80. generaciji

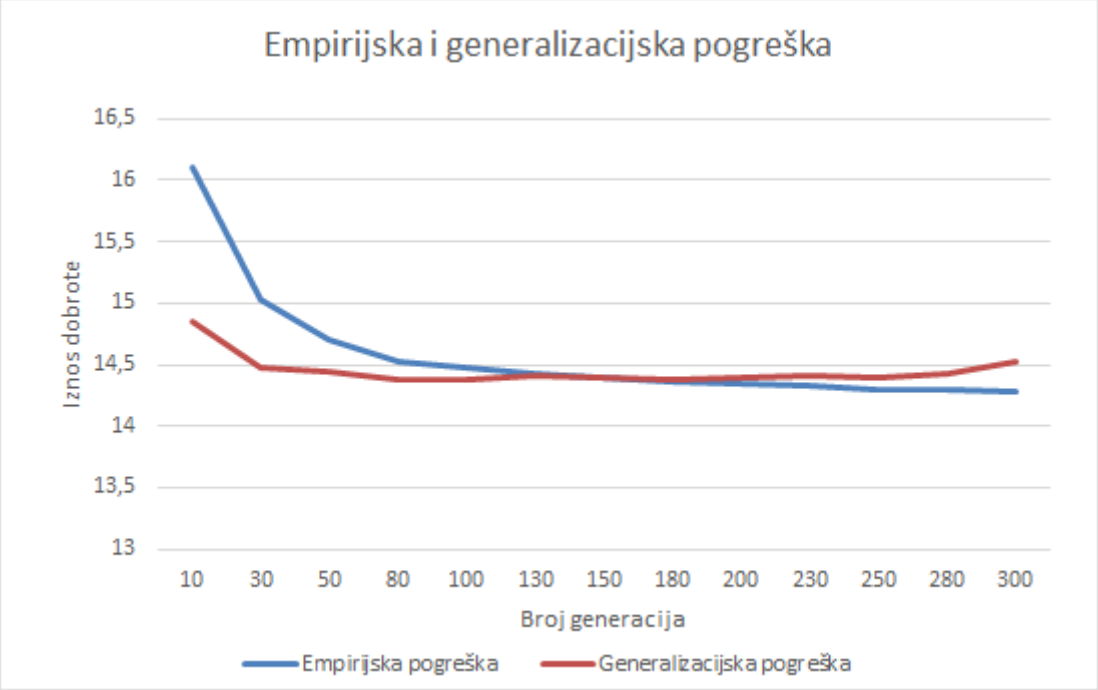
Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	14,727	12,968	19,595	14,474	1,3299
2	14,406	12,937	**16,751**	14,411	0,8460
3	14,374	13,213	17,004	14,300	**0,8078**
4	**14,299**	**12,889**	19,982	**13,890**	1,2807

Slika 6.2 prikazuje *box plot* prikaz rezultata dobivenih za 80. generaciju. Iz slike se može vidjeti kako se eksperiment za populaciju od 1000 jedinki, po vrijednosti dobrote najboljih jedinki, najmanje rasipa. To je također jedan od razloga zašto je odabrana upravo ova veličina populacije.

Slika 6.2: *Box plot* rezultata različitih veličina populacije u 80. generaciji

Slika 6.3 prikazuje empirijsku i generalizacijsku pogrešku koja se postiže za odabranu populaciju od 1000 jedinki ovisno o broju generacija. Empirijska pogreška je pogreška koju algoritam postiže na skupu za učenje, dok generalizacijska pogreška predstavlja onu pogrešku koja je postignuta na skupu za validaciju. Iz grafa se može

vidjeti da se pogreška na skupu za učenje konstantno smanjuje kako se povećava broj generacija. Ovakvo ponašanje algoritma jest i očekivano jer se algoritam sve više i više prilagođava primjerima za učenje. No, to nije slučaj i s pogreškom generalizacije koja se smanjuje do određenog broja generacija i nakon tog raste. Pojava da pogreška nakon određenog broja generacija ponovo raste naziva se prenaučenost. Iz ove slike također je vidljivo zašto se algoritam ne može objektivno ocijeniti korištenjem samo skupa za učenje. Korištenjem samo skupa za učenje može se dobiti ocjena pogreške koja je preoptimistična i zapravo ne predstavlja realnu procjenu pogreške.



Slika 6.3: Kretanje empirijske i generalizacijske pogreške ovisno o broju generacija

6.1.2. Optimizacija maksimalne dubine stabla

U ovom potpoglavlju opisat će se rezultati dobiveni za optimizaciju maksimalne dubine stabla. Tablica 6.4 prikazuje popis eksperimenata koji su obavljeni za ovu optimizaciju. Isprobano je pet različitih vrijednosti za maksimalnu dubinu stabala genetskog programiranja, od kojih je najmanje stablo bilo maksimalne dubine 5, dok je najveće bilo maksimalne dubine 13. U svim eksperimentima isključivo je optimiziran kriterij težinskog zaostajanja. Za optimalnu vrijednost maksimalne dubine stabala bit će odabrana ona vrijednost za koju se postiže najmanja srednja vrijednost dobivena na temelju najboljih jedinki dobivenih za svako pokretanje eksperimenta.

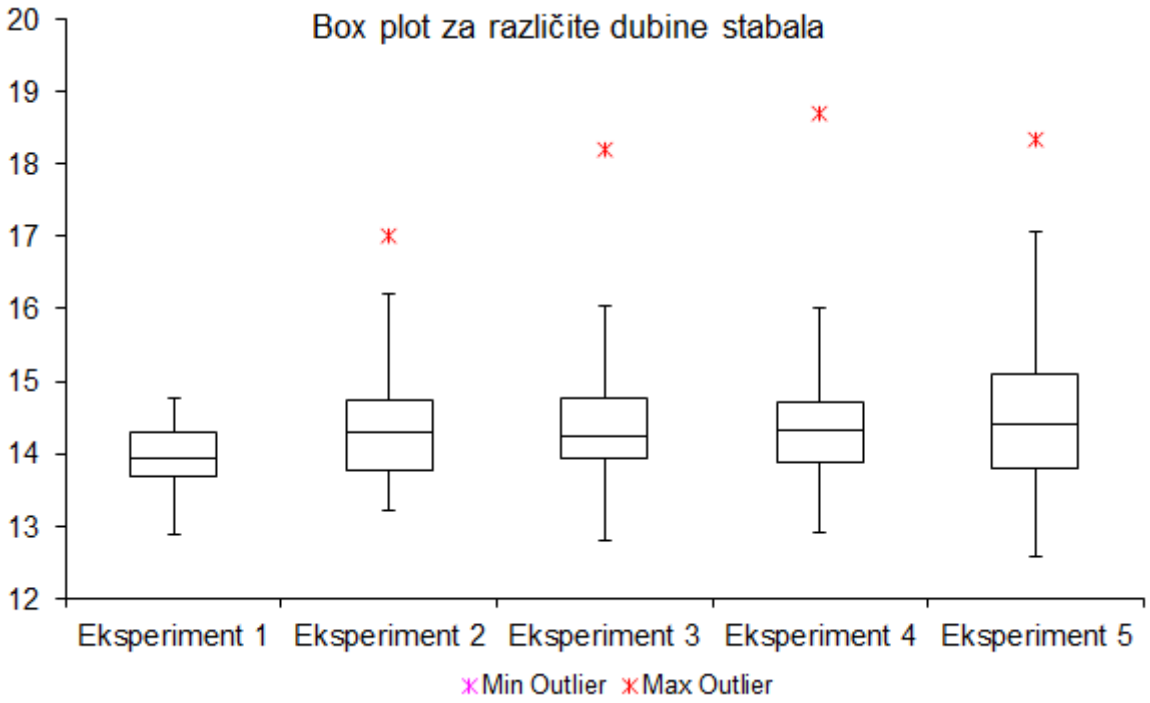
Tablica 6.4: Eksperimenti za određivanje parametra maksimalne dubine stabala

Broj eksperimenta	Maksimalna dubina stabla
1	5
2	7
3	9
4	11
5	13

Tablica 6.5 prikazuje različite kriterije koji su izračunati nad obavljenim eksperimentima. Dobiveni rezultati su pomalo neočekivani. Naime, očekivano je da će složenija stabla, odnosno stabla s većom dubinom, postići mnogo bolje rezultate od jednostavnijih stabala, no pokazano je upravo suprotno. Najjednostavnije stablo postiglo je najbolje rezultate u svim kriterijima osim u najmanjoj pronađenoj dobroti. Najbolje rješenje pronađeno je od strane algoritma koji je evoluirao najsloženija stabla. Iz navedenog se može zaključiti kako stabla s većom dubinom ipak uspijevaju pronaći jedinke s boljim iznosom dobrote, no ukupno gledano, srednja vrijednost dobrote pronađenih jedinki raste s porastom veličine stabala (što je posljedica da rješenja po iznosu dobrote puno više variraju za veća stabla). Slika 6.4 prikazuje *box plot* prikaz dobivenih rješenja. Iz slike se može vidjeti kako se rješenja dobivena algoritmom koji je koristio jednostavnije jedinke puno manje rasipaju po vrijednosti dobrote. To je vidljivo i iz tablice po iznosima standardne devijacije za svaki pojedini eksperiment. Iz navedenih razloga maksimalna dubina stabla postavljena je na vrijednost 5.

Tablica 6.5: Rezultati za različite iznose maksimalne dubine stabala

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	**13,955**	12,876	**14,757**	**13,941**	**0,4686**
2	14,374	13,213	17,004	14,300	0,8079
3	14,421	12,799	18,201	14,236	0,9568
4	14,452	12,922	18,685	14,337	0,9746
5	14,536	**12,583**	18,342	14,396	1,0454



Slika 6.4: *Box plot* za različite iznose maksimalne dubine stabala

6.1.3. Optimizacija vjerojatnosti mutacije

U ovom potpoglavlju prikazat će se rezultati dobiveni prilikom optimizacije parametra koji određuje vjerojatnost mutacije. Tablica 6.6 prikazuje popis eksperimenata koji su obavljeni. Iz tablice se može vidjeti kako je isprobano pet različitih iznosa vjerojatnosti mutacije. Kao kriterij optimizacije rasporeda korišteno je težinsko zaostajanje. Za optimalnu vrijednost vjerojatnosti mutacije biti će odabran onaj iznos za kojeg je postignut minimalan iznos srednje vrijednosti dobivene na temelju najboljih jedinki dobivenih za svako pokretanje eksperimenata.

Tablica 6.6: Eksperimenti za određivanje parametra vjerojatnosti mutacije

Broj eksperimenta	Vjerojatnost mutacije
1	0.07
2	0.1
3	0.2
4	0.3
5	0.5

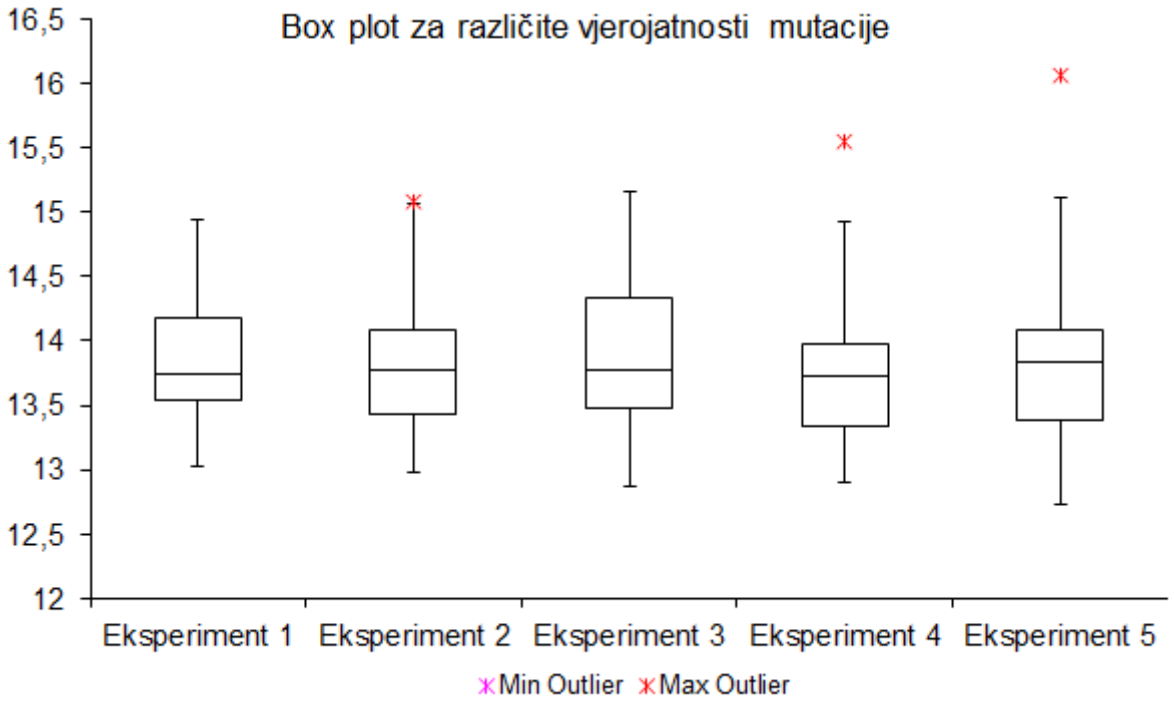
Tablica 6.7 prikazuje rezultata po raznim kriterijima dobivene za isprobane vrijed-

nosti vjerojatnosti mutacije. Kao što se iz tablice može vidjeti, niti jedan iznos vjerojatnosti mutacije ne dominira u potpunosti nad ostalim iznosima. Najbolje rješenje postignuto je za najveći iznos vjerojatnosti mutacije (0.5), najmanje rasipanje rješenja po iznosu dobrote postignuto je za najmanji iznos vjerojatnosti mutacije (0.07), dok su najbolje vrijednosti za kriterij medijana i srednje vrijednosti postignuti za vjerojatnost mutacije od 0.3. Postavlja se pitanje koju od navedenih vjerojatnosti mutacija odabrati. Naravno, odabir konkretne vrijednosti ovisi o tome kakvo je ponašanje algoritma poželjnije. U ovom slučaju odabran je iznos vjerojatnosti mutacije od 0.3 jer je glavni cilj optimizacije parametara bilo minimizirati srednju vrijednost dobrote postignutih rješenja.

Tablica 6.7: Rezultati za različite iznose vjerojatnosti mutacije

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,853	13,020	**14,949**	13,744	**0,4822**
2	13,786	12,983	15,084	13,768	0,5255
3	13,859	12,876	15,153	13,779	0,5614
4	**13,741**	12,901	15,548	**13,723**	0,5744
5	13,825	**12,736**	16,057	13,839	0,5939

Slika 6.5 prikazuje *box plot* prikaz rezultata navedenih u tablici 6.7. Iz ove slike se veoma dobro može vidjeti kako raspršenost rješenja po iznosu dobrote povećava s porastom iznosa vjerojatnosti mutacije. Druge vrijednosti, kao medijan i najmanja dobrota variraju u nešto manjoj mjeri.



Slika 6.5: *Box plot* za različite iznose vjerojatnosti mutacije

6.1.4. Optimizacija genetskih operatora križanja i mutacije

U ovom potpoglavlju prikazat će se rezultati koji su dobiveni na temelju optimizacije skupa genetskih operatora križanja i mutacije. Optimizacija je provedena na način kako je to opisano u poglavlju o optimizacijama. U okviru ovog potpoglavlja, radi preglednosti, prikazat će se samo odabrani rezultati, jer je za pojedine heuristike izveden poveći broj koraka. Prvo će se prikazati rezultati za proces pronalaska optimalnog skupa operatora križanja, nakon čega će se prikazati rezultati za proces pronalaska optimalnog skupa operatora mutacije. Kriterij ocjene rasporeda koji je optimiran u ovim eksperimentima je također težinsko zaostajanje.

6.1.4.1. Optimizacija skupa genetskih operatora križanja

U ovom potpoglavlju prikazat će se rezultati koji su dobiveni prilikom traženja optimalnog skupa operatora križanja. Svi ostali parametri algoritma će tijekom ovog postupka biti fiksirani.

Prvo će se prikazati rezultati dobiveni za izgrađujuću strategiju. Tablica 6.8 prikazuje eksperimente koji predstavljaju prvi korak izgrađujuće strategije. Treba napomenuti da prazni skup nije ispitivan, jer je postavljeno ograničenje da algoritam genetskog

programiranja mora raditi s barem jednim operatorom križanja.

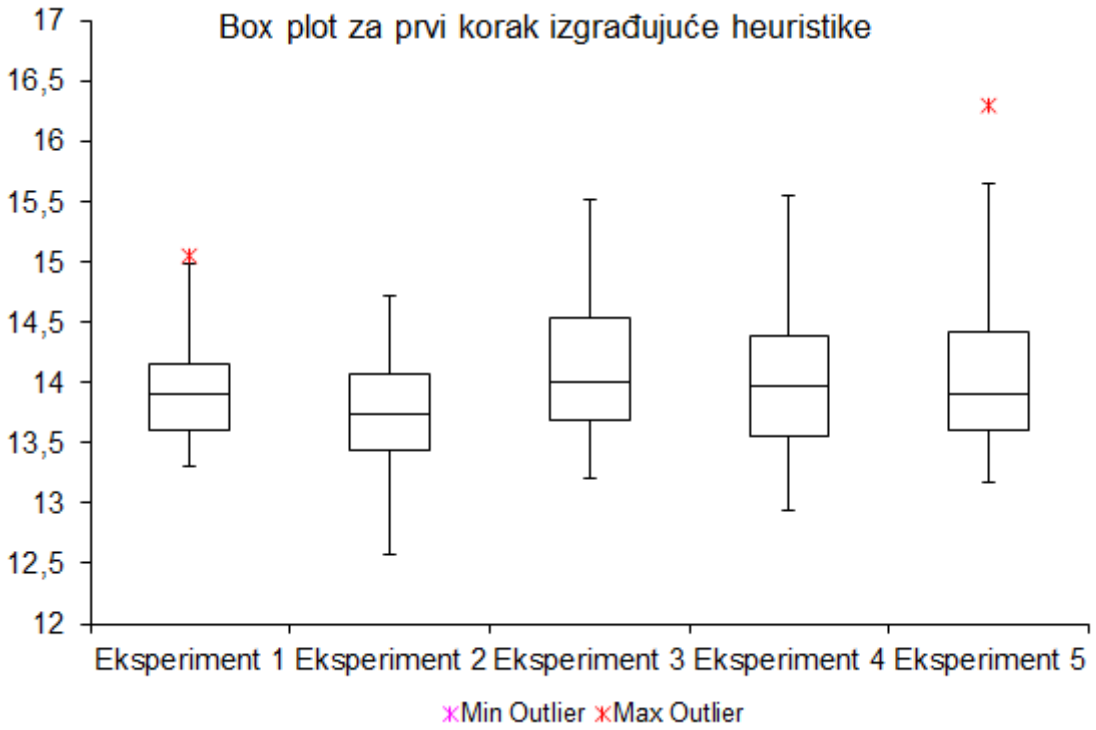
Tablica 6.8: Eksperimenti koji predstavljaju prvi korak izgrađujuće strategije za križanja

Broj eksperimenta	Skup korištenih operatora križanja
1	kontekstno očuvajuće križanje
2	veličinski pravedno križanje
3	križanje s jednom točkom
4	križanje podstabla
5	uniformno križanje

Tablica 6.9 prikazuje rezultate dobivene za prvi korak izgrađujuće heuristike. Iz tablice se jasno može vidjeti kako je gotovo po svim kriterijima (osim po iznosu standardne devijacije) eksperiment 2, koji je sadržavao samo veličinski pravedno križanje, bio bolji od ostalih eksperimenata. Iz tog razloga, u ovom koraku odabrano je veličinski pravedno križanje i ono se dodaje u skup korištenih operatora. Slika 6.6 prikazuje *box plot* prikaz za dobivene rezultate. Iz navedene slike se također može vidjeti kako je algoritam koji je koristio samo veličinski pravedno križanje postigao značajno bolje rezultate od ostalih (po iznosu medijana i najmanje pronađene dobrote rješenja).

Tablica 6.9: Rezultati za prvi korak izgrađujuće strategije za križanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,910	13,297	15,041	13,905	**0,4129**
2	**13,745**	**12,577**	**14,712**	**13,735**	0,4672
3	14,146	13,201	15,510	14,005	0,6069
4	14,024	12,938	15,542	13,966	0,5998
5	14,100	13,165	16,302	13,894	0,6961



Slika 6.6: *Box plot* prikaz za prvi korak izgrađujuće strategije za operatore križanja

Tablica 6.10 prikazuje eksperimente koji su obavljeni u okviru drugog koraka izgrađujuće heuristike. Svaki preostali operator križanja je isproban u kombinaciji s veličinski pravednim križanjem (koje se pokazalo najboljim u prošloj iteraciji) kako bi se odredilo da li njihova kombinacija postiže još bolje rezultate.

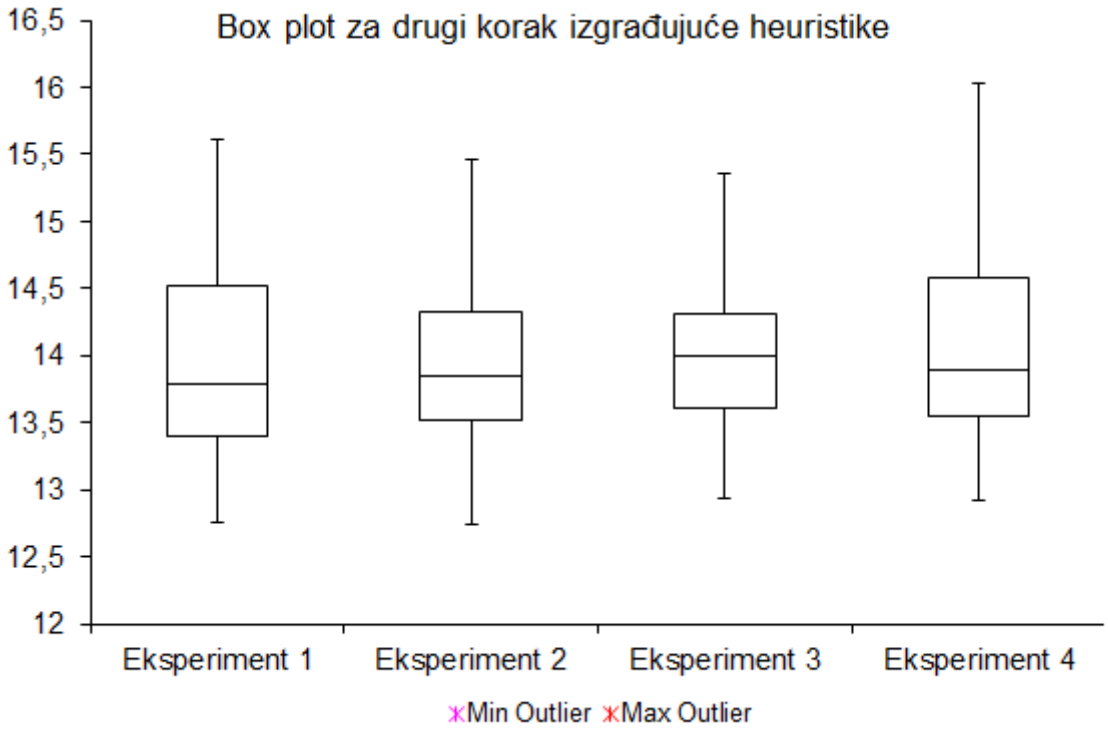
Tablica 6.10: Eksperimenti koji predstavljaju drugi korak izgrađujuće strategije za operatore križanja

Broj eksperimenta	Skup korištenih operatora križanja
1	veličinski pravedno križanje, konteksno očuvajuće križanje
2	veličinski pravedno križanje, križanje s jednom točkom
3	veličinski pravedno križanje, križanje podstabla
4	veličinski pravedno križanje, uniformno križanje

Tablica 6.11 prikazuje rezultate koji su dobiveni za drugi korak izgrađujuće heuristike. Može se vidjeti kako razlika između eksperimenata u ovom koraku nije toliko velika kao što je bila u prošlom. No eksperiment 2 postiže najbolji rezultat po kriteriju srednje vrijednosti. Slika 6.7 prikazuje *box plot* prikaz navedenih rješenja. Iz ove slike može se veoma dobro vidjeti kako razlike između rješenja nisu toliko velike.

Tablica 6.11: Rezultati za drugi korak izgrađujuće strategije za operatore križanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,939	12,759	15,618	**13,783**	0,6903
2	**13,912**	**12,746**	15,462	13,849	0,5980
3	13,917	12,940	**15,362**	13,990	**0,5346**
4	14,060	12,921	16,036	13,888	0,6558



Slika 6.7: *Box plot* prikaz za drugi korak izgrađujuće strategije za operatore križanja

U ovom koraku prestaje rad izgrađujuće heuristike. Razlog tome je što je najbolje pronađeno rješenja u ovom koraku lošije od najboljeg pronađenog rješenja iz prošlog koraka. Konačni optimalan skup operatora križanja kojeg je pronašla izgrađujuća heuristike jest skup koji se sastoji od veličinski pravednog križanja.

Nakon što je jedno rješenje pronađeno izgrađujućom heuristikom, iskoristit će se i razgrađujuća heuristika kako bi se pronašlo još jedno rješenje i time povećala vjerojatnost da će se pronaći bolje rješenje. Tablica 6.12 prikazuje eksperimente koji su

obavljeni u sklopu prvog koraka razgrađujuće heuristike. Zbog preglednosti u rezultate je uvršten i početni skup koji se sastoji od svih operatora križanja (označen kao eksperiment 1).

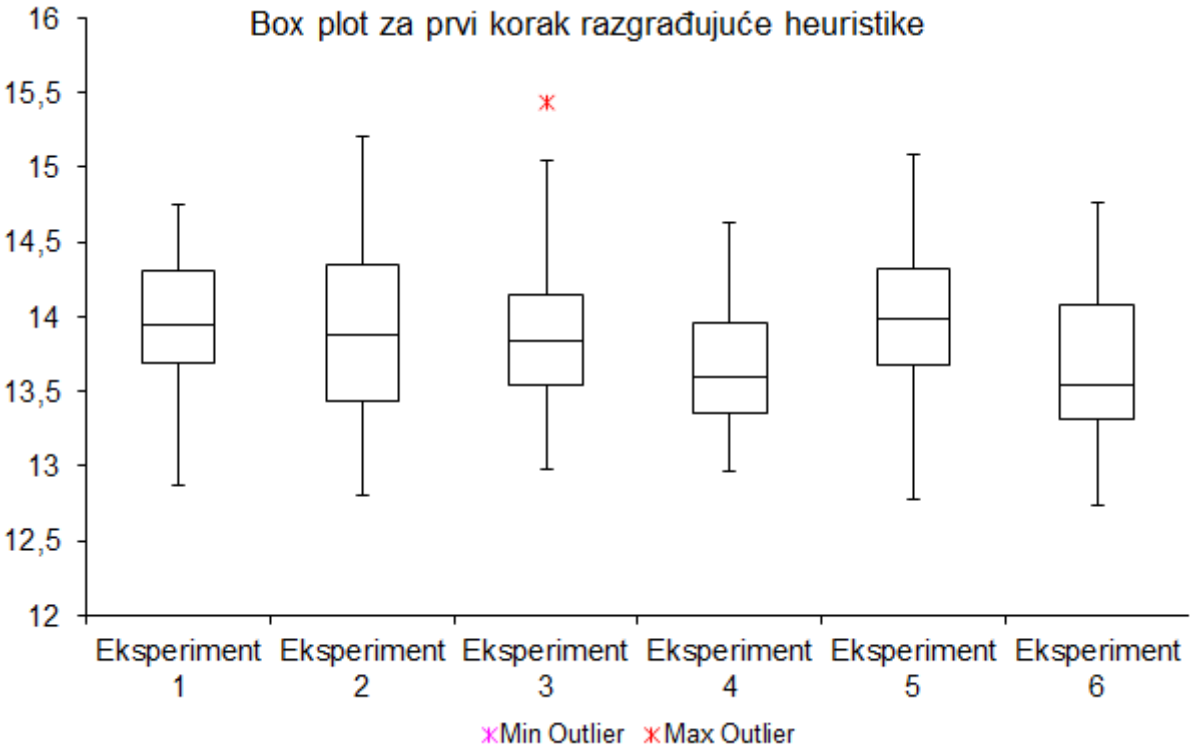
Tablica 6.12: Eksperimenti koji predstavljaju prvi korak razgrađujuće strategije ze operatore križanja

Broj eksperimenta	Skup korištenih operatora križanja
1	kontekstno očuvajuće križanje, veličinski pravedno križanje, križanje s jednom točkom, križanje podstabla,uniformno križanje
2	veličinski pravedno križanje, križanje s jednom točkom, križanje podstabla,uniformno križanje
3	kontekstno očuvajuće križanje, križanje s jednom točkom, križanje podstabla,uniformno križanje
4	kontekstno očuvajuće križanje, veličinski pravedno križanje, križanje podstabla,uniformno križanje
5	kontekstno očuvajuće križanje, veličinski pravedno križanje, križanje s jednom točkom, uniformno križanje
6	kontekstno očuvajuće križanje, veličinski pravedno križanje, križanje s jednom točkom, križanje podstabla

Tablica 6.13 prikazuje rezultate koji su dobiveni u prvom koraku razgrađujuće strategije (uz iznimku prvog eksperimenta koji prikazuje rezultate za potpuni skup). Kao što se može vidjeti, u prvom koraku su postignuti rezultati koji su bili bolji od potpunog skupa operatora. Dva eksperimenta su se posebice istakla i to eksperimenti 4 i 6. Eksperiment 4 pokazao se boljim po tome što je ukupna srednja vrijednost bila najmanja od svih ostalih eksperimenata i po tome što su se rješenja po iznosu dobrote za ovaj eksperiment najmanje raspršivala. S druge strane eksperiment 6 je bio bolji po iznosu medijana i po tome što je u tom eksperimentu pronađeno rješenje s najmanjim iznosom dobrote. Kao optimalan skup operatora križanja odabran je onaj koji je korišten u četvrtom eksperimentu jer je za taj eksperiment postignut najmanji iznos srednje vrijednosti dobrota rješenja. Slika 6.8 prikazuje *box plot* prikaz za ranije navedene rezultate. Iz slike se može vidjeti kako korištenje pojedinih operatora utječe na medijane rješenja kao i njihove raspršenosti.

Tablica 6.13: Rezultati za prvi korak razgrađujuće strategije kod operatora križanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,955	12,876	14,757	13,941	0,4686
2	13,902	12,806	15,204	13,880	0,5571
3	13,897	12,975	15,430	13,834	0,5087
4	**13,657**	12,962	**14,624**	13,601	**0,4257**
5	13,978	12,775	15,094	13,985	0,4942
6	13,666	**12,742**	14,761	**13,536**	0,4931



Slika 6.8: *Box plot* prikaz za prvi korak razgrađujuće strategije za operatore križanja

Tablica 6.14 prikazuje eksperimente koji su obavljeni u drugom koraku razgrađujuće heuristike. U ovom koraku su izbacivani operatori iz skupa koji je određen kao optimalan u prošlom koraku heuristike.

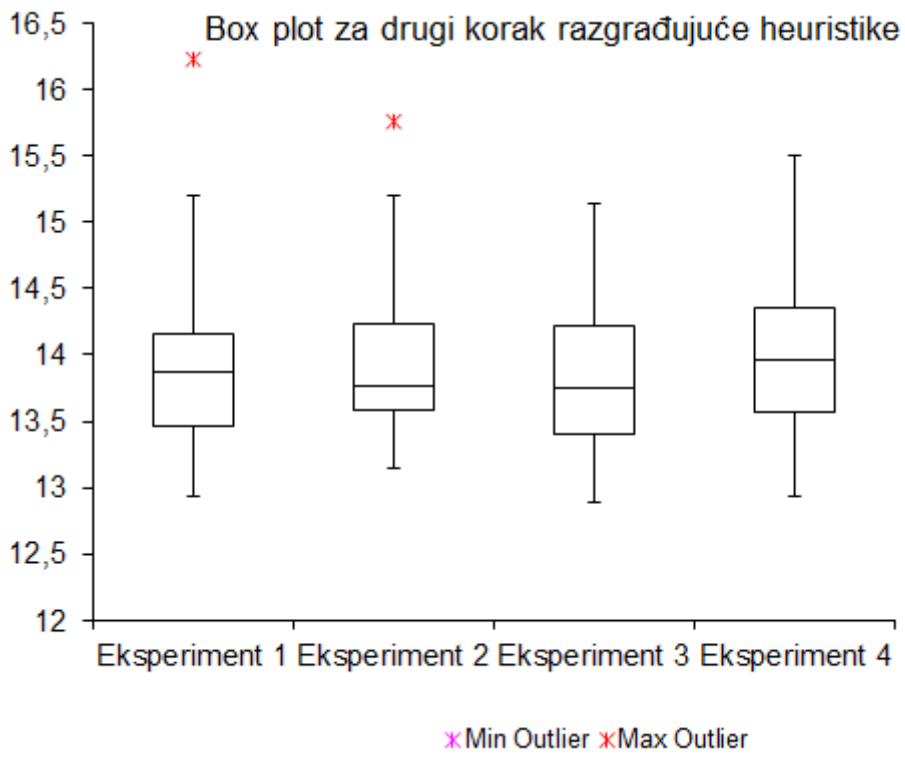
Tablica 6.14: Eksperimenti koji predstavljaju drugi korak razgrađujuće strategije za operatore križanja

Broj eksperimenta	Skup korištenih operatora križanja
1	veličinski pravedno križanje, križanje podstabla,uniformno križanje
2	kontekstno očuvajuće križanje, križanje podstabla,uniformno križanje
3	kontekstno očuvajuće križanje, veličinski pravedno križanje, uniformno križanje
4	kontekstno očuvajuće križanje, veličinski pravedno križanje, križanje podstabla

Tablica 6.15 prikazuje rezultate dobivene za drugi korak razgrađujuće strategije. Iz tablice je vidljivo kako su najbolji rezultati postignuti za eksperiment 3, koji je po gotovo svim kriterijima postigao najbolje rezultate. Iz tog razloga upravo je skup operatora križanja koji je bio korišten u ovom koraku odabran kao optimalan skup operatora. Slika 6.9 prikazuje *box plot* prikaz za ranije navedene rezultate, iz kojeg se može vidjeti kako zapravo ne postoji neka značajno velika razlika između obavljenih eksperimenata.

Tablica 6.15: Rezultati za drugi korak razgrađujuće strategije za operatore križanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,895	12,936	16,225	13,876	0,5979
2	13,984	13,148	15,761	13,771	0,5886
3	**13,831**	**12,894**	**15,140**	**13,748**	0,5502
4	13,970	12,933	15,506	13,961	**0,5484**



Slika 6.9: *Box plot* za drugi korak razgrađujuće strategije za operatore križanja

Nakon ovog koraka postupak prestaje jer je za optimalan skup koji je pronađen u ovom koraku postignuto lošije rješenje od rješenja koje je pronađeno u prošlom koraku ovog postupka. Dakle, najbolji skup operatora križanja koji je pronađen korištenjem razgrađujuće strategije jest onaj koji je bio korišten od eksperimenta 4 u prvom koraku strategije, odnosno skup koji se sastoji od: kontekstno očuvajućeg križanja, veličinski pravednog križanja, križanja podstabla i uniformnog križanja.

Nakon što su obavljene obje heuristike, izgrađujuća i razgrađujuća, dobiveno je po jedan skup operatora križanja za svaku od njih. U konačnici je potrebno od ta dva dobivena skupa odabrati onaj za kojeg su postignuti bolji rezultati. U ovom slučaju, bolje rezultate je postigao skup koji je dobiven korištenjem razgrađujuće heuristike, te se iz tog razloga on odabire kao optimalan skup genetskih operatora križanja koji će se koristiti.

6.1.4.2. Optimizacija skupa genetskih operatora mutacije

Nakon što je određen optimalan skup operatora križanja potrebno je još odrediti i optimalan skup operatora mutacije. Postupak će biti isti kao što je bio i za određivanja optimalnog skupa operatora križanja, odnosno ponovno će se koristiti izgrađujuća i

razgrađujuća heuristika.

Tablica 6.16 prikazuje eksperimente obavljene u okviru prvog koraka izgrađujuće heuristike za operatore mutacije. Kao i kod križanja, prazan skup, koji ne sadrži niti jedan genetski operator mutacije nije uzet u obzir.

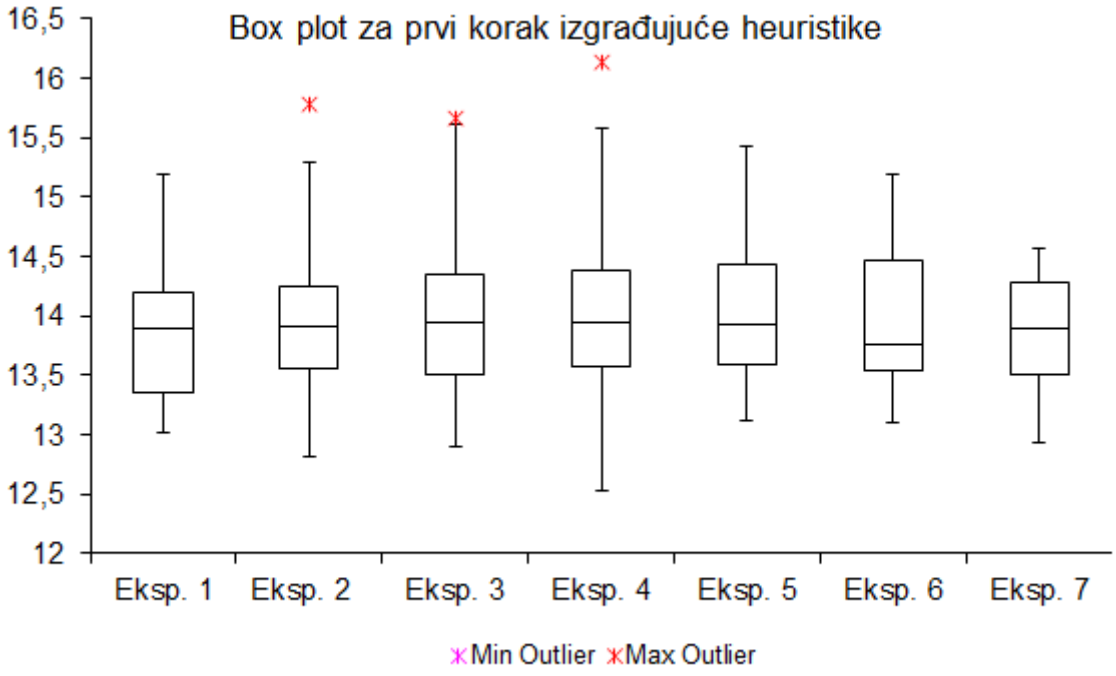
Tablica 6.16: Eksperimenti koji predstavljaju prvi korak izgrađujuće strategije za operatore mutacije

Broj eksperimenta	Skup korištenih operatora mutacije
1	komplementirajuća mutacija
2	Gaussova mutacije
3	podizajuća mutacija
4	permutirajuća mutacija
5	nadomještajuća mutacija
6	smanjujuća mutacija
7	mutacija podstabla

Tablica 6.17 prikazuje rezultate koji su dobiveni za prvi korak izgrađujuće heuristike. Može se vidjeti kako niti jedan operator mutacije nije zapravo postigao dominantne rezultate u svim kriterijima zbog čega je posljedično veoma teško odrediti koji je operator mutacije zapravo bio najučinkovitiji. Iako se eksperimenti po dobivenim srednjim vrijednostima međusobno ne razlikuju odviše, ta razlika dolazi puno više do izražaja u drugim kriterijima kao što su primjerice najbolje pronađeno rješenje i iznos standardne devijacija. No, kao i za sve postupke do sada koristit će se kriterij najmanje srednje vrijednosti. Iz tog razloga će se kao optimalan skup genetskih operatora mutacije odabrati onaj koji je bio korišten od strane eksperimenta 7 i koji se sastoji od smanjujuće mutacije. Taj skup će se probati proširiti u idućem koraku kako bi se dobio skup za koji se postižu još bolji rezultati. Slika 6.10 prikazuje *box plot* prikaz navedenih rezultata. Iz ove slike može se vidjeti kako razlike u medijanima pojedinih eksperimenata nisu toliko velike, no da raspršenost rješenja po iznosu dobrote dosta ovisi o tome koji je genetski operator mutacije korišten. Srećom, niti jedan operator nije statistički značajno bolji ili lošiji.

Tablica 6.17: Rezultati za prvi korak izgrađujuće strategije za operatore mutacije

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	**13,837**	13,020	15,192	13,887	0,5313
2	13,947	12,813	15,773	13,908	0,5853
3	13,965	12,900	15,667	13,935	0,5702
4	13,967	**12,534**	16,135	13,947	0,6134
5	14,012	13,124	15,419	13,9184	0,5557
6	13,951	13,099	15,190	**13,753**	0,5867
7	13,871	12,925	**14,559**	13,896	**0,4486**



Slika 6.10: *Box plot* za prvi korak izgrađujuće strategije za operatore mutacije

Rezultati za drugi korak izgrađujuće heuristike se zbog preglednosti neće prikazati. Bitno je samo napomenuti kako je u drugom koraku najbolje rezultate postigao skup operatora mutacije koji se sastojao od komplementirajuće i smanjujuće mutacije. Ovaj skup je postigao bolje rezultate od najboljeg skupa pronađenog u prvom koraku i iz tog

razloga postupak prelazi u treći korak koji će biti opisan u nastavku.

Tablica 6.18 prikazuje eksperimente koji su pokrenuti u sklopu trećeg koraka izgrađujuće heuristike.

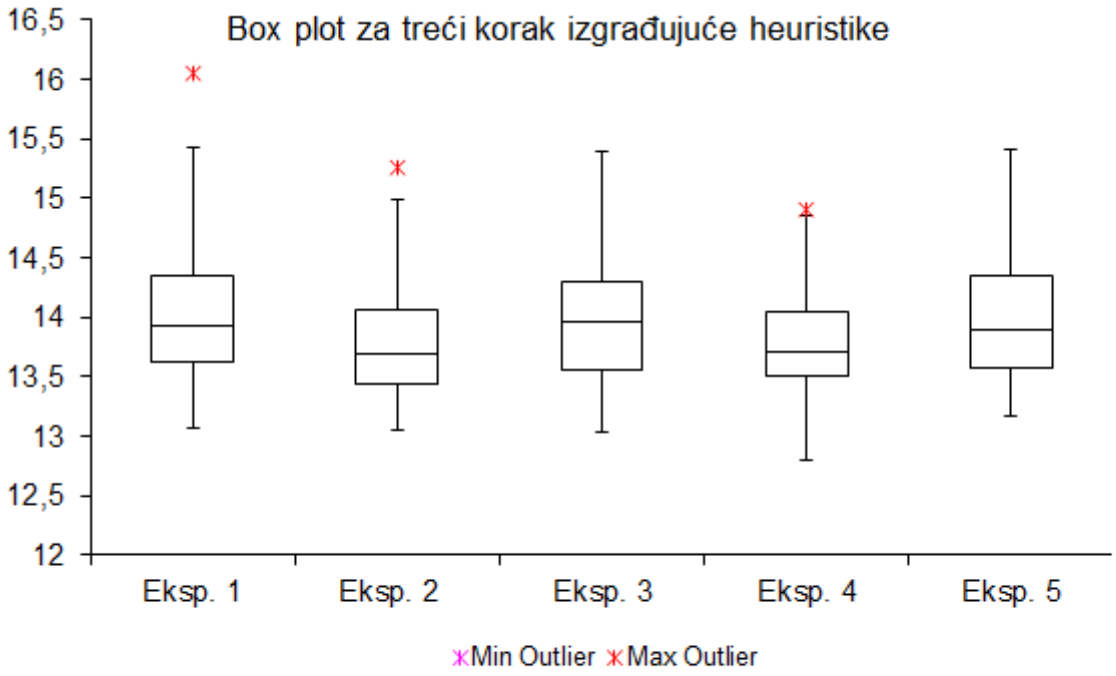
Tablica 6.18: Eksperimenti koji predstavljaju treći korak izgrađujuće strategije

Broj eksperimenta	Skup korištenih operatora mutacije
1	komplementirajuća mutacija, smanjujuća mutacija, Gaussova mutacija
2	komplementirajuća mutacija, smanjujuća mutacija, podizajuća mutacija
3	komplementirajuća mutacija, smanjujuća mutacija, nadomještajuća mutacija
4	komplementirajuća mutacija, smanjujuća mutacija, permutirajuća mutacija
5	komplementirajuća mutacija, smanjujuća mutacija, mutacija podstabla

Tablica 6.19 prikazuje rezultate dobivene u trećem koralu izgrađujuće heuristike. Iz prikazanih eksperimenata vidljivo je kako je eksperiment 4 postigao ponešto bolje rezultate od ostalih eksperimenata. To je također vidljivo i na slici 6.11 koja prikazuje *box plot* prikaz dobivenih rezultata.

Tablica 6.19: Rezultati za treći korak izgrađujuće strategije za operatore mutacije

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	14,024	13,063	16,049	13,921	0,6074
2	13,805	13,056	**15,260**	**13,688**	0,4903
3	13,950	13,036	15,397	13,96	0,5425
4	**13,770**	**12,799**	15,899	13,714	**0,4517**
5	14,001	13,172	15,415	13,890	0,5406



Slika 6.11: *Box plot* za treći korak izgrađujuće strategije za operatore mutacije

Kako su za najbolji skup iz trećeg koraka postignuti bolji rezultati od rezultata dobivenih za najbolji skup u drugom koraku, ovaj postupak prelazi u idući korak. Kako u tom novom koraku nije pronađen skup koji je postigao bolje rezultate od rezultata dobivenih u trećem koraku, rezultati za taj korak se neće posebno prikazivati. U konačnici je kao optimalan skup kroz izgrađujuću heuristiku određen skup koji se sastoji od komplementirajuće mutacije, smanjujuće mutacije i permutirajuće mutacije.

Nakon što je korištenjem izgrađujuće heuristike pronađen optimalan skup, isto to će se napraviti i korištenjem razgrađujuće heuristike. Tablica 6.20 prikazuje popis eksperimenata koji su obavljeni u sklopu prvog koraka razgrađujuće heuristike. Zbog preglednosti u ove rezultate je uklopljen i rezultat koji je dobiven i za skup koji se sastoji od svih operatora mutacija, označen kao eksperiment 1.

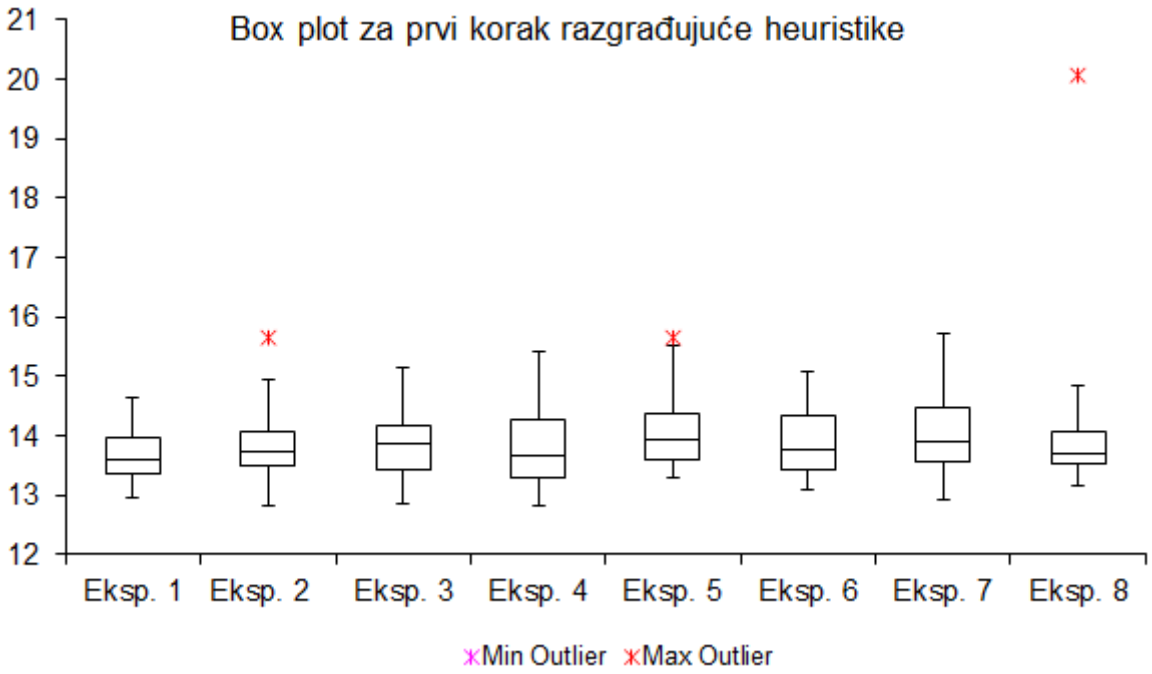
Tablica 6.20: Eksperimenti koji predstavljaju prvi korak razgrađujuće strategije

Broj eksperimenta	Skup korištenih operatora mutacije
1	komplementirajuća mutacija, gaussova mutacija, podizajuća mutacija, permutirajuća mutacija, nadomještajuća mutacija, smanjujuća mutacija, mutacija podstabla
2	komplementirajuća mutacija, podizajuća mutacija, permutirajuća mutacija, nadomještajuća mutacija, smanjujuća mutacija, mutacija podstabla
3	komplementirajuća mutacija, Gaussova mutacija, permutirajuća mutacija, nadomještajuća mutacija, smanjujuća mutacija, mutacija podstabla
4	Gaussova mutacija, podizajuća mutacija, permutirajuća mutacija, nadomještajuća mutacija, smanjujuća mutacija, mutacija podstabla
5	komplementirajuća mutacija, Gaussova mutacija, podizajuća mutacija, permutirajuća mutacija, smanjujuća mutacija, mutacija podstabla
6	komplementirajuća mutacija, Gaussova mutacija, podizajuća mutacija, nadomještajuća mutacija, smanjujuća mutacija, mutacija podstabla
7	komplementirajuća mutacija, Gaussova mutacija, podizajuća mutacija, permutirajuća mutacija, nadomještajuća mutacija, mutacija podstabla
8	komplementirajuća mutacija, Gaussova mutacija, podizajuća mutacija, permutirajuća mutacija, nadomještajuća mutacija, smanjujuća mutacija

Tablica 6.21 prikazuje rezultate koji su dobiveni za prvi korak razgrađujuće heuristike. Kao što je vidljivo iz rezultata, niti jedan eksperiment nije postigao bolji iznos srednje vrijednosti od skupa koji se sastoji od svih operatora. To ujedno znači da se ovaj postupak prekida odmah u prvom koraku. Slika 6.12 prikazuje *box plot* prikaz navedenih rješenja. Iz ove slike se može vidjeti kako su svi eksperimenti po iznosu dobrote dosta raspršeniji od rezultata koji su postignuti za eksperiment 1 u kojem je korišten potpun skup operatora mutacije. Kao optimalan skup u ovoj heuristici pokazao se skup koji je sastavljen od svih isprobanih genetskih operatora mutacije.

Tablica 6.21: Rezultati za prvi korak razgrađujuće strategije kod operatora mutacije

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	**13,657**	12,962	**14,624**	**13,602**	**0,4257**
2	13,845	**12,806**	15,6532	13,7201	0,6152
3	13,835	12,841	15,147	13,854	0,5228
4	13,843	12,811	15,398	13,662	0,6786
5	14,042	13,273	15,652	13,913	0,5453
6	13,890	13,086	15,074	13,773	0,5660
7	14,031	12,938	15,709	13,889	0,6232
8	13,961	13,139	20,077	13,695	1,0146



Slika 6.12: *Box plot* za prvi korak razgrađujuće strategije za operatore mutacije

Nakon obavljene izgrađujuće i razgrađujuće heuristike za pronalaženjem optimalnog skupa genetskih operatora mutacije, potrebno je još između dva rješenja, od kojih je po jedno dobiveno za svaku heuristiku, odabrati ono koje je postiglo bolje rezultate.

Iz rezultata se može zaključiti kako je to upravo skup koji se sastoji od svih isprobanih genetskih operatora mutacije. Posljedično, taj skup je odabran kao optimalan skup genetskih operatora mutacije.

6.1.5. Rezultati za sve kriterije

U ovom potpoglavlju biti će prikazane vrijednosti rezultata koje su dobivene za svaki pojedini kriterij ocjene rasporeda korištenjem optimalnih parametara određenih kroz prethodno opisane eksperimente.

Tablica 6.22: Rezultati po različitim kriterijima ocjene rasporeda

Kriterij	Vrijednost				
	avg	min	max	medijan	stdev
Twt	13,657	12,962	14,624	13,602	0,4257
Nwt	7,0054	6,3842	7,9387	7,0050	0,3261
Fwt	155,29	153,79	158,56	155,01	0,8794
$Cmax$	38,290	38,018	38,679	38,264	0,1504

6.2. Rezultati optimizacije funkcijskim čvorovima

U ovom potpoglavlju prikazat će se rezultati dobiveni korištenjem novih funkcijskih čvorova. Kao što je to bio slučaj i kod određivanja optimalnog skupa genetskih operatora križanja i mutacije, za određivanja optimalnog skupa funkcijskih čvorova korištene su izgrađujuća i razgrađujuća heuristika. Najprije će se prikazati rezultati dobiveni za izgrađujuću heuristiku.

Tablica 6.23 prikazuje prvi korak izgrađujuće heuristike za određivanje optimalnog skupa funkcijskih čvorova. Zbog preglednosti u rezultate je uvršten i eksperiment koji je obavljen nad polaznim skupom operatora te je taj eksperiment označen s brojem 1. Kao što se može vidjeti, osnovni skup operatora sačinjen je od aritmetičkih operatora zbrajanja, oduzimanja, množenja i dijeljenja.

Tablica 6.23: Eksperimenti koji predstavljaju prvi korak izgrađujuće strategije za funkcijske čvorove

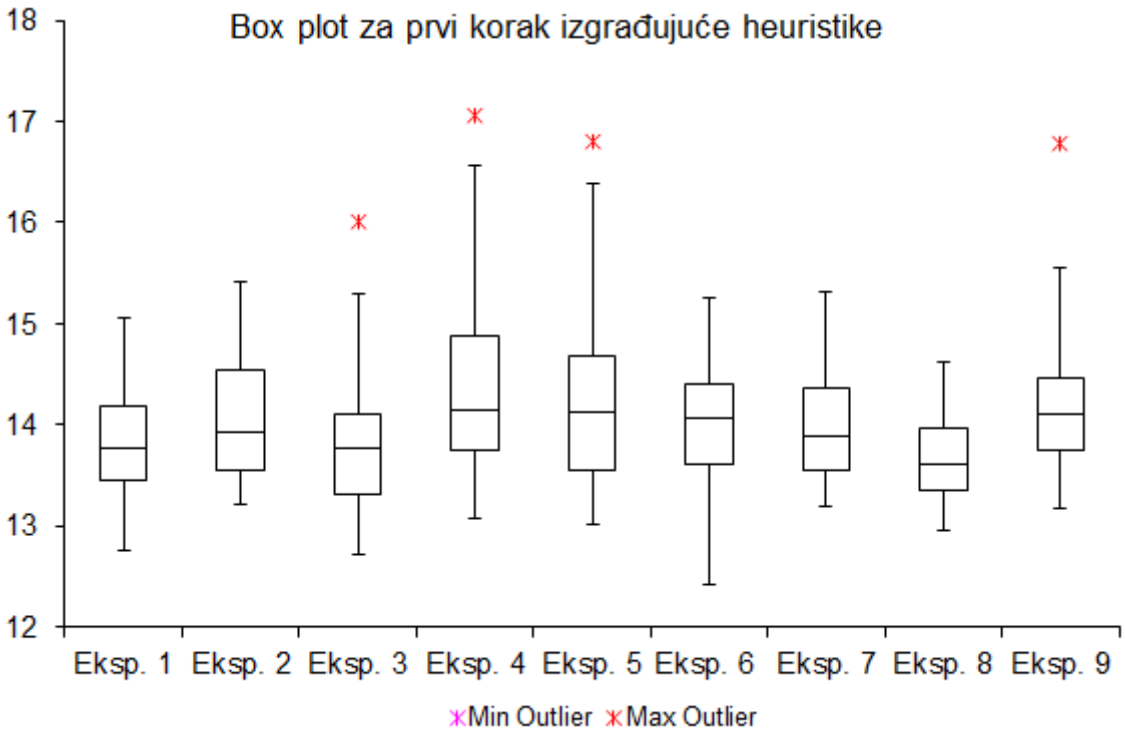
Broj eksperimenta	Skup korištenih funkcijskih čvorova
1	+, -, *, /
2	+, -, *, /, ABS
3	+, -, *, / , AVG
4	+, -, *, / , $IFGT$
5	+, -, *, / , $IFLT$
6	+, -, *, /, MAX
7	+, -, *, /, MIN
8	+, -, *, /, POS
9	+, -, *, / , SQR

Tablica 6.26 prikazuje rezultate koji su dobiveni za svaki pojedini obavljeni eksperiment. Kao što je iz tablice vidljivo, najbolji rezultate za sve kriterije, osim minimalne vrijednosti dobrote, postižu se za eksperiment 8 koji predstavlja skup funkcijskih operatora koji se sastoji od funkcijskih čvorova $+$, $-$, $*$, $/$ i POS. Iz tog razloga upravo je ovaj skup odabran kao optimalan u ovom koraku algoritma. Prije nego što se može krenuti na iduću iteraciju potrebno je provjeriti je li postignut napredak u odnosu na osnovni skup operatora. Iz tablice se može vidjeti kako je taj uvjet zadovoljen i iz tog razloga se kreće u drugi korak heuristike u kojem će se odabrani skup funkcijskih čvorova pokušati dodatno proširiti.

Tablica 6.24: Rezultati za prvi korak izgrađujuće strategije za funkcijske čvorove

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,843	12,752	15,053	13,774	0,5374
2	14,084	13,205	15,405	13,933	0,6349
3	13,796	**12,729**	16,016	13,762	0,6345
4	14,330	13,070	17,051	14,135	0,8177
5	14,223	13,007	16,797	14,133	0,8458
6	14,068	12,969	15,252	14,065	0,5095
7	13,982	13,185	15,316	13,882	0,5489
8	**13,657**	12,962	**14,624**	**13,6015**	**0,4257**
9	14,263	13,175	16,787	14,100	0,8173

Slika 6.13 prikazuje *box plot* prikaz rezultata prikazanih u tablici 6.26. Iz ove slike može se uočiti kako dodavanje pojedinih funkcijskih čvorova može na različite načine utjecati na strukturu dobivenih rješenja. Iz navedene slike se može zaključiti kako čvorovi koji predstavljaju grananja uzrokuju veliku raspršenost rješenja (što je vidljivo i po iznosu standardnih devijacija).



Slika 6.13: *Box plot* prikaz za prvi korak izgrađujuće strategije za funkcijske čvorove

Tablica 6.23 prikazuje eksperimente obavljene u drugom koraku izgrađujuće heuristike. U ovom koraku osnovni skup predstavlja skup za koji je utvrđeno da je optimalan u prošlom koraku heuristike.

Tablica 6.25: Eksperimenti koji predstavljaju drugi korak izgrađujuće strategije za funkcijske čvorove

Broj eksperimenta	Skup korištenih funkcijskih čvorova
1	+, -, *, /, POS, ABS
2	+, -, *, /, POS, AVG
3	+, -, *, /, POS, $IFGT$
4	+, -, *, /, POS, $IFLT$
5	+, -, *, /, POS, MAX
6	+, -, *, /, POS, MIN
7	+, -, *, /, POS, SQR

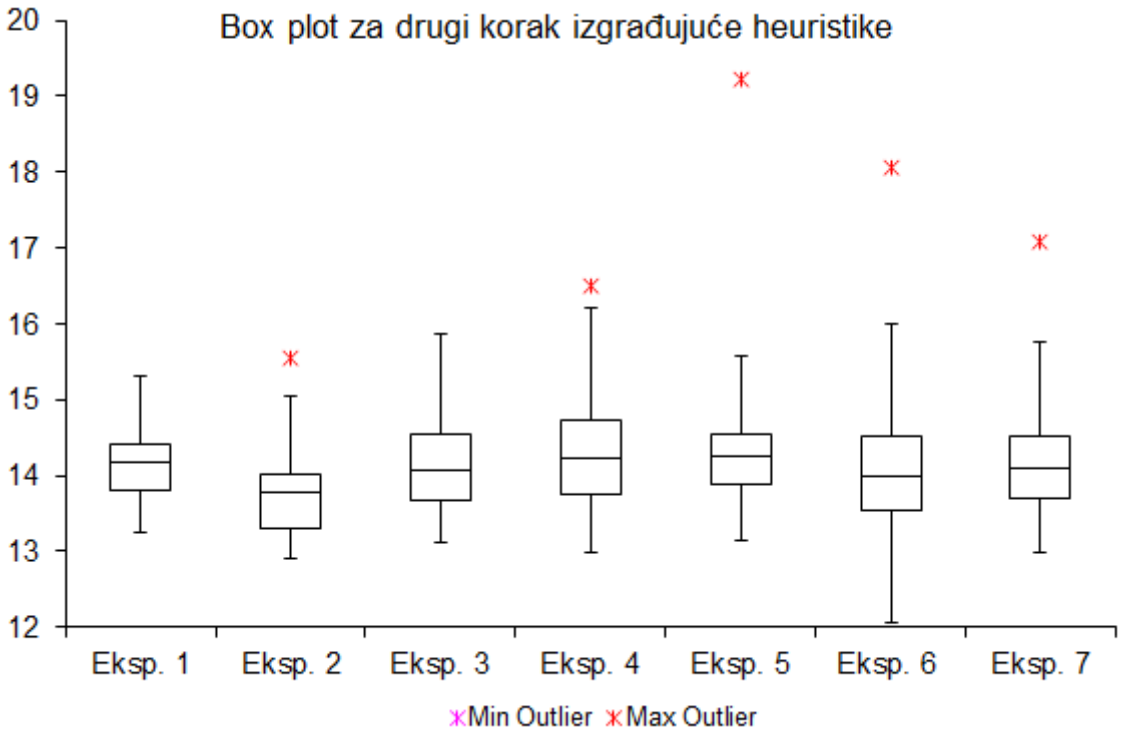
Tablica 6.26 prikazuje rezultate koji su dobiveni za drugi korak izgrađujuće heuristike. Iz tablice se može vidjeti kako niti jedan eksperiment ne postiže najbolje rezultate za većinu kriterija, no najnižu vrijednost za kriterij srednje vrijednosti postigao je eks-

periment 2 koji na optimalan skup određen u prošlom koraku dodaje još funkcijski čvor AVG. Skup operatora navedenog eksperimenta se iz tog razloga odabire kao optimalan u ovom koraku. No nažalost, odabrani skup ne predstavlja poboljšanje u odnosu na skup koji je odabran u prošlom koraku te se iz tog razloga algoritam prekida i ne izvodi se idući korak. Slika 6.14 prikazuje *box plot* prikaz dobivenih rezultata. Iz slike i tablice može se vidjeti kako skup koji koristi funkcijski čvor AVG postiže vidljivo bolji rezultat za kriterij medijana od ostalih korištenih funkcijskih čvorova. Ta činjenica svakako može predstavljati motivaciju za isprobavanjem čvorova koji su po funkcionalnosti koju obavljaju slični čvoru AVG.

Tablica 6.26: Rezultati za drugi korak izgrađujuće strategije za funkcijske čvorove

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	14,188	13,254	**15,297**	14,181	**0,5233**
2	**13,765**	12,901	15,539	**13,769**	0,5332
3	14,141	13,110	15,875	14,064	0,6583
4	14,330	12,987	16,499	14,231	0,7541
5	14,340	13,133	19,230	14,248	0,9231
6	14,087	**12,756**	18,048	13,996	0,8699
7	14,206	12,994	17,707	14,096	0,7028

Kao optimalan skup funkcijskih čvorova, korištenjem izgrađujuće heuristike, odabran je skup koji se sastoji od funkcijskih čvorova $+$, $-$, $*$, $/$ i POS koji je pronađen u prvom koraku heuristike.



Slika 6.14: *Box plot* prikaz za drugi korak izgrađujuće strategije za operatore križanja

Nakon što je odrađena izgrađujuća heuristika, korištenjem razgrađujuće heuristike probat će se ostvariti bolja rješenja. Tablica 6.27 prikazuje eksperimente koji su obavljeni u okviru prvog koraka razgrađujuće heuristike. Zbog preglednosti je u tablicu s rezultatima uvršten i osnovni skup sastavljen od svih funkcijskih čvorova označen kao eksperiment 1.

Tablica 6.27: Eksperimenti koji predstavljaju prvi korak razgrađujuće strategije za funkcijske čvorove

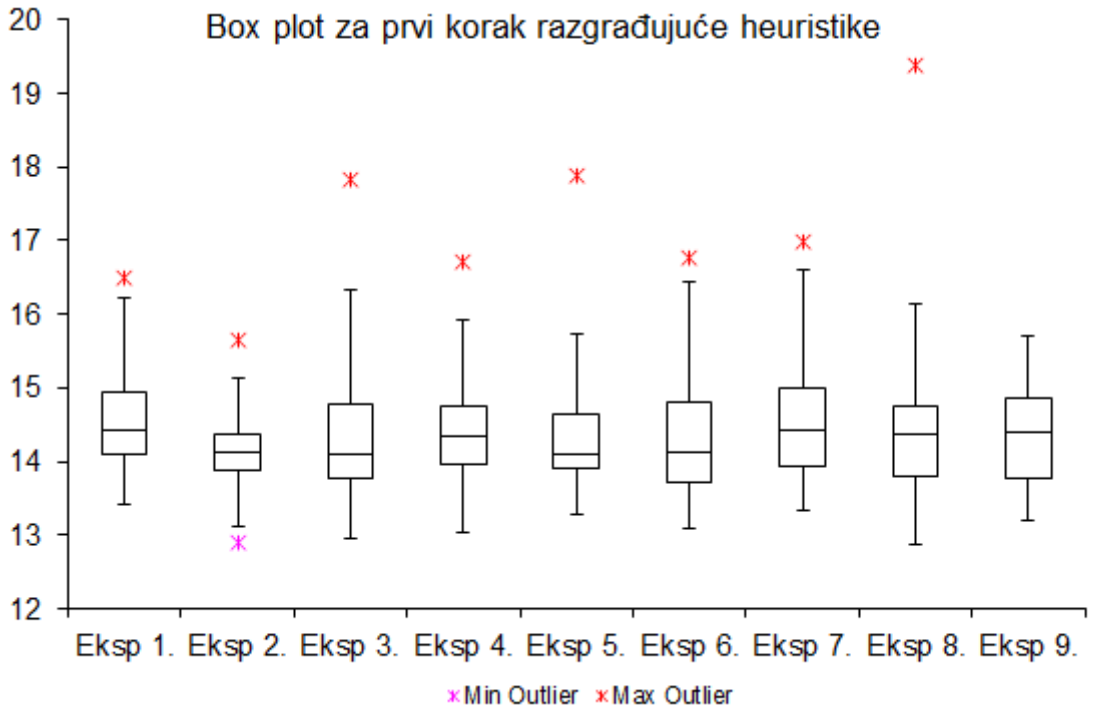
Broj eksperimenta	Skup korištenih funkcijskih čvorova
1	+, -, *, /, POS, ABS, AVG, MAX, MIN, $IFGT$, $IFLT$, SQR
2	+, -, *, /, POS, AVG, MAX, MIN, $IFGT$, $IFLT$, SQR
3	+, -, *, /, POS, ABS, MAX, MIN, $IFGT$, $IFLT$, SQR
4	+, -, *, /, POS, ABS, AVG, MAX, MIN, $IFLT$, SQR
5	+, -, *, /, POS, ABS, AVG, MAX, MIN, $IFGT$, SQR
6	+, -, *, /, POS, ABS, AVG, MIN, $IFGT$, $IFLT$, SQR
7	+, -, *, /, POS, ABS, AVG, MAX, $IFGT$, $IFLT$, SQR
8	+, -, *, /, ABS, AVG, MAX, MIN, $IFGT$, $IFLT$, SQR
9	+, -, *, /, POS, ABS, AVG, MAX, MIN, $IFGT$, $IFLT$

Tablica 6.30 prikazuje rezultate koji su dobiveni za prvi korak razgrađujuće heuristike. Iz tablice se može vidjeti kako je najniža vrijednost za kriterij srednje vrijednosti postignuta od strane eksperimenta 2. Iz tog razloga je upravo skup koji je korišten od strane tog eksperimenta (skup svih funkcijskih čvorova, osim čvora ABS) odabran kao optimalan skup u ovom koraku. Prije nego se može krenuti u drugi korak razgrađujuće heuristike, potrebno je provjeriti da li navedeni eksperiment postiže bolje rezultate od eksperimenta 1 koji koristi početni skup sastavljen od svih operatora. Iz tablice se može vidjeti kako je to zadovoljeno i odabrani skup se koristi kao početni skup u idućem koraku izgrađujuće heuristike.

Tablica 6.28: Rezultati za prvi korak razgrađujuće strategije za funkcijske čvorove

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	14,559	13,403	16,484	14,420	0,7395
2	**14,154**	12,8937	**15,639**	14,119	**0,5803**
3	14,331	12,940	17,8375	**14,094**	0,9259
4	14,411	13,041	16,721	14,328	0,7236
5	14,374	13,266	17,879	14,108	0,8248
6	14,352	13,093	16,762	14,136	0,9349
7	14,547	13,323	16,994	14,412	0,8114
8	14,526	**12,879**	19,3706	14,368	1,0864
9	14,382	13,201	15,705	14,399	0,6509

Slika 6.15 prikazuje *box plot* prikaz ranije spomenutih rezultata. Kao što se može vidjeti iz slike, kod ovih eksperimenata svojstvena je bila pojava dosta *outlier* vrijednosti. Iz tog se može zaključiti da skupovi koji se sastoje od većeg broja funkcijskih čvorova uzrokuju veću raspršenost rješenja



Slika 6.15: *Box plot* prikaz za prvi korak razgrađujuće strategije za funkcijske čvorove

Tablica 6.29 prikazuje eksperimente obavljene u okviru drugog koraka razgrađujuće heuristike.

Tablica 6.29: Eksperimenti koji predstavljaju drugi korak razgrađujuće strategije za funkcijske čvorove

Broj eksperimenta	Skup korištenih funkcijskih čvorova
1	+, -, *, /, POS, MAX, MIN, $IFGT$, $IFLT$, SQR
2	+, -, *, /, POS, AVG, MAX, MIN, $IFLT$, SQR
3	+, -, *, /, POS, AVG, MAX, MIN, $IFGT$, SQR
4	+, -, *, /, POS, AVG, MIN, $IFGT$, $IFLT$, SQR
5	+, -, *, /, POS, AVG, MAX, $IFGT$, $IFLT$, SQR
6	+, -, *, /, AVG, MIN, $IFGT$, $IFGT$, $IFLT$, SQR
7	+, -, *, /, POS, AVG, MAX, MIN, $IFGT$, $IFLT$

Tablica 6.30 prikazuje rezultate koji su dobiveni kroz drugi korak razgrađujuće heuristike. Iz rezultata se može vidjeti kako je po kriteriji srednje vrijednosti eksperiment 4 pokazao najbolje rezultate. Iz tog razloga je skup koji je korišten od navedenog eksperimenta odabran kao optimalan skup u ovom koraku heuristike. Nažalost, odabrani

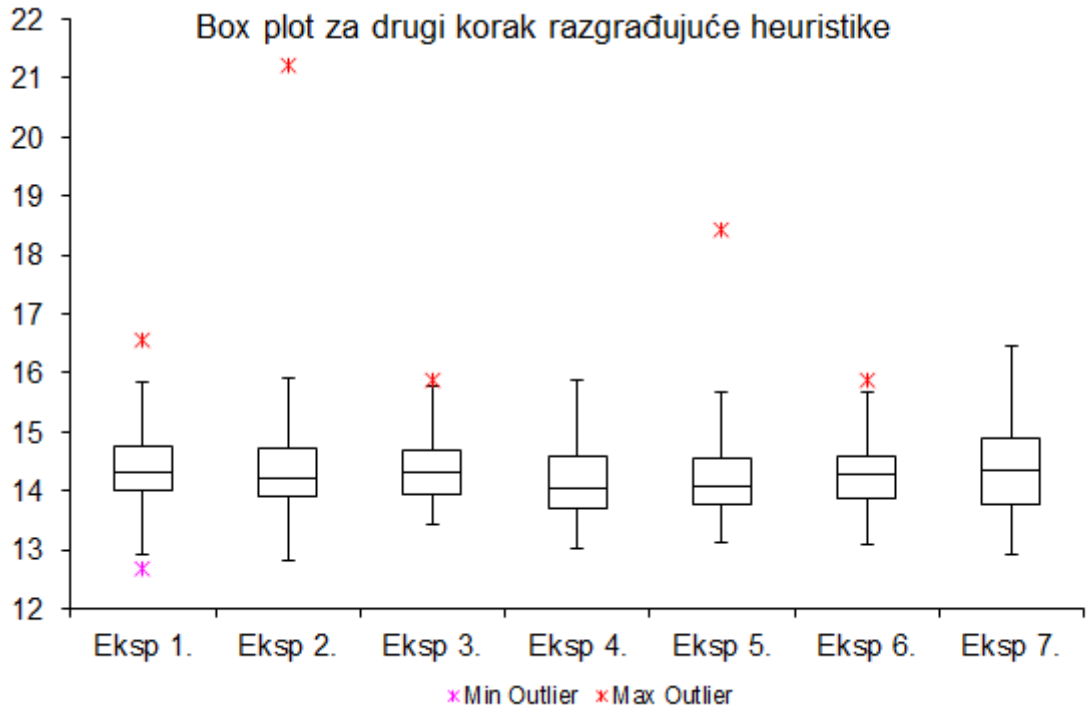
eksperiment u ovom koraku nije postigao rezultate koji su bolji od rezultata u prošlom koraku heuristike te se iz tog razloga izvođenje heuristike prekida. Slika 6.16 prikazuje *box plot* prikaz rezultata dobivenih u drugom koraku razgrađujuće heuristike. Može se vidjeti kako u ovim eksperimentima pojava *outlier* vrijednosti nije bila toliko česta kao u prošlom koraku.

Tablica 6.30: Rezultati za drugi korak razgrađujuće strategije za funkcijske čvorove

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	14,389	**12,691**	16,553	14,310	0,7422
2	14,449	12,804	21,225	14,207	1,2192
3	14,320	13,418	**15,865**	14,298	**0,4985**
4	**14,167**	13,022	**15,868**	14,055	0,6660
5	14,329	13,113	18,433	14,066	1,0001
6	14,250	13,099	15,892	14,266	0,5478
7	14,407	12,904	16,443	14,358	0,8208

Kako eksperiment za optimalan skup funkcijskih čvorova u ovom koraku nije postigao bolji iznos kriterija srednje vrijednosti od eksperimenta za optimalan skup funkcijskih čvorova koji je određen u prošlom koraku, heuristika se zaustavlja. Kao optimalan skup operatora u sklopu razgrađujuće heuristike pronađen je skup koji se sastoji od svih funkcijskih čvorova osim čvora ABS.

Nakon što su kroz ove dvije heuristike određena dva skupa, potrebno je između njih odabrati onaj koji je ima manji iznos kriterija srednje vrijednosti. Iz tablica s rezultatima se može vidjeti kako je skup funkcijskih čvorova koji je određen korištenjem izgrađujuće heuristike postigao bolje rezultate te je u konačnici taj skup odabran kao optimalan skup funkcijskih čvorova (sastavljen od funkcijskih čvorova $+$, $-$, $*$, $/$ i POS).



Slika 6.16: *Box plot* prikaz za drugi korak razgrađujuće strategije za funkcijske čvorove

6.2.1. Rezultati za sve kriterije

U ovom potpoglavlju biti će prikazani rezultati koji su dobiveni za svaki pojedini kriterij ocjene rasporeda korištenjem optimalnog skupa funkcijskih čvorova koji su određen kroz prethodno opisane eksperimente.

Tablica 6.31: Rezultati za sve kriterije ocjene rasporeda

Kriterij	Vrijednost				
	avg	min	max	medijan	stdev
Twt	13,657	12,962	14,624	13,602	0,4257
Nwt	7,0054	6,3842	7,9387	7,0049	0,3261
Fwt	155,29	153,79	158,56	155,01	0,879
$Cmax$	38,290	38,018	38,679	38,264	0,1504

6.3. Rezultati semantičkog genetskog programiranja

U ovom poglavlju usporedit će se vrijednosti rješenja koja su dobivena običnim genetskim programiranjem s rješenjima koja su dobivena semantičkim genetskim programiranjem. Oba navedena algoritma su u potpunosti jednaka, u smislu da koriste iste parametre i genetske operatore te se razlikuju samo po tome da su rješenja jednog algoritma semantički ispravna, dok kod drugog to ne mora biti slučaj. Parametri koji su se koristili prilikom izvođenja eksperimenata su jednaki onima prikazanim u tablici u poglavlju o optimizacijama, osim što je za operator križanja korišteno samo križanje s jednom točkom, dok je za operator mutacije korištena mutacija podstabla. Tablica 6.32 prikazuje eksperimente izvedene u sklopu ove optimizacije. U sljedećih nekoliko potpoglavlja prikazat će se usporedba između rezultata dobivenih od strane klasičnog genetskog programiranja i semantičkog genetskog programiranja, po pojedinom kriteriju ocjene rasporeda.

Tablica 6.32: Eksperimenti za semantičko genetsko programiranje

Broj eksperimenta	vrsta genetskog programiranja	kriterij ocjene rasporeda
1	klasično	Twt
2	semantičko	Twt
3	klasično	Nwt
4	semantičko	Nwt
5	klasično	Fwt
6	semantičko	Fwt
7	klasično	C_{max}
8	semantičko	C_{max}

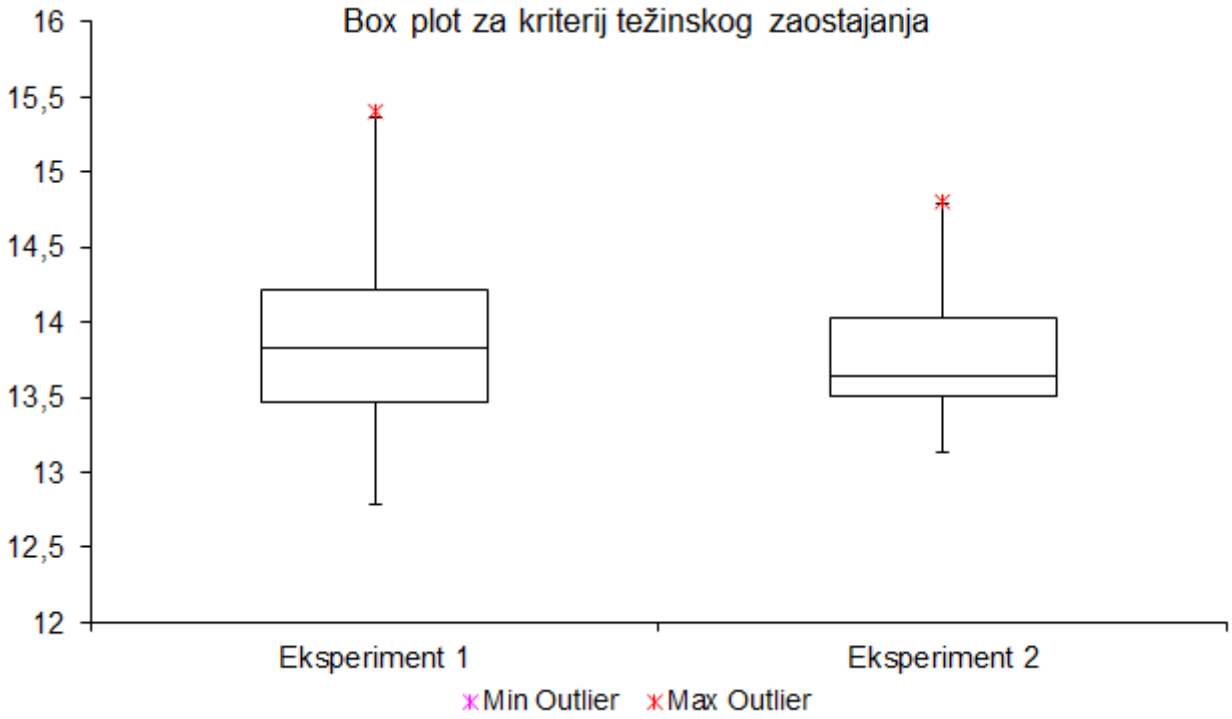
6.3.1. Kriterij težinskog zaostajanja

U ovom potpoglavlju detaljnije će se prikazati rezultati koji su dobiveni ako je kao kriterij ocjene rasporeda korišteno težinsko zaostajanje. Tablica 6.36 prikazuje usporedbu između rezultata dobivenih za kriterij težinskog zaostajanja od strane klasičnog genetskog programiranja (eksperiment jedan) i semantičkog genetskog programiranja (eksperiment dva). Kao što se može vidjeti semantičko genetsko programiranje postiže bolje rezultate u svim kriterijima osim u minimalnoj vrijednosti dobrote. Može se zaključiti kako je semantičko genetsko programiranje ponešto stabilnije od klasičnog genetskog programiranja jer su rezultati manje raspršeni. Iz tog razloga može

se zaključiti kako semantičko genetsko programiranje ima ponešto veću vjerojatnost da izgenerira bolja rješenje (jer se rješenja manje rasipaju), no u konačnici klasično genetsko programiranje uspijeva generirati rješenja koja su bolja po minimalnom iznosu dobrote. Slika 6.17 prikazuje *box plot* prikaz dobivenih rješenja za navedene eksperimente. Iz navedene slike se veoma dobro može vidjeti kako su rješenja dobivena od semantičkog genetskog programiranja više grupirana oko vlastitog medijana vrijednosti nego je to slučaj kod klasičnog genetskog programiranja. Kod klasičnog genetskog programiranja može se uočiti kako su ekstremne vrijednosti puno udaljenije od medijana.

Tablica 6.33: Rezultati za kriterij težinskog zaostajanja

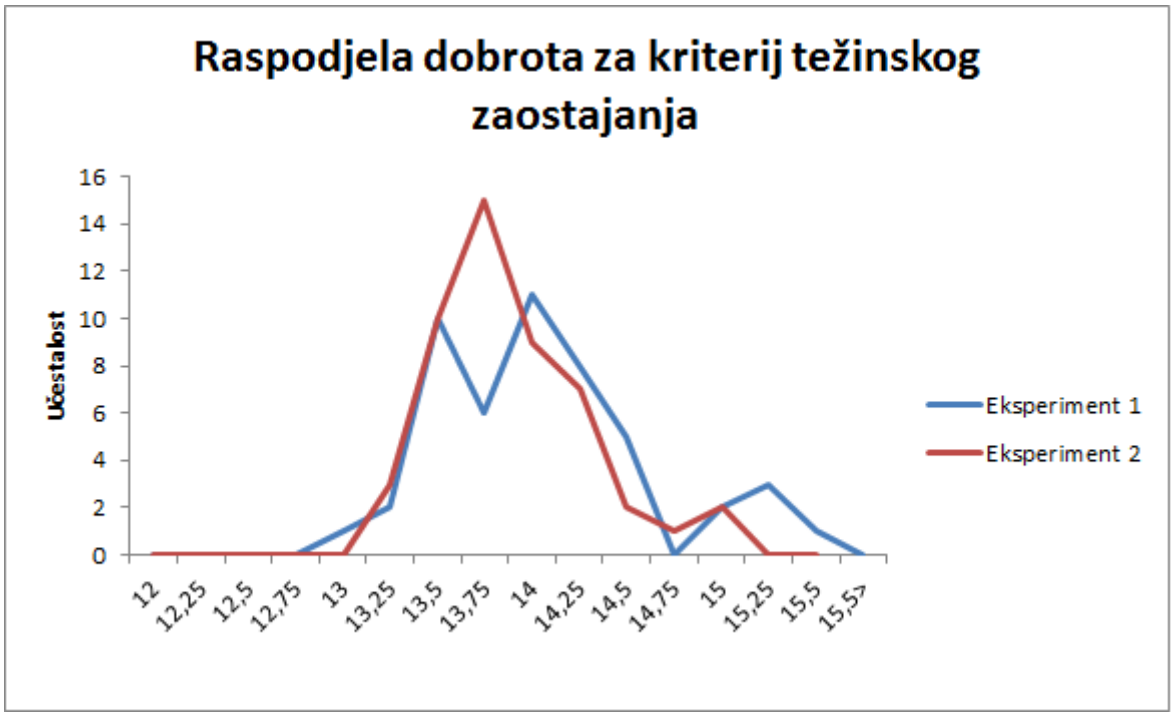
Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,928	**12,786**	15,404	13,830	0,5779
2	**13,776**	13,132	**14,808**	**13,640**	**0,4140**



Slika 6.17: *Box plot* za kriterij težinskog zaostajanja

Slika 6.18 prikazuje raspodjelu dobrota rješenja za pojedini eksperiment. Nave-

dena slika samo potvrđuje tvrdnje koje su bile iznesene ranije o raspršenosti rezultata pojedinih eksperimenata.



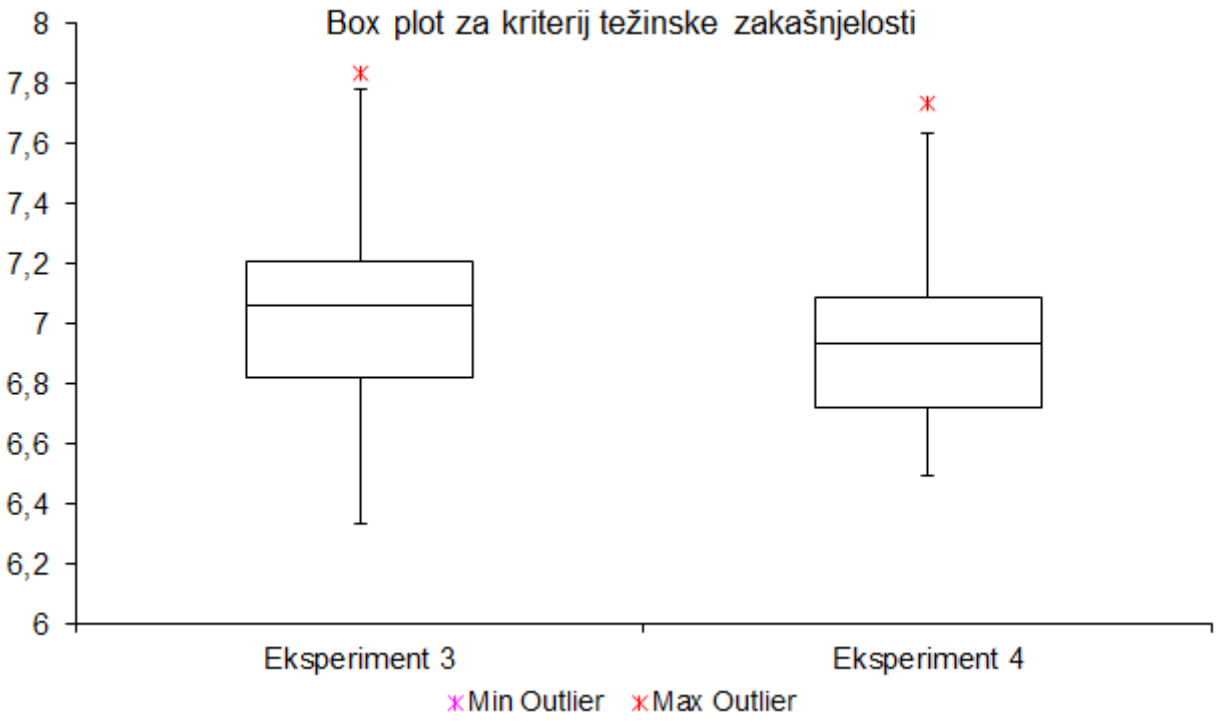
Slika 6.18: Raspodjela dobrota rješenja za kriterij težinskog zaostajanja

6.3.2. Kriterij težinske zakašnjelosti

U ovom potpoglavlju prikazat će se rezultati za kriterij težinske zakašnjelosti. Tablica 6.36 prikazuje usporedbu rezultata između klasičnog genetskog programiranja (eksperiment tri) i semantičkog genetskog programiranja (eksperiment četiri). Iz tablice se može uočiti slično ponašanje koje je bilo uočeno i za kriterij težinskog zaostajanja. Semantičko genetsko programiranje postiže bolje rezultate za sve kriterije osim za kriterij minimalne dobrote rješenja. To se ponovo može pripisati manjoj raspršenosti rješenja kod semantičkog genetskog programiranja, što se dobro može vidjeti i iz *box plot* prikaza na slici 6.19.

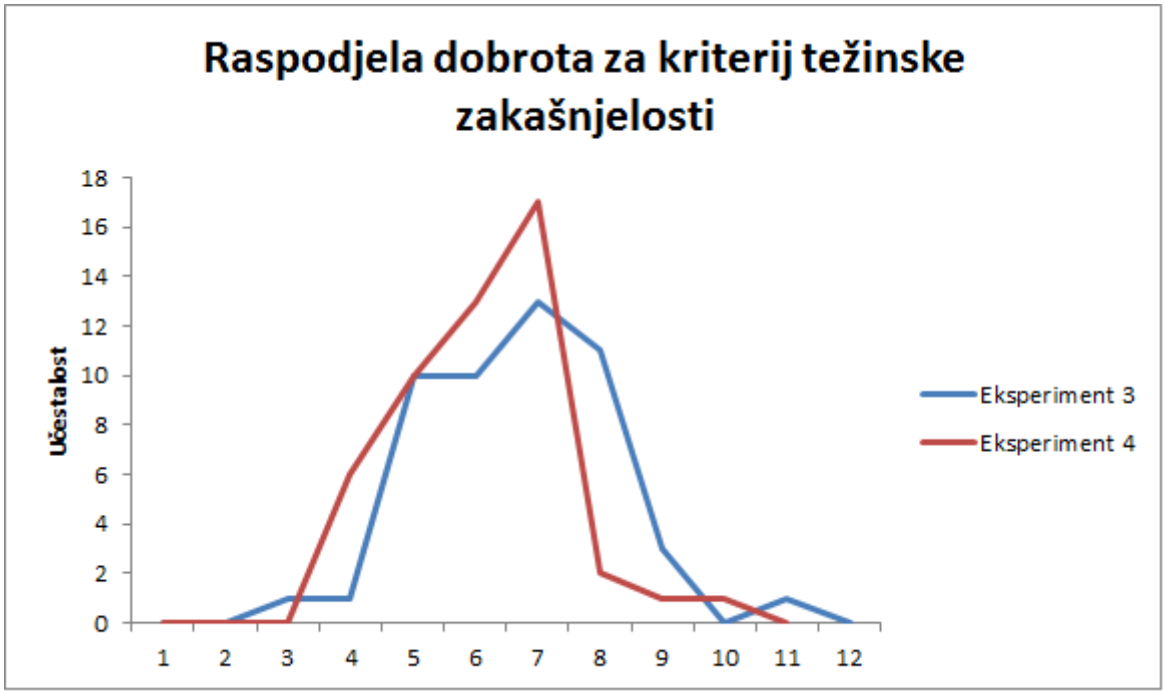
Tablica 6.34: Rezultati za kriterij težinske zakašnjelosti

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
3	7,0218	**6,3318**	7,8361	7,0580	0,2856
4	**6,9257**	6,4899	**7,7367**	**6,9340**	**0,2612**



Slika 6.19: *Box plot* za kriterij težinske zakašnjelosti

Slika 6.20 prikazuje histogram raspodjele dobrota za navedene eksperimente. Iz ove slike se također može vidjeti kako je raspršenost rješenja kod semantičkog genetskog programiranja manja.



Slika 6.20: Raspodjela dobrota rješenja za kriterij težinske zakašnjelosti

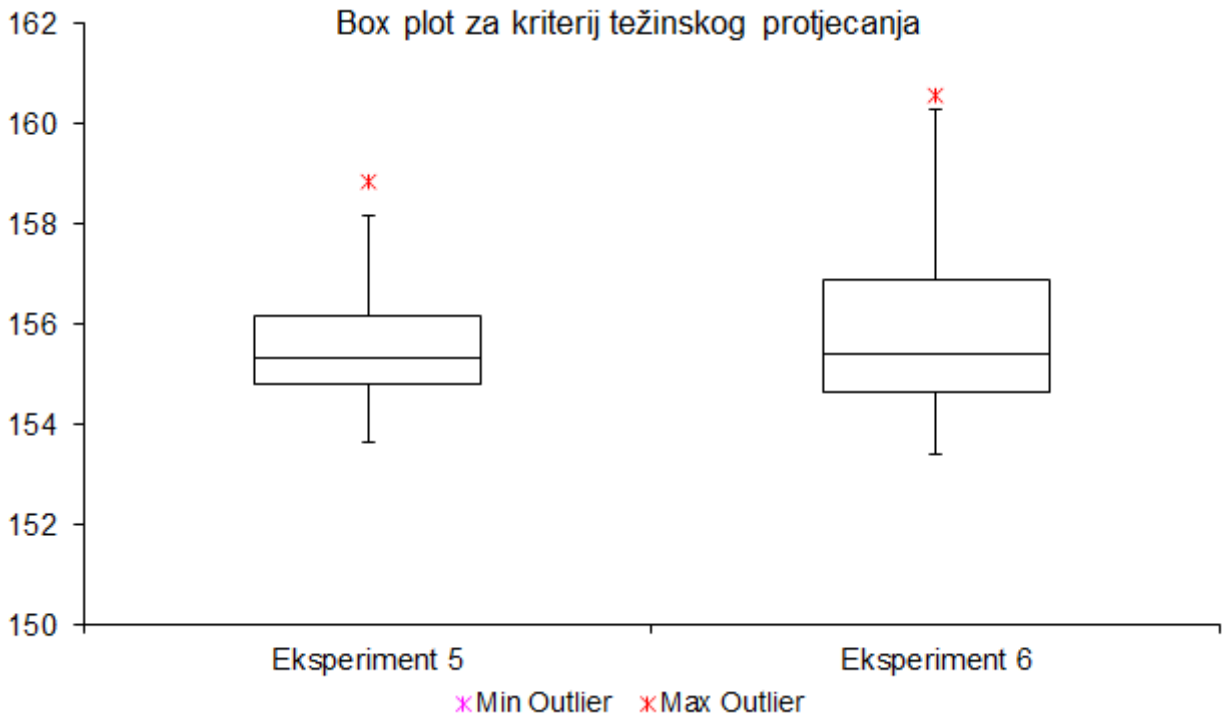
6.3.3. Kriterij težinskog protjecanja

U ovom potpoglavlju prikazat će se rezultati pri korištenju težinskog protjecanja kao kriterija za ocjenu rasporeda. Rezultati dobiveni za ovaj kriterij su ponešto iznenađujući. Naime, za razliku od prošla dva kriterija gdje je semantičko genetsko programiranje bilo bolje u svim kriterijima osim u kriteriju minimalne vrijednosti dobrote, ovdje je situacija upravo obrnuta. U ovom slučaju klasično genetsko programiranje daje rezultate koji su manje raspršeni po vrijednosti dobrote, dok je semantičko genetsko programiranje uspjelo pronaći rješenje koje je po ukupnoj vrijednosti dobrote najbolje. To se veoma dobro može vidjeti i na slici 6.21. Iz navedenih rezultata može se zaključiti kako ponašanje ovih algoritama ipak nije lako predvidivo i kako se za različite kriterije algoritmi mogu različito ponašati, što svakako uvodi dodatnu problematiku prilikom izrade algoritama.

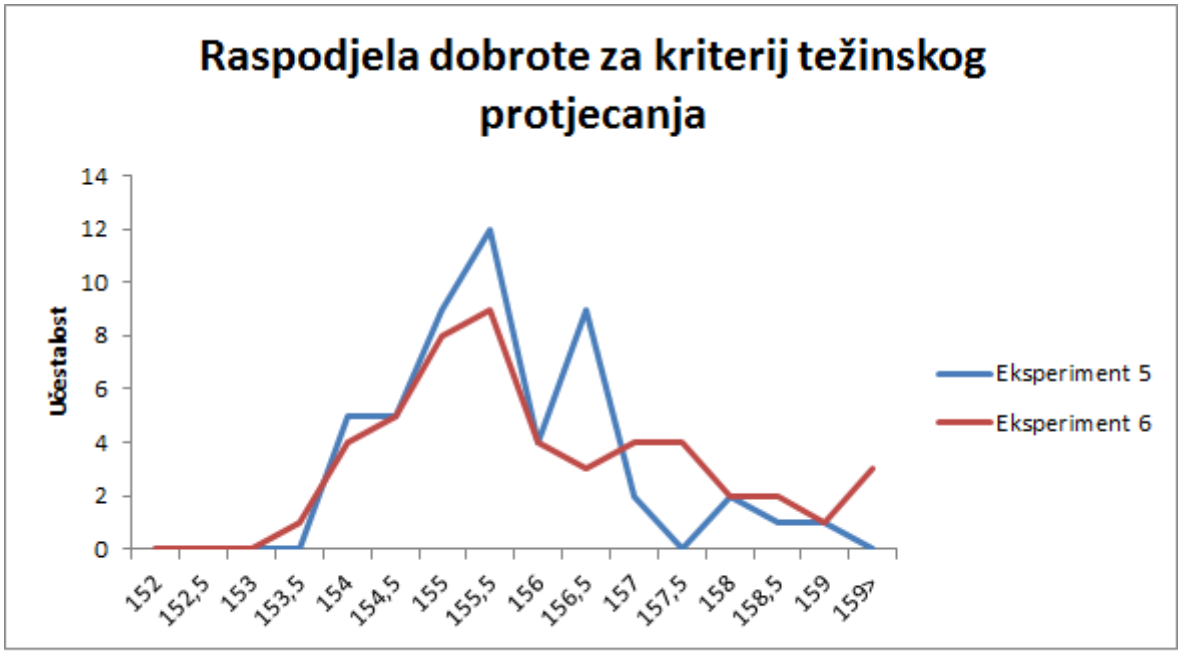
Tablica 6.35: Rezultati za kriterij težinskog protjecanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
5	**155,43**	153,64	**158,84**	**155,31**	**1,1116**
6	155,89	**153,39**	160,57	155,42	1,7246

Slika 6.22 prikazuje histogram raspodjele dobrote za kriterij težinskog protjecanja. I iz ove slike se također može vidjeti kako su sada rješenja ponešto raspršenija kod semantičkog genetskog programiranja.



Slika 6.21: *Box plot* za kriterij težinskog protjecanja



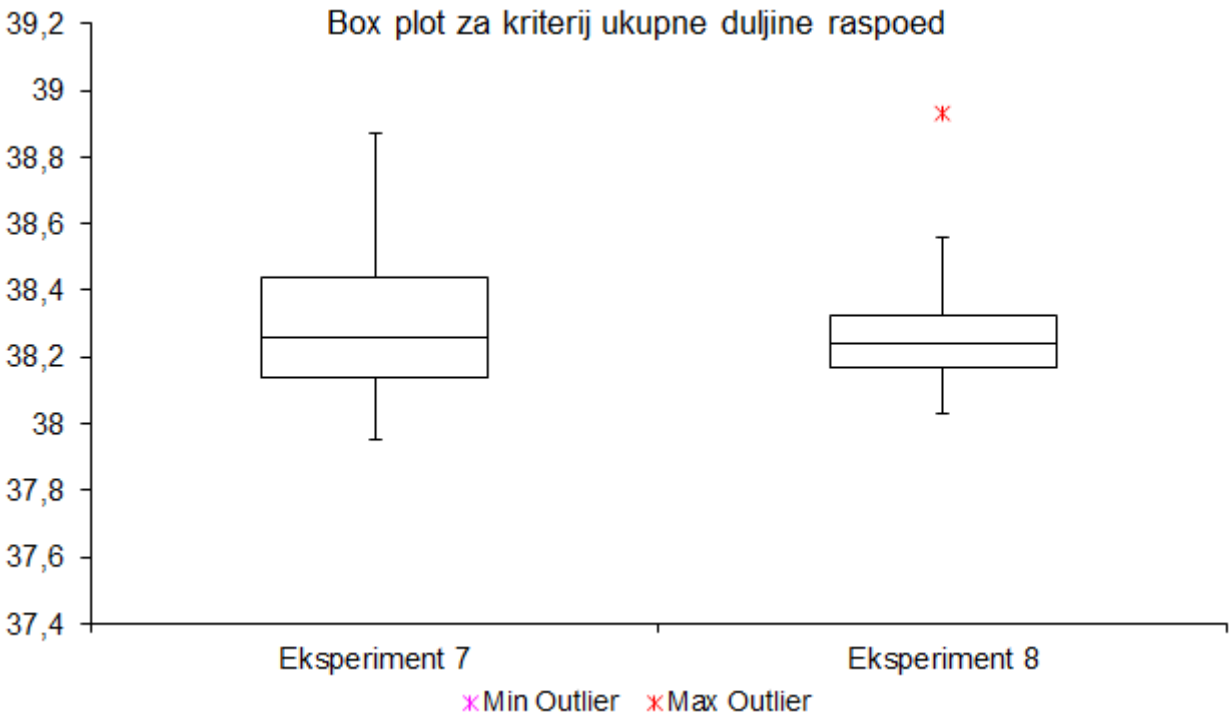
Slika 6.22: Raspodjela dobrota rješenja za kriterij težinskog protjecanja

6.3.4. Kriterij ukupne duljine rasporeda

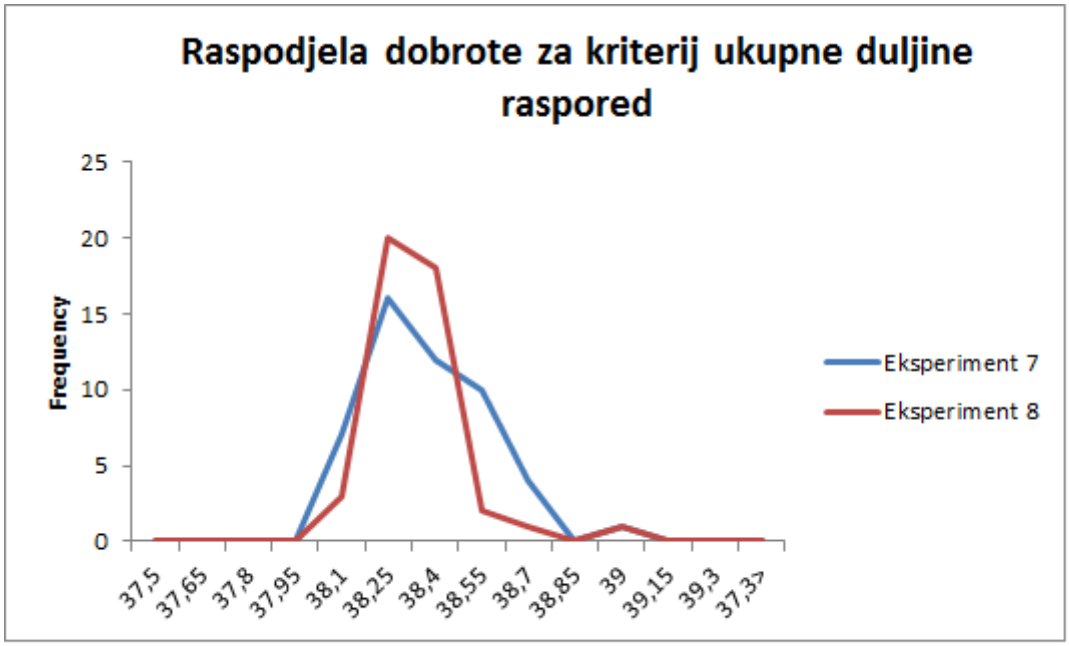
U ovom potpoglavlju prikazat će se rezultati koji su dobiveni prilikom korištenja ukupne duljine rasporeda kao kriterija ocjene samog rasporeda. Za ovaj kriterij su rezultati sličniji rezultatima dobivenim za prva dva navedena kriterija. Semantičko genetsko programiranje bolje je od klasičnog genetskog programiranja u svim kriterijima osim minimalne i maksimalne vrijednosti dobrote koje su postignute. To se može vidjeti i iz tablice 6.36. Kao što je vidljivo, klasično genetsko programiranje je postiglo najbolje rješenje po vrijednosti dobrote, dok je semantičko genetsko programiranje postiglo najgore rješenje po vrijednosti dobrote. No ako se prouči slika 6.23 koja prikazuje *box plot* prikaz za ove eksperimente i slika 6.24 koja prikazuje histogram vrijednosti dobrota postignutih rješenja, onda se može zaključiti kako se vrlo vjerojatno radi o vrijednosti koja predstavlja *outlier*. Razlozi pojave *outlier*-a mogu biti razne, od loše inicijalne populacije, do zapinjanja algoritma genetskog programiranja u nekom lokalnom optimumu. Što god razlog bio, iz priloženih slika se ipak vidi kako su ostala rješenja po vrijednosti svoje dobrote ipak više grupirana oko vlastitog medijana.

Tablica 6.36: Rezultati za kriterij ukupne duljine rasporeda

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
7	38,29412	**37,9514**	**38,8732**	38,25995	0,200633
8	**38,26833**	38,0326	38,92595	**38,2419**	**0,151596**



Slika 6.23: *Box plot* za kriterij ukupne duljine rasporeda



Slika 6.24: Raspodjela dobrota rješenja za kriterij ukupne duljine rasporeda

6.4. Rezultati optimizacije iterativnim pravilima raspoređivanja

U ovom potpoglavlju prikazat će se rezultati dobiveni korištenjem iterativnih pravila raspoređivanja u genetskom programiranju. Kako je za korištenje iterativnih pravila raspoređivanja potrebno uvesti nove primitive u korišten sustav, opet će biti potrebno pronaći optimalan skup primitiva za koje se postižu najbolji rezultati. Iz tog razloga i ovdje će biti korištene izgrađujuća i razgrađujuća heuristika kako bi se pronašao optimalan skup čvorova za algoritam genetskog programiranja koje koristi iterativna pravila raspoređivanja.

Tablica 6.37 prikazuje popis obavljenih eksperimenata u okviru prvog koraka izgrađujuće heuristike. Kako algoritam genetskog programiranja koji koristi iterativna pravila nema smisla izvoditi bez ijednog terminala koji je korišten od iterativnih pravila raspoređivanja, prazan skup koji ne sadrži niti jedan od navedenih terminala nije uzet u obzir.

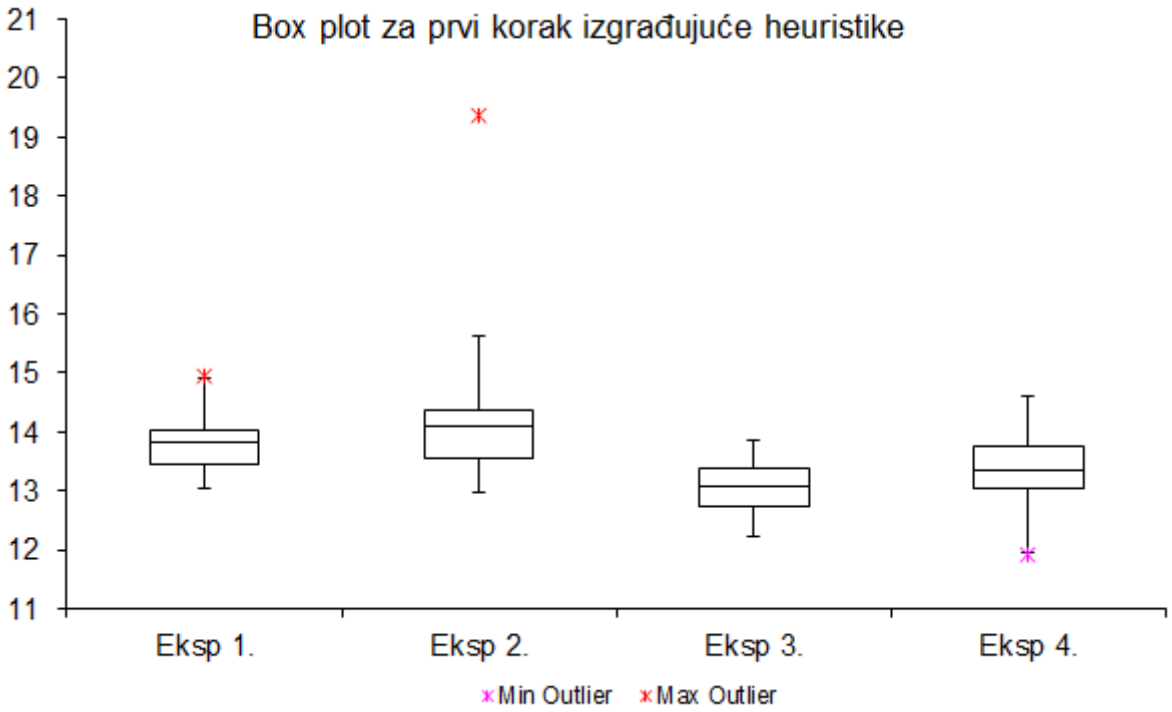
Tablica 6.37: Eksperimenti koji predstavljaju prvi korak izgrađujuće strategije za iterativna pravila raspoređivanja

Broj eksperimenta	Čvorovi za iterativna pravila raspoređivanja
1	$INDLATE$
2	$ISLATE$
3	$NLATE$
4	$LATENESS$

Tablica 6.37 prikazuje rezultate koji su dobiveni za prvi korak izgrađujuće heuristike. Iz rezultata se može vidjeti kako su čvorovi $INDLATE$ i $ISLATE$ postigli dosta loše rezultate koji nisu bolji od rezultata osnovne inačice algoritma genetskog programiranja. S druge strane čvorovi $LATENESS$ i $NLATE$ postižu mnogo bolje rezultate. Eksperiment 3, koji je sadržavao čvor $NLATE$ je postigao najbolju vrijednost za kriterij srednje vrijednosti. No najbolja jedinka po iznosu dobrote je pronađena u eksperimentu koji je sadržavao čvor $LATENESS$. Kako je ipak kriterij srednje vrijednosti do sada uvijek bio korišten za odabir optimalnog skupa, tako će se i ovdje temeljem ovog kriterija kao optimalan skup odabrati onaj koji se sadrži od čvora $NLATE$. Slika 6.25 prikazuje *box plot* prikaz za navedene rezultate.

Tablica 6.38: Rezultati za prvi korak izgrađujuče strategije za iterativna pravila raspoređivanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,779	13,028	14,936	13,815	**0,3994**
2	14,109	12,973	19,356	14,086	0,9260
3	**13,054**	12,230	**13,871**	**13,072**	0,4025
4	13,406	**11,911**	14,596	13,340	0,5885



Slika 6.25: *Box plot* prikaz za prvi korak izgrađuče strategije za iterativna pravila raspoređivanja

Tablica 6.39 prikazuje eksperimente koji su obavljeni u drugom koraku izgrađujuće heuristike.

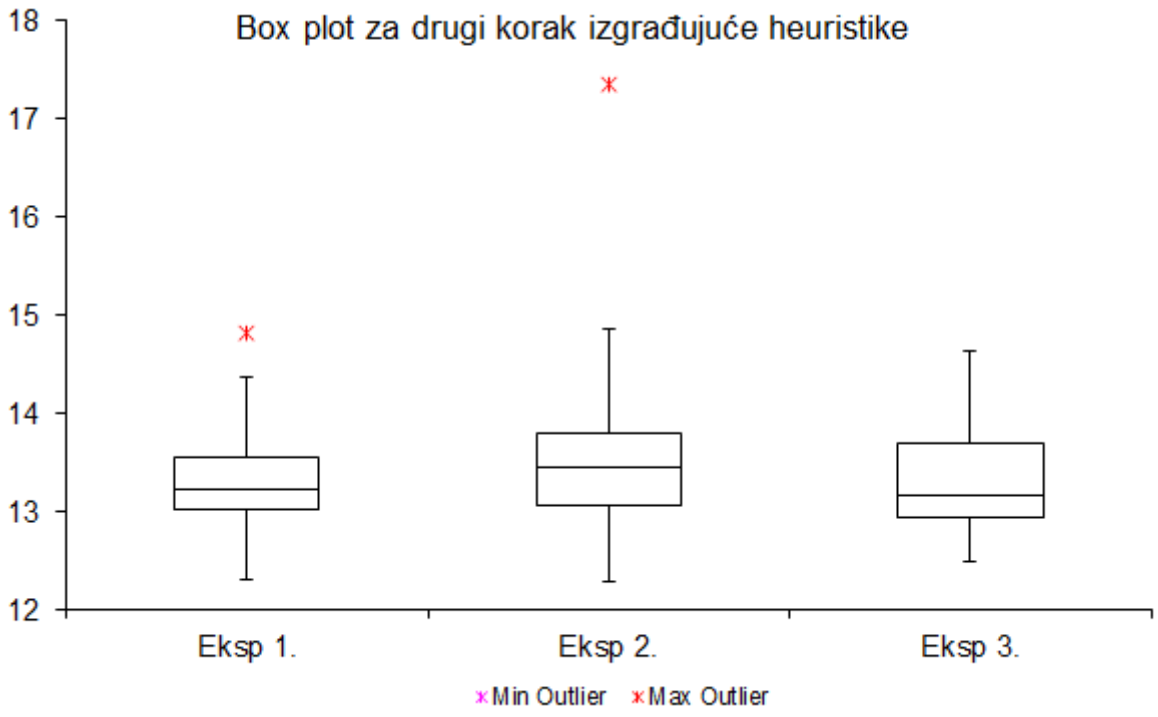
Tablica 6.39: Eksperimenti koji predstavljaju drugi korak izgrađujuće strategije za iterativna pravila raspoređivanja

Broj eksperimenta	Čvorovi za iterativan pravila raspoređivanja
1	$INDLATE, NLATE$
2	$ISLATE, NLATE$
3	$LATENESS, NLATE$

Tablica 6.40 prikazuje rezultate koji su dobiveni za eksperimente u ovom koraku heuristike. Može se vidjeti kako je eksperiment 1 postigao najbolje rezultate u ovom koraku heuristike (što se kriterija srednje vrijednosti tiče). Nažalost, rezultati koji su postignuti nisu bolji od rezultata koji su postignuti u prošlom koraku ove heuristike i zbog tog razloga se postupak zaustavlja i kao optimalan skup čvorova koja se koriste za iterativan pravila raspoređivanja odabran je skup koji se sastoji od čvora $NLATE$. Slika 6.26 prikazuje *box plot* prikaz za navedene rezultate.

Tablica 6.40: Rezultati za drugi korak izgrađujuće strategije za iterativna pravila raspoređivanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	**13,281**	12,297	14,827	13,221	**0,4723**
2	13,496	**12,290**	17,357	13,443	0,8097
3	13,341	12,489	**14,641**	**13,164**	0,5496



Slika 6.26: *Box plot* prikaz za drugi korak izgrađujuće strategije za iterativna pravila raspoređivanja

Tablica 6.41 predstavlja eksperimente koji su obavljeni u prvom koraku razgrađujuće heuristike. Zbog preglednosti je u rezultate dodan i eksperiment koji predstavlja skup koji se sastoji od svih čvorova korištenih prilikom iterativnih pravila raspoređivanja. Taj eksperiment je označen s brojem 1.

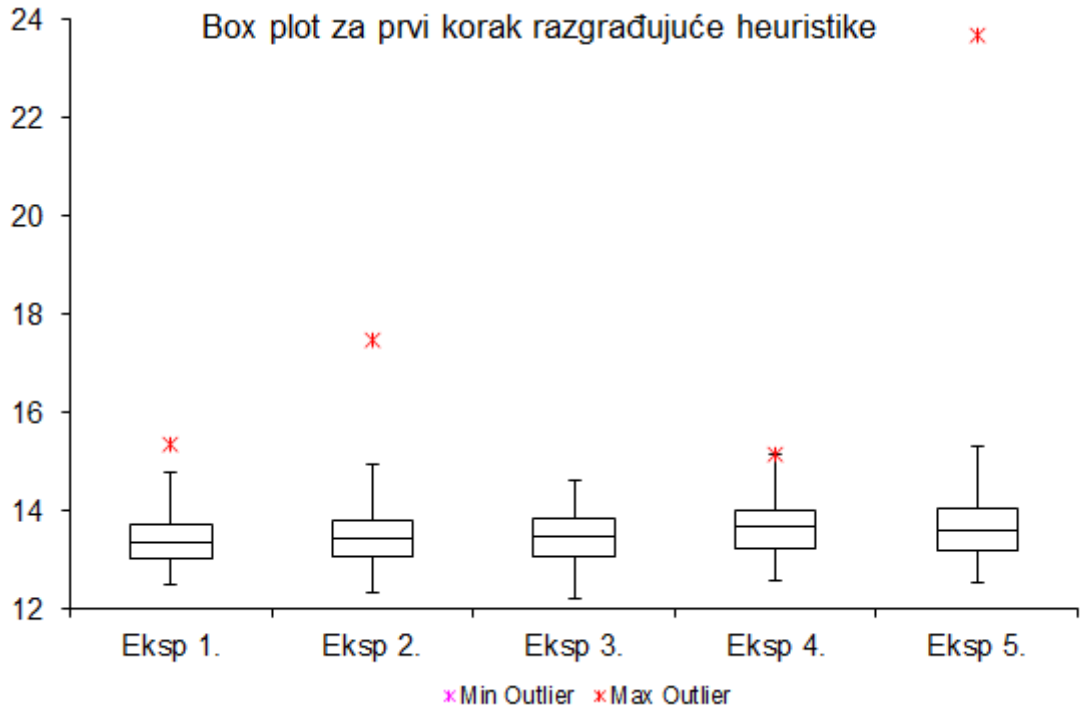
Tablica 6.41: Eksperimenti koji predstavljaju prvi korak razgrađujuće strategije za iterativna pravila raspoređivanja

Broj eksperimenta	Čvorovi za iterativan pravila raspoređivanja
1	$ISLATE$, $INDLATE$, $LATENESS$, $NLATE$
2	$ISLATE$, $LATENESS$, $NLATE$
3	$INDLATE$, $LATENESS$, $NLATE$
4	$ISLATE$, $INDLATE$, $NLATE$
5	$ISLATE$, $INDLATE$, $LATENESS$

Tablica 6.41 predstavlja rezultate ranije navedenih eksperimenata. Iz rezultata se može vidjeti kako je po iznosu kriterija srednje vrijednosti najbolji rezultat postignut od strane eksperimenta 1, što znači da niti jedan eksperiment koje je bio obavljen u prvom koraku heuristike nije postigao bolje rezultate od navedenog eksperimenta te se iz tog razloga heuristika prekida odmah u prvom koraku. Kao optimalan skup odabran je skup koji se sastoji od svih postojećih čvorova za iterativna pravila raspoređivanja. Slika 6.27 prikazuje *box plot* prikaz za navedene rezultate.

Tablica 6.42: Rezultati za prvi korak razgrađujuće strategije za iterativna pravila raspoređivanja

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	**13,459**	12,497	15,354	**13,367**	0,6515
2	13,511	12,323	17,467	13,412	0,7638
3	13,469	**12,211**	**14,629**	13,475	0,6014
4	13,683	12,561	15,159	13,673	**0,5985**
5	13,792	12,528	23,663	13,581	1,1520



Slika 6.27: *Box plot* prikaz za prvi korak razgrađujuće strategije za iterativna pravila raspoređivanja

Nakon što su obavljene obje heuristike potrebno je između njihova dva rezultata odabrati onaj koji je postigao manju vrijednost po kriteriju srednje vrijednosti. Iz tablica se može vidjeti kako je ipak skup koji se sastoji samo od čvora $NLATE$ postigao značajno bolje rezultate i iz tog razloga je onda on odabran kao optimalan skup čvorova korištenih od strane iterativnih pravila raspoređivanja.

6.4.1. Rezultati za sve kriterije

U ovom potpoglavlju biti će prikazani rezultati koji su dobiveni za svaki pojedini kriterij ocjene rasporeda korištenjem genetskog programiranja s iterativnim pravilima raspoređivanja.

Tablica 6.43: Rezultati sa sve kriterije ocjene rasporeda

Kriterij	Kriterij				
	avg	min	max	medijan	stdev
Twt	13,054	12,230	13,871	13,072	0,4025
Nwt	6,7027	6,1519	7,3131	6,6961	0,2533
Fwt	154,08	151,56	156,86	153,90	1,0243
$Cmax$	38,034	37,736	38,448	37,999	0,1822

6.5. Rezultati optimizacije GEP-om

U ovom poglavlju prikazat će se rezultati koji su dobiveni korištenjem GEP prikaza jedinki. Ovaj prikaz jedinki ne koristi parametar dubine, već je veličina jedinke određena kroz broj gena u jedinci i veličinom glave gena. Iz tog razloga bilo je potrebno provesti optimizaciju i ovih parametara kako bi se odredile njihove optimalne vrijednosti. U prvom koraku bio je optimiran parametar koji određuje broj gena u jedinci, pri čemu je veličina glave gena bila fiksirana (na vrijednost 10). U drugom koraku bio je optimiran parametar koji određuje veličinu glave gena, pri čemu je broj gena bio postavljen na optimalnu veličinu koja je bila pronađena u prvom koraku. Kao kriterij ocjene rasporeda u ovim eksperimentima korišten je kriterij težinske zakašnjelosti. Nakon što su određene optimalne vrijednosti ova dva parametra, algoritam je iskorišten za optimizaciju sva četiri kriterija ocjene rasporeda.

6.5.1. Optimizacija broja gena u jedinci

U ovom potpoglavlju prikazat će se eksperimenti obavljeni u okviru optimizacije broja gena u jedinci. Tablica 6.44 prikazuje eksperimente obavljene u okviru ove optimizacije. Iz tablice se može vidjeti kako su isprobane jedinke koje su bile sastavljene od 2, 3, 4 i 5 gena.

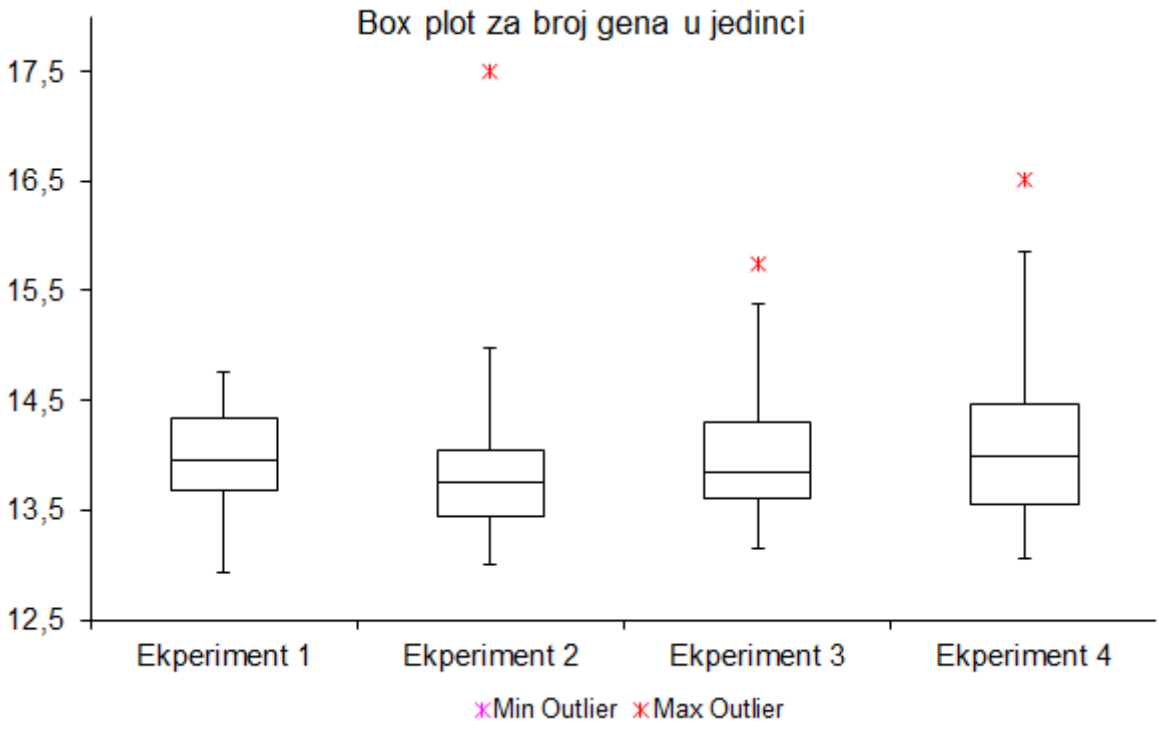
Tablica 6.44: Eksperimenti za određivanje broja gena u jedinci

Broj eksperimenta	Broj gena u jedinci	Veličina glave gena
1	2	10
2	3	10
3	4	10
4	5	10

Tablica 6.45 prikazuje rezultate koji su dobiveni za optimizaciju broja gena u jedinci. Kako je iz tablice vidljivo, najbolji rezultati su dobiveni za jedinke sastavljene od dva i tri gena. Jedinke koje su bile sastavljene od više gena su postigle nešto lošija rješenja. Ako se usporede rezultati dobiveni između jedinki koje su bile sastavljene od 2 i 3 gena može se uočiti da je za jedinke sastavljene od 3 gena postignuta manja vrijednost po kriteriju srednje vrijednosti i medijana. S druge strane, najjednostavnije jedinke (one koje su se sastojale samo od 2 gena) su pokazale svojstvo najmanjeg rasipanja te je korištenjem ovih jedinki pronađeno najbolje rješenje. No bez obzira na to, kao vrijednost parametra odabran je broj gena od 3. Slika 6.28 prikazuje *box plot* prikaz navedenih rezultata. Iz ove slike se može vidjeti kako se eksperimenti zapravo jako malo razlikuju po vrijednosti najboljeg pronađenog rješenja, nego više po raspršenosti pronađenih rješenja i vrijednosti oko koje su ta rješenja centrirana.

Tablica 6.45: Rezultati za optimizaciju broja gena u jedinci

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	13,993	**12,935**	**14,757**	13,953	**0,4496**
2	**13,869**	13,005	17,504	**13,750**	0,7087
3	14,018	13,146	15,752	13,838	0,5944
4	14,043	13,051	16,517	13,990	0,6468



Slika 6.28: *Box plot* za optimizaciju broja gena u jedinci

6.5.2. Optimizacija veličine glave gena

U ovom potpoglavlju prikazat će se eksperimenti koji su obavljeni u svrhu optimizacije veličine glave gena. Isprobano je pet različitih veličina glave gena prikazane u tablici 6.46. Kako se iz tablice može vidjeti, kroz sve eksperimente broj gena bio je fiksiran na iznos od tri gena po jedinci.

Tablica 6.46: Eksperimenti za određivanje veličine glave gena u jedinci

Broj eksperimenta	Broj gena u jedinci	Veličina glave gena
1	3	6
2	3	8
3	3	10
4	3	12
5	3	14

Tablica 6.47 prikazuje rezultate koji su postignuti prilikom optimiranja veličine glave gena. Kao što se može vidjeti iz tablice, GEP postiže veoma dobre rezultate za veoma malene jedinke. Tako je primjerice eksperiment koji je sadržavao jedinke koje

su imale glavu gena veličine 6 postigao najbolja rješenja što se tiče kriterija srednje vrijednosti i medijana vrijednosti. S druge strane, najbolje rješenje je pronađeno od strane eksperimenta koji je koristio najsloženije jedinke. Kako je do sada za određivanje optimalnog iznosa nekog parametra uvijek bio korišten kriterij srednje vrijednosti, tako će se korištenjem toga kriterija i sada za optimalnu veličinu glave gena odabrati vrijednost 6. No u budućnosti bi svakako trebalo napraviti mnogo detaljniju pretragu parametara jer su broj gena i veličina glave gena međusobno veoma povezani parametri.

Tablica 6.47: Rezultati za optimizaciju veličine glave gena u jedinci

Broj eksperimenta	Kriterij				
	avg	min	max	medijan	stdev
1	**13,856**	13,058	15,139	**13,685**	0,5552
2	13,887	13,270	**14,843**	13,863	**0,4341**
3	13,869	13,005	17,504	13,750	0,7087
4	14,022	13,096	15,921	13,954	0,5914
5	13,963	**12,716**	15,985	13,899	0,6902

Slika 6.29 prikazuje *box plot* prikaz rješenja koja su dobivena prilikom optimiranja parametra koji predstavlja veličinu glave gena.

Slika 6.29: *Box plot* za optimizaciju veličine glave gena u jedinci

6.5.3. Rezultati za sve kriterije

U ovom potpoglavlju biti će prikazani rezultati koji su dobiveni za svaki pojedini kriterij ocjene rasporeda korištenjem GEP prikaza jedinki.

Tablica 6.48: Rezultati sa sve kriterije ocjene rasporeda

Kriterij	Kriterij				
	avg	min	max	medijan	stdev
Twt	13,856	12,058	15,139	13,685	0,5552
Nwt	6,9413	6,4399	7,5526	6,9253	0,2486
Fwt	154,96	153,53	158,09	154,83	1,0420
$Cmax$	38,217	37,945	38,729	38,219	0,1416

6.6. Ocjena rezultata

U ovom potpoglavlju međusobno će se usporediti rezultati dobiveni kroz ove različite optimizacije s rezultatima koji su dobiveni nekim drugim heurističkim postupcima. Tablica 6.49 prikazuje objedinjene rezultate svih isprobanih heuristika. Prva četiri retka prikazuju rezultate dobivene korištenjem nekih deterministčkih raspoređivaća koji su ranije opisani u radu. Iz tablice se može vidjeti kako su se najboljima pokazali Min-min i Sufferage postupci (ovisno o kriteriju). Min-max postupak postiže ponešto lošije rezultate, dok Max-min postiže najlošije rezultate za sve kriterije. U petom retku su prikazani rezultati za takozvani *search based* postupak [10]. Ovaj postupak je kombinacija genetskog algoritma i algoritma optimizacije kolonijom mrava na način da su za svaku instancu problema pokrenuta oba algoritma i od svih rješenja koja su dobivena kroz ova dva algoritma odabrano je ono najbolje (i to je napravljeno za svaki kriterij). Ostatak tablice predstavljaju rezultati koji su dobiveni korištenjem genetskog programiranja. Za svaku inačicu algoritma genetskog programiranja prikazano je četiri retka rezultata. Svaki od tih redaka predstavlja zaseban eksperiment u kojem je bio optimiran određen kriterij ocjene rasporeda. Za svaki taj eksperiment prikazani su rezultati za sve kriterije ocjene rasporeda (dakle za kriterij koji je optimiran ali i za sve ostale kriterije). Za svaki pojedini kriterij prikazane su tri vrijednosti: iznos kriterija za najbolje pronađeno rješenje, srednja vrijednost najboljih rješenja u svakom pokretanju, i iznos kriterija za najlošije pronađeno rješenje. U tablicu su također uvršteni i rezultati koji su dobiveni korištenjem osnovnog algoritma genetskog programiranja prije ikakvih optimizacija. Također je potrebno napomenuti da u tablici nisu navedeni rezultati koji su dobiveni za optimizaciju funkcijskim čvorovima jer su rezultati istovjetni onima koji su dobiveni prilikom optimizacije parametara (jer je prilikom optimizacije parametara nesvjesno korišten upravo optimalan skup funkcijskih čvorova). U nastavku poglavlja napravit će se detaljnija usporedba rezultata za svaki pojedini kriterij između svih navedenih heurističkih postupaka.

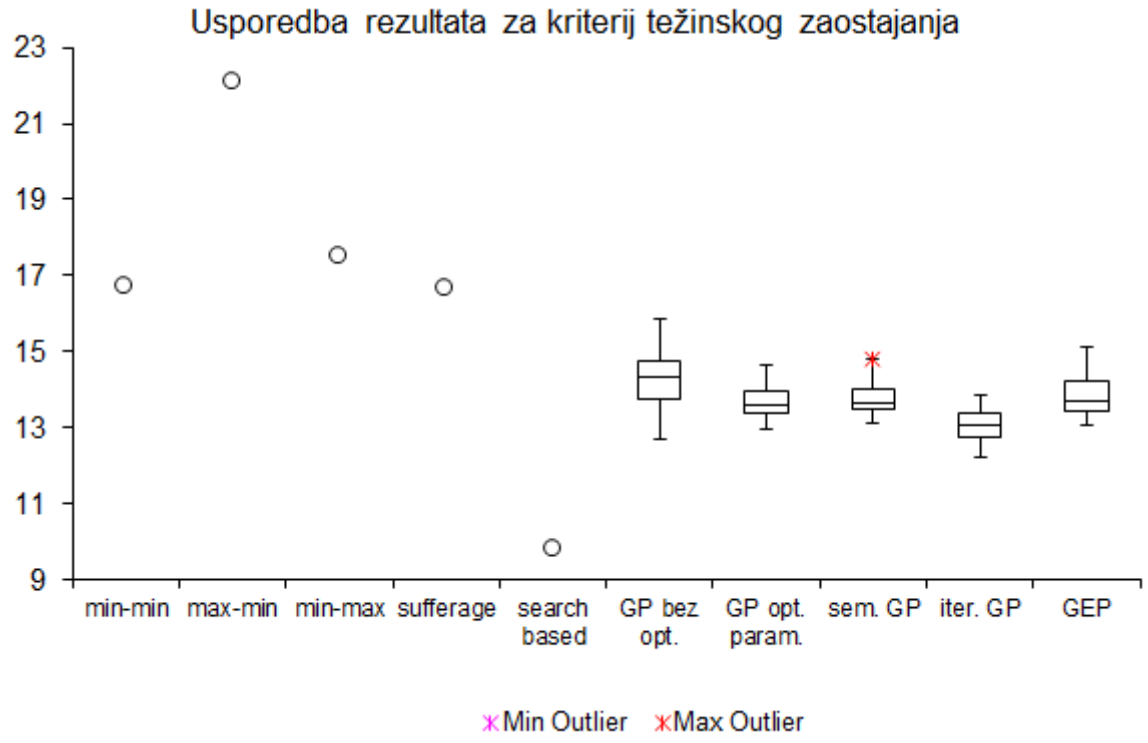
Tablica 6.49: Usporedba rezultata različitih heuristika

Heuristika	Vrijednosti kriterija											
	Twt			Nwt			Fwt			Cmax		
min-min	16,717			7,1437			157,20			38,312		
max-min	22,067			8,1385			195,89			38,835		
min-max	17,492			7,7933			167,30			38,065		
sufferage	16,652			7,1948			160,97			37,927		
search based	9,7800			-			192,30			36,580		
GP prije optimizacija	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Twt opt.	12,710	14,257	15,863	6,3168	6,8430	7,5839	172,57	198,75	285,68	38,750	39,732	41,010
Nwt opt.	13,861	23,869	58,237	6,3779	6,9989	7,7206	159,46	192,79	290,40	38,416	41,047	52,396
Fwt opt.	16,517	17,326	18,871	6,9117	7,1882	7,3601	153,83	155,81	158,58	38,269	38,669	39,369
Cmax opt.	17,098	21,474	27,772	7,2354	7,9053	8,9696	157,95	184,15	217,78	37,960	38,350	38,816
GP s optimiranim parametrima	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Twt opt.	12,962	13,657	14,624	6,1983	6,7119	7,2325	170,54	204,25	353,24	38,689	37,774	42,237
Nwt opt.	13,569	19,103	50,453	6,3842	7,0043	7,9387	158,74	192,56	337,00	38,536	40,306	43,398
Fwt opt.	16,406	17,116	18,670	6,7633	7,1179	7,3686	153,97	155,29	185,56	38,315	38,688	39,158
Cmax opt.	16,570	19,880	26,925	7,2275	7,6990	8,2391	158,41	175,72	203,07	38,018	38,290	38,679
Semnatički GP	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Twt opt.	13,132	13,776	14,808	6,4360	6,7674	7,3812	173,50	188,70	385,41	38,749	39,513	43,153
Nwt opt.	13,365	15,786	34,720	6,4899	6,9257	7,7344	170,99	192,35	348,92	38,727	40,338	44,994
Fwt opt.	16,323	17,256	18,653	6,9541	7,1816	7,4758	153,39	155,87	160,57	38,139	38,629	39,179
Cmax opt.	16,443	19,491	23,827	7,2130	7,6812	8,4339	156,50	172,91	192,26	38,033	38,268	38,929
Iterativni GP	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Twt opt.	12,230	13,054	13,871	6,1760	6,6270	7,5124	168,45	202,31	336,33	38,717	39,724	42,887
Nwt opt.	13,272	20,887	71,042	6,1519	6,7026	7,3131	159,29	195,32	313,60	38,366	40,211	44,041
Fwt opt.	16,360	17,274	19,232	6,9268	7,1632	7,4445	151,56	154,07	156,86	38,218	38,595	39,171
Cmax opt.	16,820	21,325	30,169	7,0655	7,7872	8,4273	155,35	182,60	219,52	37,736	38,034	38,448
GEP	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Twt opt.	13,058	13,856	15,139	6,3117	6,7619	7,3343	162,38	192,39	342,06	38,563	39,499	41,533
Nwt opt.	13,675	17,617	32,895	6,4399	6,9413	7,5526	157,80	177,05	242,70	38,497	39,750	43,914
Fwt opt.	16,441	17,199	18,863	6,9335	7,2047	7,4877	153,53	154,96	158,09	38,322	38,622	39,131
Cmax opt.	16,740	20,270	27,330	7,2318	7,6890	8,3596	157,53	176,94	215,27	37,945	38,217	38,729

6.6.1. Kriterij težinskog zaostajanja

U ovom potpoglavlju detaljno će se usporediti rezultati svih heuristika za kriterij težinskog zaostajanja. Od deterministčkih algoritama najbolju vrijednost za ovaj kriterij je postigla je Sufferage heuristika. Već sam osnovni algoritam genetskog programiranja bez optimizacija postiže rezultate koji su značajno bolji od rezultata postignutih ijednim deterministčkim raspoređivačem. Kako se iz tablice može vidjeti, rezultati koji su postignuti od strane svih optimiranih inačica genetskog programiranja su bolji od osnovne inačice genetskog programiranja po iznosu srednje vrijednosti za navedeni kriterij. Iterativno genetsko programiranje se po iznosu postignutih rezultata posebice ističe, jer je ono postiglo najznačajnija unaprjeđenja. Osim toga, iterativno genetsko programiranje je jedino od svih optimiziranih postupaka genetskog programiranja postiglo bolji rezultat od genetskog programiranja prije optimizacije, po kriteriju minimalne pronađene vrijednosti. Također je potrebno spomenuti da je iterativno genetsko programiranje pronašlo rješenje koje je po iznosu kriterija bolje od onoga prikazanog u tablici (no ono nije postignuto s optimalnim skupom primitiva) i iznosi 11.911. Iz navedenog može se vidjeti da rješenja koja su postignuta ovim optimizacijama još uvijek lošija od rješenja koje je postignuto *search based* metodom, no dobiveni rezultati predstavljaju značajan napredak u odnosu na rješenja koja su dobivena bez provedenih optimizacija. Iz tablice se još dodatno može vidjeti kako se optimiranjem kriterija težinskog zaostajanja ujedno optimira i kriterij težinske zakašnjelosti i da su dobivene vrijednosti težinske zakašnjelosti kompetitivne onima koje su dobivene kada je optimiran i sam kriterij težinske zakašnjelosti.

Slika 6.30 prikazuje *box plot* prikaz rezultata za kriterij težinskog zaostajanja. Iz te slike se može vidjeti kako su isprobane inačice genetskog programiranja u svim segmentima nadmašile deterministčke heuristike. Čak i najlošije pronađeno rješenje još uvijek postiže bolje rezultate od deterministčkih raspoređivača.

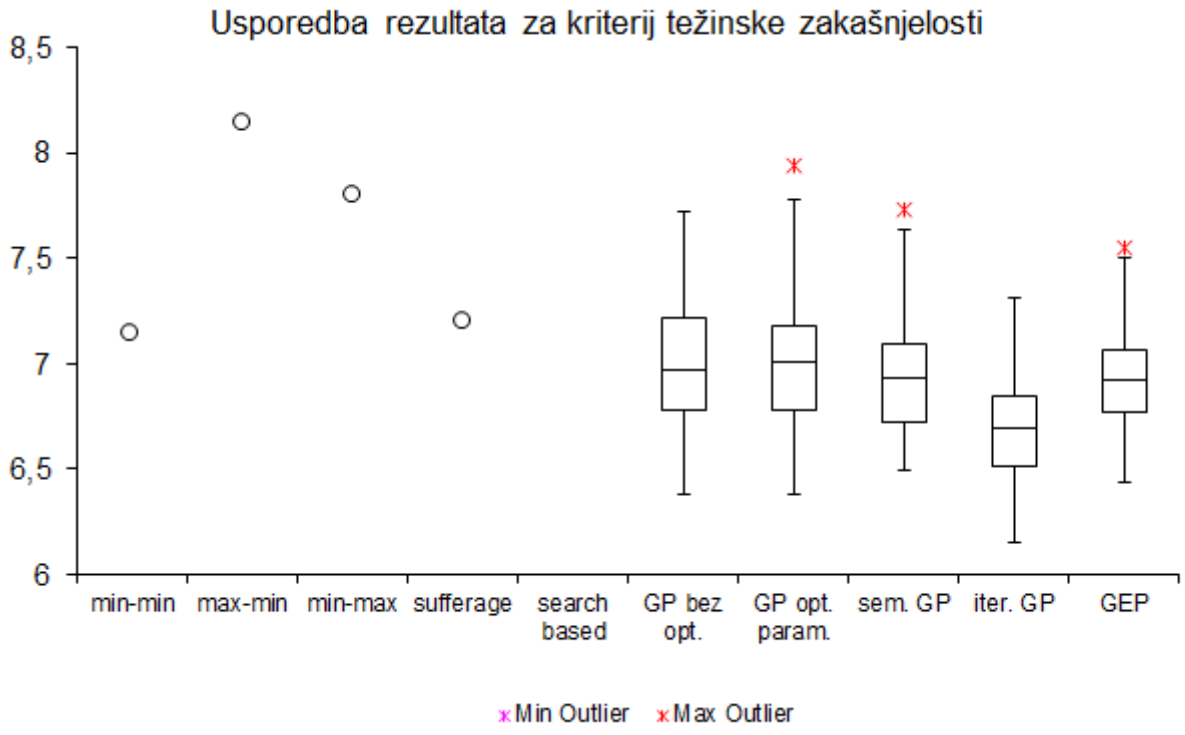


Slika 6.30: *Box plot* za kriterij težinskog zaostajanja

6.6.2. Kriterij težinske zakašnjelosti

U ovom potpoglavlju detaljno će se usporediti rezultati svih heuristika za kriterij težinske zakašnjelosti. Kod deterministčkih raspoređivača, minimalnu vrijednost za ovaj kriterij je postigla Min-min heuristika. Što se rezultata različitih inačica genetskog programiranja tiče, može se uočiti kako su rezultati veoma interesantni. Kao prvo, bitno je uočiti da su za ovaj kriterij veoma često postignuti bolji rezultati kada je bio optimiran kriterij težinskog zaostajanja, što svakako povlači zaključak da ovaj kriterij sam za sebe nije isplativo optimirati. Nadalje, može se uočiti kako je osnovni algoritam genetskog programiranja postigao čak i bolje rezultate po ovom kriteriju od algoritma genetskog programiranja koji je imao optimirane parametre. Ostale isprobane inačice algoritma genetskog programiranja su postigle rezultate koji su po srednjoj vrijednosti bolje od neoptimirane inačice, no nisu uspjele pronaći bolja rješenja po minimalnom iznosu kriterija. Jedina iznimka je iterativno genetsko programiranje koje je po apsolutno svim vrijednostima bolje od svih ostalih isprobanih inačica algoritma genetskog programiranja. No bez obzira na to, može se uočiti kako dobiveno poboljšanje nije veoma veliko. Nažalost, za ovaj kriterij ne postoje rezultati od *search based* postupka s kojim bi se mogla napraviti usporedba.

Slika 6.31 prikazuje *box plot* prikaz rezultata za kriterij težinske zakašnjelosti. Iz ove slike se može vidjeti kako u ovom kriteriju dominacija genetskog programiranja nije toliko velika kao što je bila za prošli kriterij. Također se može uočiti kako su rezultati za sve isprobane inačice genetskog programiranja više manje podjednaki, jedino su rezultati za iterativno genetsko programiranje nešto bolji od ostalih.



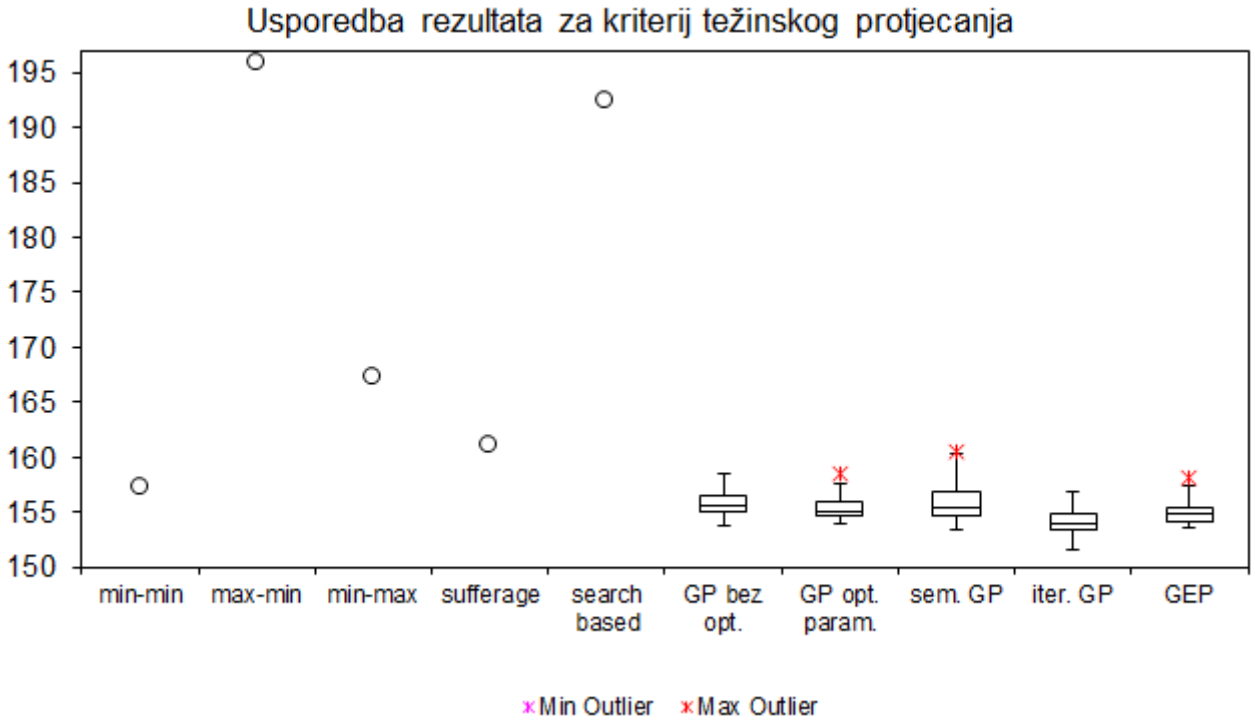
Slika 6.31: *Box plot* za kriterij težinske zakašnjelosti

6.6.3. Kriterij težinskog protjecanja

U ovom potpoglavlju detaljno će se usporediti rezultati svih heuristika za kriterij težinskog protjecanja. Kod deterministčkih raspoređivača, najbolji iznos za ovaj kriterij postigla je Min-min heuristika. Kao što se može vidjeti, osnovna inačica algoritma genetskog programiranja postiže rezultate koji su bolji od rezultata svih heuristika. Gotove sve isprobane optimirane inačice algoritma genetskog programiranja postižu rezultate koji su bolji od osnovne, neoptimirane inačice algoritma. Potrebno je napomenuti kako je GEP za ovaj kriterij postigao dosta dobre rezultate. Najbolje rezultate je opet postigao algoritam iterativnog genetskog programiranja. Mora se napomenuti da je postigao i veoma značajna unaprjeđenja što se rezultata za ovaj kriterij tiče, što posebno iznenađuje jer je iterativno genetsko programiranje bilo prilagođeno za krite-

rij težinskog zaostajanja. Svakako postoji mogućnost da se ostvare još bolji rezultati ako bi se uveli neki primitivi koji bi više odgovarali kriteriju težinskog protjecanja. *Search based* postupak je za ovaj kriterij postigao veoma loše rezultate i nije ni približno usporediv s rezultatima koji su postignuti genetskim programiranjem.

Slika 6.32 prikazuje *box plot* prikaz rezultata za kriterij težinskog protjecanja. Iz slike se može vidjeti kako dobiveni rezultati za ovaj kriterij uvelike variraju o tome koja je heuristika korištena. Od korištenih algoritama genetskih programiranja može se vidjeti kako iterativno genetsko programiranje postiže općenito najbolje rješenje.



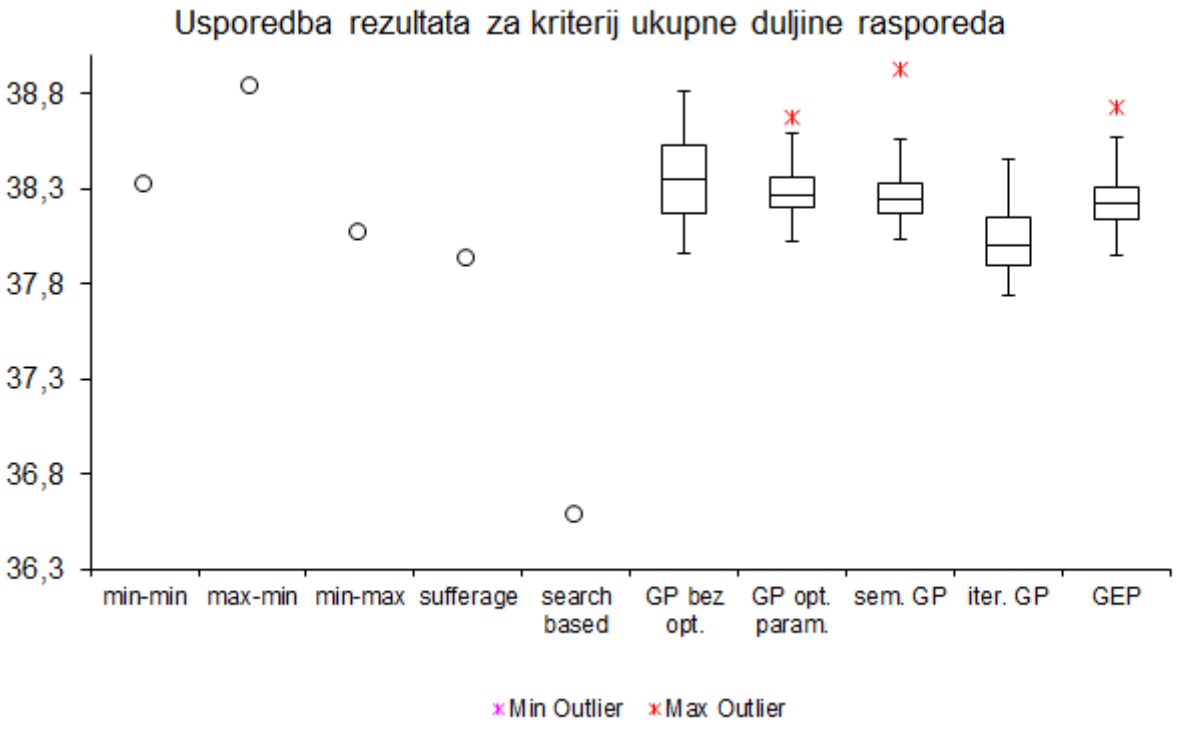
Slika 6.32: *Box plot* za kriterij težinskog protjecanja

6.6.4. Kriterij ukupne duljine rasporeda

U ovom potpoglavlju detaljno će se usporediti rezultati svih heuristika za kriterij maksimalne duljine rasporeda. Za ovaj kriterij najbolje rješenje, od deterministčkih raspoređivača, je postigla Sufferage heuristika. U usporedbi s isprobanim inačicama algoritama genetskog programiranja mora se uočiti kako je po prvi puta jedna deterministička heuristika kompetitivna po rezultatima. Po iznosu srednje vrijednosti sve isprobane inačice algoritma su bile lošije. Jedino je iterativno genetsko programiranje uspjelo pronaći bolje rješenje od Sufferage heuristike, no i dalje je po srednjoj vrijednosti pronađenih rješenja lošije. Svakako postoji opcija, kao i kod prošlog kriterija,

da se iterativno genetsko programiranja prilagodi za optimiranje ovog kriterija i da se time postignu bolji rezultati. Također, može se vidjeti kako je i za ovaj kriterij *search based* postupak postigao najbolje rješenje, no ono se po iznosu ne razlikuje odviše od najboljeg pronađenog rješenja korištenjem iterativnog genetskog programiranja.

Slika 6.31 prikazuje *box plot* prikaz rezultata za kriterij ukupne duljine rasporeda. Iz slike se može vidjeti kako su po ovom kriteriju deterministički raspoređivači dosta izjednačeni s algoritmima genetskog programiranja. Sufferage heuristika je postigla rezultat koji je bolji od svih rezultata postignutih od algoritama genetskog programiranja, jedino je iterativno genetsko programiranje uspjelo pronaći ponešto bolje rješenje. *Search based* postupak je i u ovom kriteriju postigao najbolje rješenje.



Slika 6.33: *Box plot* za kriterij ukupne duljine rasporeda

7. Zaključak

U ovom radu proučeno je i isprobano nekoliko postupaka optimizacije raspoređivanja na nesrodnim strojevima s ciljem poboljšanja kvalitete dobivenih rješenja. Od isprobanih postupaka optimizacije pojedini su se pokazali kao veoma uspješni, dok su se druge strane određeni optimizacijski postupci rezultirali rješenjima lošije kvalitete. U nastavku su prikazani glavni zaključci o svakoj isprobanoj optimizaciji.

Optimizacijom parametara algoritma nastojalo se pronaći parametre uz koje će algoritam postizati što bolje rezultate. Iako u većini slučajeva pri određivanju parametara genetskog programiranja nije bilo većih iznenađenja, ipak se mogu izdvojiti dvije nepredviđene pojave. Prva pojava jest da algoritam genetskog programiranja postiže bolja rješenja za manje dubine stabla. Iako algoritam može lakše naučiti manja stabla, ipak je bilo očekivano da će nešto veća stabla zbog svoje ekspresivnosti postići ponešto bolje rezultate. Druga neočekivana pojava jest bila ta se postižu bolja rješenja ukoliko se ne koristi križanje s jednom točkom. Iako su isprobane i druge kombinacije križanja i mutacije, one se nisu pokazale toliko efikasnim. Razlog zašto baš to križanje loše utječe na izgradnju rješenja nije poznato.

Optimizacijom dodavanja novih funkcijskih čvorova nastojalo se povećati ekspresivnost algoritma genetskog programiranja i na taj način dobiti bolja rješenja. Nažalost, ovaj postupak se i nije pokazao previše efikasnim jer se uključivanjem novih funkcijskih čvorova nisu postigli bolji rezultati, već naprotiv, rezultati su veoma često bili i dosta lošiji. Jedinu iznimku predstavlja čvor koji računa srednju vrijednost između svoje djece. Iako korištenjem navedenog čvora nisu postignuti ukupno bolji rezultati, ipak su za navedeni postignuti najbolji rezultati od svih novododanih čvorova.

Optimizacijom semantičkim genetskim programiranjem nastojalo se izraditi rješenja koja će biti semantički ispravna. Ova metoda je u većini slučajeva generirala rješenja koja su se manje rasipala (odnosno rješenja su imala manju devijaciju) od klasičnog genetskog programiranja. Iako je prosjek dobrota pronađenih rješenja kod semantičkog genetskog programiranja bio manji, ipak je klasično genetsko programiranje, u većini slučajeva, uspjelo pronaći rješenje s minimalnim iznosom dobrote.

Iterativna pravila raspoređivanja ugrađena su u postupak raspoređivanja zasnovanog na prilagodljivim pravilima s ciljem poboljšanja kvalitete generiranih rješenja. Ovaj postupak, koji koristi informacije iz prethodno izrađenih rasporeda kako bi u idućoj iteraciji izgradio kvalitetnije rasporede, postigao je najbolje rezultate od svih optimizacijskih postupaka koji su bili isprobani. Problem kod ovog postupka je taj što je potrebno odrediti nove ulaze u prioritetnu funkciju koji predstavljaju određene veličine iz prethodno generiranih rasporeda. Kvaliteta dobivenih rezultata ovisi o odabranim ulaznim veličinama, iz čega proizlazi potreba za pronalaskom što kvalitetnijeg skupa ulaznih veličina.

GEP je također implementiran i isproban kako bi se ocijenio utjecaj drugačije reprezentacije jedinki na kvalitetu rješenja, i na taj način možda dobila bolja rješenja. Kroz eksperimente je pokazano kako je GEP za određene kriterije postigao veoma dobre rezultate te da postoji potencijal za nastavak istraživanja primjene ovog pristupa za raspoređivanje na nesrodnim strojevima.

Kao što je iz cjelokupnog rada vidljivo, raspoređivanje na nesrodnim strojevima je veoma težak problem. Iako je kroz rad isproban širok niz različitih optimizacija koje djeluju na različite segmente algoritma genetskog programiranja, nisu se sve optimizacije pokazale uspješnima. U konačnici su postignuta unaprjeđenja u kvaliteti dobivenih rješenja u odnosu na osnovnu inačicu bez optimizacija. Bez obzira na postignute rezultate, svakako postoji još prostor za isprobavanje drugih optimizacijskih tehnika ili čak rafiniranje tehnika opisanih u ovom radu kako bi se postigla još bolja rješenja.

LITERATURA

[1] P. D'haeseleer. Context Preserving Crossover in Genetic Programming. *6th European Conference, EuroGP 2003*, stranice 256–261, 1994.

[2] P. G. Espeju, V. Sebastian, i H. Francisco. A Survey on the Application of Genetic Programming to Classification. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS*, stranice 121 – 144, 2010.

[3] K. Etminani i M. Naghibzadeh. A Min-Min Max-Min Selective Algorihtm for Grid Task Scheduling. *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, stranice 1 – 7, 2007.

[4] C. Ferreira. Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. *Complex Systems* , stranice 87–129, 2001.

[5] H. Izakian i A. Abraham. Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments. *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, stranice 8–12, 2009.

[6] H. Jabeen i A. R. Baig. Review of Classification Using Genetic Programming. *International Journal of Engineering Science and Technology*, stranice 94 – 103, 2010.

[7] D. Jakobović. *RASPOREĐIVANJE ZASNOVANO NA PRILAGODLJIVIM PRAVILIMA*. Doktorska disertacija, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2005.

[8] D. Jakobović i L. Budin. Dynamic Scheduling with Genetic Programming. *9th European Conference, EuroGP 2006*, 3905:73–84, 2006.

[9] D. Jakobović i K. Marasović. Evolving priority scheduling heuristics with genetic programming. *Applied Soft Computing*, stranica 2781–2789, 2012.

[10] D. Jakobović, I. Grudenić, i L. Jelenković. Genetic Programming Heuristics for Scheduling of Dynamic Unrelated Machines.

[11] G.K. Kamalan i V. Murali Bhaskaran. A new Heuristic approach:Min-mean Algorithm For Scheduling Meta-Tasks on Heterogenous Computing Systems. *IJCSNS International Journal of Computer Science and Network Security*, 10:24–31, 2010.

[12] G.K. Kamalan i V. Murali Bhaskaran. An Improved Min-Mean Heuristic Scheduling Algorithm for Mapping Independent Tasks on Heterogenous Computing Environment. *INTERNATIONAL JOURNAL OF COMPUTATIONAL COGNITION*, 8:85–91, 2010.

[13] M. Keijzer. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. *6th European Conference, EuroGP 2003*, 2610:70–82, 2003.

[14] M. Keijzer i V. Babović. Dimensionally Aware Genetic Programming. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2:1069–1076, 1999.

[15] W. B. Langdon. Size Fair and Homologous Tree Crossovers for Tree Genetic Programming. *Genetic Programming and Evolvable Machines*, stranice 95–119, 2000.

[16] X. Li, C. Zhou, i P. C. Nelson. Prefix Gene Expression Programming. *Genetic and Evolutionary Computation Conference (GECCO)*, 2005.

[17] S. Nguyen, M. Zhang, i K. C. Johnston, M. Tan. Learning Iterative Dispatching Rules for Job Shop Scheduling with Genetic Programming. *The International Journal of Advanced Manufacturing Technology*, stranice 85–100, 2013.

[18] S. Nguyen, M. Zhang, M. Johnston, i K. C. Tan. A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem. *Evolutionary Computation, IEEE Transactions on*, stranice 621 – 639, 2013.

[19] L. Nie, X. Shao, L. Gao, i W. Li. Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. *The International Journal of Advanced Manufacturing Technology*, stranice 729–747, 2010.

[20] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, i Sholz-Reiter B. Evolutionary generationofdispatchingrulesetsforcomplexdynamic schedulingproblems. *International Journal of Production Economics*, stranica 67–77, 2013.

[21] R. Poli, W.B. Langdon, N.F. McPhee, i Koza J.R. *A Field Guide to Genetic Programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[22] M. Čupić, B. Dalbelo Bašić, i M. Golub. *Neizrazito, evolucijsko i neuroračunarstvo*. -, 2013.

Optimizacija raspoređivanja u okruženju nesrodnih strojeva

Sažetak

U ovom radu opisan je problem raspoređivanja u okolini nesrodnih strojeva. Opisan je algoritam genetskog programiranja kao i raspoređvanje zasnovano na prilagodljivim pravilima koje zapravo predstavlja primjenu genetskog programiranja na izradu rasporeda. Opisan je niz optimizacija (optimizacija parametara, dodavanje novih funkcijskih čvorova, semantičko genetsko programiranje, iterativna pravila raspoređivanje i GEP) koje su implementirane i isprobane kako bi se postigla kvalitetnije izrada rasporeda od strane genetskog programiranja. Prikazani su i komentirani rezultati koji su dobiveni za svaku pojedinu optimizacijsku tehniku.

Ključne riječi: raspoređivanje na nesrodnim strojevima, genetsko programiranje, semantičko genetsko programiranje, raspoređivanje zasnovano na prilagodljivim pravilima, iterativna pravila raspoređivanja, GEP

Optimization of scheduling for unrelated machines

Abstract

In this thesis the problem of scheduling in the unrelated machines environment was described. The genetic programming algorithm as well as scheduling based on adaptive rules, which in itself is nothing but the application of genetic programming to generating schedules, were also described. A number of optimization techniques (parameter optimization, adding new functional nodes, semantic genetic programming, iterative dispatching rules and gene expression programming) which were implemented and tested in order to achieve improvements in generating the schedules, were also described. Finally, for each of the mentioned optimization techniques the results of it's usage are shown and commented upon.

Keywords: scheduling on unrelated machines, genetic programming, semantic genetic programming, scheduling based on adaptive rules, iterative dispatching rules, GEP