SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br.46

SEMANTIČKO GENETSKO PROGRAMIRANJE

Neven Balenović

Zagreb, lipanj 2010.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br.46

SEMANTIČKO GENETSKO PROGRAMIRANJE

Neven Balenović

Zagreb, lipanj 2010.

Zahvala

Ovaj rad izrađen je na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave pod vodstvom mentora Doc. dr. sc. Domagoja Jakobovića.

Zahvaljujem mentoru na pomoći oko izrade ovog rada, kao i na stalnoj dostupnosti, strpljenju i vremenu koje je izdvojio tijekom cijelog semestra.

Sadržaj

1 UVOD

Za velik broj problema nije poznat egzaktan algoritam rješavanja, a problemi su presloženi za rješavanje iscrpnom pretragom. Kod takvih problema nije cilj pronaći optimalno rješenje, već samo rješenje koje je „dovoljno dobro“. S tim ciljem razvijaju se *heuristički algoritmi*.

Genetsko programiranje spada u skupinu postupaka koji se zajednički nazivaju evolucijskim računanjem (engl. *evolutionary computation*, EC), među koje se još ubrajaju genetski algoritmi, evolucijske strategije itd. Osnovna ideja za genetsko programiranje uzeta je iz prirode, odnosno procesa evolucije kojim bolje jedinke preživljavaju i stvaraju potomstvo, dok lošije izumiru. Postupak održava skup rješenja (populaciju) nad kojima primjenjuje genetske operatore (selekcija, križanje i sl.) s ciljem pronalaženja sve boljih rješenja. Posebnost genetskog programiranja je u tome što je svako rješenje predstavljeno strukturom promjenjive veličine i oblika, što je idealna postavka za predstavljanje računalnih programa. Na kraju postupka, najbolje rješenje (ili više njih) predstavlja rješenje izvornog problema.

U ovom radu genetsko programiranje primijenjeno je na problem raspoređivanja u proizvoljnoj obradi (engl. *job shop problem*). Raspoređivanje je proces koji se bavi dodjelom ograničenih sredstava skupu aktivnosti s ciljem optimiranja jednog ili više mjerila vrednovanja. Sredstva mogu biti strojevi u proizvodnom pogonu, procesor računala, piste na aerodromu itd. Aktivnosti mogu biti razne operacije u proizvodnji, izvođenja računalnih programa, slijetanje ili uzlijetanje aviona i druge. Rješenje problema raspoređivanja je raspored koji definira kada će se i na kojem sredstvu odvijati pojedina aktivnost.

U *drugom poglavlju* rada opisan je problem raspoređivanja poslova. Dana je definicija i primjer raspoređivanja poslova u proizvoljnoj obradi i navedena su najčešće korištena pravila raspoređivanja te mjerila vrednovanja rasporeda. U *trećem poglavlju* navode se osnovni pojmovi vezani uz genetsko programiranje i opisuje način rada

genetskog programa. *Četvrto poglavlje* opisuje ulogu semantike u genetskom programu te daje primjere semantičke ispravnosti jedinki. U *petom poglavlju* opisana je implementacija problema raspoređivanja u proizvoljnoj obradi pomoću genestkog programiranja. *Šesto poglavlje* prikazuje rezultate dobivene testiranjem programa nad skupom generiranih primjera te primjera preuzetih iz literature.

2 PROBLEM RASPOREĐIVANJA U PROIZVOLJNOJ OBRADI

U ovom poglavlju opisan je problem raspoređivanja poslova (operacija) na različita sredstva. Ukratko je opisan općeniti problem raspoređivanja te svojstva poslova, a posebna je pažnja posvećena problemu raspoređivanja u proizvoljnoj obradi. Za potonji su navedena najčešće korištena pravila raspoređivanja te načini vrednovanja dobivenog rasporeda.

2.1 Opis problema raspoređivanja

Raspoređivanje je u širem smislu postupak izrade bilo kakvog rasporeda. U različitim okruženjima postupak izrade rasporeda može se podijeliti na više stupnjeva, ovisno o algoritmu i problemu. U problemima koji uključuju samo jedno sredstvo, raspoređivanje se sastoji od određivanja redoslijeda aktivnosti na tom sredstvu. Ako se koristi više odvojenih sredstava, prije izrade redoslijeda potrebno je odrediti koje aktivnosti će se odvijati na kojem sredstvu. U ovom radu promatrat će se raspoređivanje na više odvojenih sredstava. Postupak dodjeljivanja aktivnosti sredstvima nazivat će se *pridruživanjem* (engl. *matching*), a postupak izrade redoslijeda na pojedinom sredstvu *uređivanjem* (engl. *sequencing*).

Raspoređivanje je postupak koji se odvija u vrlo širokom skupu uvjeta, no uvijek uključuje obavljanje radnji (aktivnosti) koje zauzimaju određena sredstva u nekom vremenskom periodu. Radnje koje se obavljaju mogu se nazivati poslovima ili projektima, a mogu se sastojati i od manjih cjelina kao što su zadaci ili operacije. Svaka aktivnost zahtijeva određeno sredstvo u određenoj količini vremena. Sredstva također mogu imati elementarne dijelove koji se nazivaju strojevi, procesori ili ćelije, itd. Problemi raspoređivanja se često usložnjavaju raznim ograničenjima (neke aktivnosti prethode drugima, trajanje postavljanja, itd.), ali takvi primjeri ne će u ovom radu biti razmatrani.

2.2 Svojstva poslova

Objekt čije se izvođenje želi omogućiti postupkom raspoređivanja najčešće se naziva posao (engl. *job*) ili zadatak (engl. *task*). U nekim okruženjima raspoređivanja posao se može sastojati od više aktivnosti ili operacija, dok se u nekim drugim uvjetima (npr. kod raspoređivanja na jednom stroju) smatra da se posao sastoji samo od jedne nedjeljive aktivnosti.

Skup svih poslova obično se označava sa J, a pojedini posao sa J_j. Isto tako, skup zadataka se označava sa T, a pojedini zadatak sa T_j. Svakom od elemenata iz navedenih skupova pridružena su neka od svojstava opisanih u nastavku. Pri opisu svojstava i okoline raspoređivanja, pod pojmom „*vrijeme*“ podrazumijevat će se neki jedinstveni trenutak u vremenskom slijedu, dok će se pod pojmom „*trajanje*“ podrazumijevati određena količina vremena, odnosno razlika između dva jedinstvena trenutka.

Trajanje izvođenja (p_j) – svaka nedjeljiva aktivnost zahtijeva određenu količinu vremena koja se naziva trajanje izvođenja (engl. *processing time*) i označava kao p_j, gdje je j indeks dotične aktivnosti (posla). Ako se radi o okruženju s više strojeva, tada se trajanje izvođenja posla J_j na stroju i označava s $\boldsymbol{p_{ij}}$.

Vrijeme pripravnosti (r_j) – vrijeme pripravnosti (engl. *ready time, release time*) ili vrijeme pojavljivanja je trenutak u kojem posao dolazi u sustav, odnosno u kojem postaje raspoloživ za izvođenje. Nijedan posao ne može se izvoditi prije svog vremena pripravnosti. S druge strane, posao može čekati početak izvođenja neodređeno dugo nakon svog vremena pripravnosti, bilo zbog zauzetosti sredstva, bilo zbog nekog drugog neispunjenog uvjeta.

Vrijeme željenog završetka (d_j) – vrijeme željenog završetka (engl. *due date*) nekog posla je trenutak do kojega se očekuje da posao treba završiti s izvođenjem. Posao može završiti i nakon tog trenutka, ali se u tom slučaju stvara određeni trošak.

Težina (w_j) – težina (engl. *weight*) odnosno razina prvenstva nekog posla određuje prioritet dotičnog zadatka u sustavu. Težina se najčešće koristi u određivanju troškova pri

ocjenjivanju rasporeda, gdje ona predstavlja neku stvarnu mjeru kvalitete rasporeda (npr. novčani trošak posla po danu kašnjenja). U stvarnim uvjetima, gdje se ocjena rasporeda obavlja na temelju više kriterija, poslu se može pridružiti i više od jedne težinske vrijednosti.

2.3 Raspoređivanje u proizvoljnoj obradi

Okruženje proizvoljne obrade predstavljeno je poslovima od kojih se svaki sastoji od zadanog broja operacija. Operacije se izvode predefiniranim redoslijedom na točno određenim strojevima. U najopćenitijem obliku proizvoljne obrade, svaki posao može imati proizvoljan broj operacija (različit od broja strojeva), a različite se operacije u slijedu mogu izvoditi i na istim strojevima (tzv. višestruka proizvoljna obrada, engl. *reentrant job shop*). U literaturi se ipak najčešće javlja inačica problema u kojoj je broj operacija svakog posla jednak broju strojeva, a svaki posao se izvodi na svakom stroju točno jednom. Radi lakše usporedbe učinkovitosti s primjerima iz literature, u ovom se radu rješava inačica u kojoj je broj operacija svakog posla jednak broju strojeva.

2.3.1 Formalna definicija i primjer

Problem raspoređivanja u proizvoljnoj okolini veličine *n x m* (*n x m job shop problem*) je sljedeći:

- Skup od *n* poslova $\{J_j\}_{1\leq j\leq n}$ potrebno je izvršiti na skupu od *m* strojeva $\{M_r\}_{1\leq r\leq m}$. Obrada posla J_j na stroju M_r predstavlja operaciju $O_{j,i,r}$ pri čemu $i \in \{1, ..., m\}$ određuje broj operacije u skupu operacija pojedinog posla, a *j* i *r* predstavljaju indeks posla, odnosno stroja na kojem se posao obrađuje.

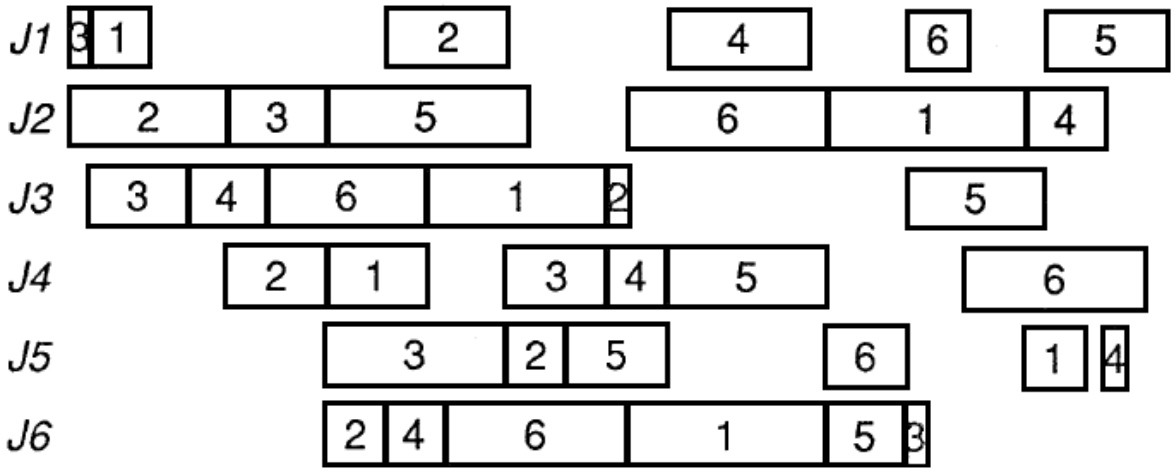
U tablici 2.1 dan je primjer jednostavnog problema raspoređivanja u proizvoljnoj obradi sa šest poslova i šest strojeva (6x6 *job shop problem*).

Tablica 2.1 – problem raspoređivanja u proizvoljnoj obradi sa šest strojeva i poslova

Posao	Indeks operacije(vrijeme izvršavanja operacije)					
1	3(1)	1(3)	2(6)	4(7)	6(3)	5(6)
2	2(8)	3(5)	5(10)	6(10)	1(10)	4(4)
3	3(5)	4(4)	6(8)	1(9)	2(1)	5(7)
4	2(5)	1(5)	3(5)	4(3)	5(8)	6(9)
5	3(9)	2(3)	5(5)	6(4)	1(3)	4(1)
6	2(3)	4(3)	6(9)	1(10)	5(4)	3(1)

Svaki element E_{ij} tablice sadrži podatak o indeksu stroja na kojem se izvršava *j*-ta operacija posla *i* te podatak o trajanju te operacije (prikazan u zagradi). Iz tablice je vidljivo da se svaki posao izvodi na svakom stroju točno jednom, odnosno problem je ekvivalentan problemima kakvi će se rješavati u ovom radu.

Optimalno rješenje zadanog problema (tablica 2.1) prikazano je *Ganttovim dijagramom* na slici 2.1. Za svaki posao prikazani su indeksi strojeva na kojima se operacija obavlja (brojevi na slici) te vrijeme trajanja dotične operacije (grafički – veličinom pravokutnika). Takav prikaz rješenja nije pogodan za zapis u računalu, ali je vizualno privlačan i jednostavan.



Slika 2.1 – prikaz rješenja Ganttovim dijagramom

Jednostavniji zapis rješenja dan je u tablici 2.2. Tablica prikazuje vremena završetaka pojedinih operacija svakog posla, pri čemu je za evaluaciju kvalitete rasporeda najčešće bitan samo zadnji stupac tablice, odnosno vrijeme završetka cijelog posla.

Tablica 2.2 – vremena završetaka pojedinih operacija

Posao	Vrijeme završetka operacije					
1	1	4	22	37	45	55
2	8	13	23	38	48	52
3	6	10	18	27	28	49
4	13	18	27	30	38	54
5	22	25	30	42	51	53
6	16	19	28	38	42	43

2.3.2 Pravila raspoređivanja

U uvjetima raspoređivanja u stvarnom vremenu ili u okruženju u kojem je potrebno brzo reagirati na promjene stanja sustava, najčešće se za raspoređivanje koristi algoritam koji relativno brzo može doći do potrebnog rješenja. Pri tome potrebno rješenje ne mora biti cjelokupni raspored od početka do (eventualnog) završetka rada sustava, već samo opis sljedećeg stanja sustava, po modelu dinamičkog raspoređivanja.

Rješavanje problema u opisanim uvjetima uglavnom se rješava primjenom pravila raspoređivanja. **Pravilo raspoređivanja** u užem smislu podrazumijeva samo funkciju koja, koristeći određene parametre sustava, definira metriku odnosno prioritet elemenata sustava. Rješenje koje se dobiva primjenom pravila raspoređivanja definira pridruživanje koje je potrebno obaviti prilikom sljedeće promjene stanja sustava.

Svim pravilima raspoređivanja na raspolaganju su informacije o poslovima, njihovim operacijama i stanju na strojevima, a prilikom definiranja funkcija prioriteta koriste se sljedeće oznake:

- $\boldsymbol{p_{ij}}$ – trajanje jedne operacije posla *j* na stroju *i* (budući da svaki posao ima broj operacija jednak broju strojeva, a informacija o redoslijedu operacija se ne koristi u pravilu raspoređivanja, nije potrebno navoditi redni broj operacije)
- $\boldsymbol{w_j}$ – težinska vrijednost posla
- $\boldsymbol{d_j}$ – željeno vrijeme završetka posla
- $\boldsymbol{twk_j}$ – (engl. *total work*) ukupno trajanje svih operacija nekog posla
- $\boldsymbol{twkr_j}$ – (engl. *total work remaining*) ukupno trajanje svih preostalih operacija nekog posla (operacija čije izvođenje još nije započelo)

U nastavku su navedena pravila raspoređivanja koja se koriste za ocjenu učinkovitosti genetskog programiranja u ovom radu s pripadajućom funkcijom prioriteta kojom se odabire sljedeća operacija. Pritom je najbolja mjera jednaka najvećoj postignutoj vrijednosti funkcije prioriteta.

- **WSPT** pravilo:

$$\pi_j = \frac{w_j}{p_{ij}} \tag{2.1}$$

- **WSPT/TWKR** pravilo:

$$\pi_j = \frac{w_j \cdot twkr_j}{p_{ij}} \tag{2.2}$$

- **WTWKR (engl. *weighted total work remaining*)** pravilo:

$$\pi_j = \frac{w_j}{twkr_j} \tag{2.3}$$

- **SLACK/TWKR (engl. *slack per remaining process time*)** pravilo:

$$\pi_j = \frac{w_j \cdot twkr_j}{d_j - twkr_j} \tag{2.4}$$

- **EDD** pravilo:

$$\pi_j = \frac{1}{d_j} \tag{2.5}$$

- **COVERT (engl. *cost over time*)** pravilo:

$$\pi_j = \frac{w_j}{p_{ij}}\left[1 - \frac{\left(d_j - ELT_{ij} - time\right)^+}{h \cdot ELT_{ij}}\right]^+ \tag{2.6}$$

gdje je *h* parametar heuristike, a ELT_{ij} (engl. *estimated lead time*) označava procjenu količine vremena koju će posao *j* provesti u sustavu nakon završetka operacije na trenutnom stroju (*i*). Ova se veličina može procijeniti na više načina, no najčešće se postavlja na umnožak zadane konstante s preostalim trajanjem obrade, tj. $ELT_{ij} = k \cdot twkr_j$. Uobičajena vrijednost za *k* koja se i ovdje koristi jednaka je 3, a za parametar *h* autori sugeriraju vrijednost 0.5 (Morton 1993).

- **RM** pravilo za proizvoljnu obradu:

$$\pi_j = \frac{w_j}{p_{ij}} \cdot exp\left[\frac{\left(d_j - ELT_{ij} - time\right)^+}{h \cdot \overline{p}_i}\right] \tag{2.7}$$

gdje je $\overline{p_i}$ prosječno trajanje svih operacija na promatranom stroju, a *h* je empirijski parametar heuristike. ELT_{ij} se računa na jednak način kao i za *COVERT* pravilo.

2.3.3 Vrednovanje rasporeda

Vrednovanje rasporeda moguće je obaviti po više kriterija koji su često vrlo složeni i međusobno proturječni (Jones i Rabelo 1998). Kriteriji vrednovanja rasporeda u velikoj mjeri utječu na izbor odgovarajućeg algoritma raspoređivanja. Ocjena rasporeda moguća je nakon izvođenja rasporeda i nakon prikupljanja izlaznih veličina u sustavu (svi do sada opisani parametri predstavljaju zapravo ulazne veličine problema). Slijedeći konvenciju iz (Leung 2004), izlazne veličine sustava opisane su velikim slovima.

- **Vrijeme završetka C_j** (engl. *completion time*) – trenutak u kojemu aktivnost *j* završava izvođenje.

- **Protjecanje F_j** (engl. *flowtime*) – količina vremena koju je neka aktivnost provela u sustavu:

$$F_j = C_j - r_j \tag{2.8}$$

- **Kašnjenje L_j** (engl. *lateness*) – razlika (pozitivna ili negativna) između vremena završetka i vremena željenog završetka:

$$L_j = C_j - d_j \tag{2.9}$$

- **Zaostajanje T_j** (engl. *tardiness*) – pozitivni iznos kašnjenja neke aktivnosti; ako je kašnjenje negativno, zaostajanje je jednako nuli:

$$T_j = max\{0, L_j\} \tag{2.10}$$

- **Zakašnjelost U_j** – označava je li neka aktivnost prekoračila željeno vrijeme završetka:

$$U_j = \begin{cases} 1, & T_j > 0 \\ 0, & T_j = 0 \end{cases} \tag{2.11}$$

Treba napomenuti kako se u literaturi i za označavanje zadatka koristi simbol T_j, no u ovom će radu taj simbol označavati isključivo zaostajanje neke aktivnosti. Navedene veličine koriste se za definiranje većine mjerila vrednovanja rasporeda. Kriteriji vrednovanja koji se koriste u ovom radu navedeni su u nastavku.

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

$$C_{max} = max\{C_j\} \tag{2.12}$$

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

$$F_w = \sum_j w_j \cdot F_j \tag{2.13}$$

- **Težinsko zaostanjanje T_w** (engl. *weighted tardiness*) – jednako je težinskoj sumi zaostajanja svih poslova:

$$T_w = \sum_j w_j \cdot T_j \tag{2.14}$$

- **Težinski zbroj zaostalih poslova ili težinska zakašnjelost U_w** (engl. *weighted number of tardy jobs*) – definira se kao težinska suma svih zaostalih poslova:

$$U_w = \sum_j w_j \cdot U_j \tag{2.15}$$

3 GENETSKO PROGRAMIRANJE

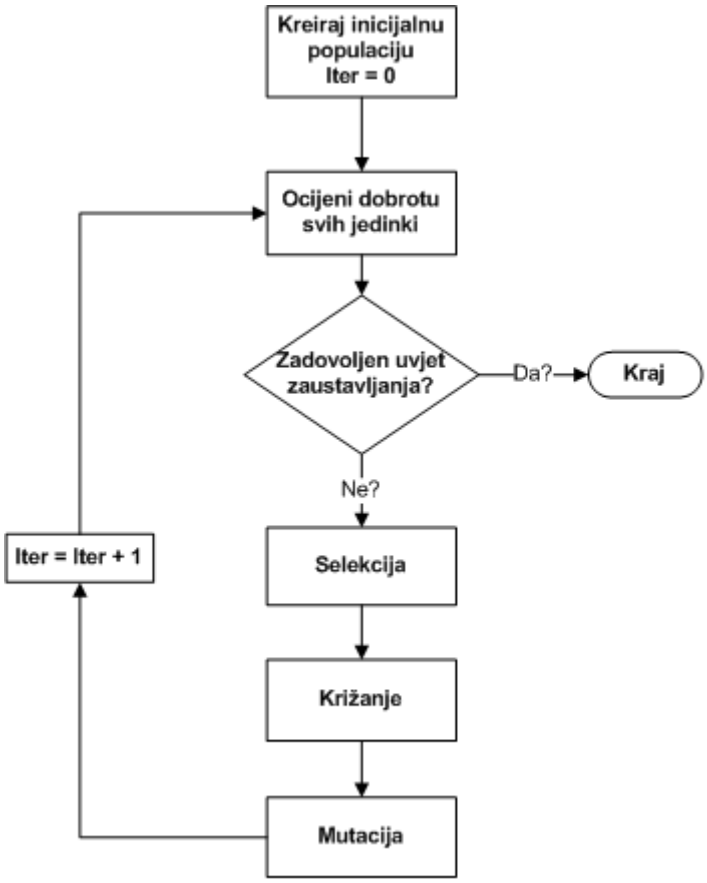
U prirodi snažnije i dominantnije jedinke nadvladavaju slabije te na taj način preživljavaju, prenoseći dobra svojstva svojeg DNA na potomke. Koristeći mehanizme evolucije (prirodna selekcija, križanje te mutacija), priroda stvara jače i prilagodljivije jedinke, dok lošije jedinke (i njihov DNA) izumiru. Jedinke koje su po nekom svojstvu dominantnije od ostalih daljnjim će križanjem proizvesti nove, sa svojstvima bližim „savršenstvu“.

Sličan postupak evolucije postoji i u modernom računarstvu, pri čemu je svaka jedinka jedan računalni program koji, bolje ili lošije, rješava zadani problem. Problemi koji se rješavaju genetskim programiranjem najčešće su složeni problemi za koje nije poznat egzaktan algoritam rješavanja, pa se genetsko programiranje koristi kao heuristika za dobivanje „dovoljno dobrog“ rješenja.

Na slici 3.1 prikazan je osnovni princip optimizacije rješenja (programa) pomoću genetskog programiranja. Najprije se stvara početni skup jedinki (populacija) koji najčešće vrlo loše rješava zadani problem. Nakon toga započinje iterativni postupak određivanja dobrota postojećih jedinki te stvaranja novih postupcima križanja i mutacije. Program završava nakon što se zadovolji uvjet zaustavljanja genetskog programa.

3.1 Prikaz jedinki

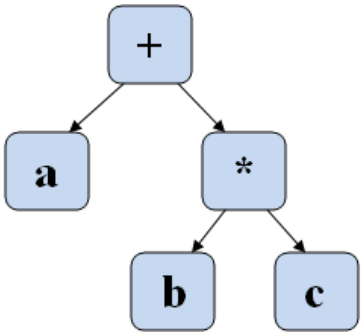
Odabir dobrog načina prikaza jedinki je jedan od najvažnijih preduvjeta da bi neki genetski program pronašao „dovoljno dobro“ rješenje. Jedinke se najčešće prikazuju u obliku stabla, pri čemu svaki čvor predstavlja neko svojstvo jedinke ili izvršava neku funkciju. Svaka jedinka je računalni program koji (bolje ili lošije) rješava zadani problem pa genetski zapis jedinke (čvorovi stabla) mora biti takav da prilikom križanja i mutacije ne dolazi do nemogućih rješenja. Drugim riječima, unutar „definicije jedinke“ moraju biti samo najvažnija svojstva.



Slika 3.1 - Optimizacija pomoću genetskog programiranja

3.2 Generiranje početne populacije

Početna populacija (nulta generacija) stvara se na temelju slučajnog odabira. Jedinke u početnoj populaciji najčešće vrlo loše rješavaju zadani problem pa se može reći da je njihova uloga generiranje genetskog materijala pomoću kojeg će se kasnije stvoriti bolje jedinke. Čvorovi stabla (svojstva jedinke) mogu se podijeliti na završne i nezavršne pri čemu samo nezavršni čvorovi mogu biti listovi. Ako smo u nekom trenutku generirali nezavršni čvor, nakon njega moramo generirati dovoljan broj završnih čvorova da popunimo stablo. Primjer jednostavne jedinke koja izvodi funkciju $f = a + b \cdot c$ prikazan je na slici 3.2.



Slika 3.2 – Jedinka koja izvršava funkciju f = a + b*c

Jedinka je dobivena tako da su redom generirani čvorovi „+“, „a“, „*“, „b“ i „c“ te zatim dodani u stablo. Čvorovi „+“ i „*“ su nezavršni čvorovi i zahtijevaju dodatne informacije (završne čvorove) za izvođenje. Završni čvorovi u danom primjeru su „a“, „b“ i „c“. Moglo bi se reći da su završni čvorovi genetski materijal jedinke, dok nezavršni čvorovi opisuju kako se taj genetski materijal odabire i kombinira. Prikaz jedinke u obliku stabla je uobičajen u genetskom programiranju jer se putem njega mogu najlakše izvesti operatori križanja i mutacije.

Izgradnja početne populacije se može izvesti na tri načina:

- *grow* metodom
- *full* metodom
- *ramped half-and-half* metodom

Grow metoda – počevši od korijena, slučajno se odabire vrsta čvora koji će se dodati u stablo. Pritom je moguće birati završne ili nezavršne čvorove. Ako je odabran završni čvor, sustav odabire bilo koji završni čvor i dodaje ga u stablo, inače dodaje neki nezavršni čvor. Postupak se rekurzivno ponavlja dok svi listovi stabla nisu završni ili dok se ne dostigne maksimalna dozvoljena dubina stabla.

Full metoda – slična je grow metodi, ali je unaprijed definirano na kojoj se minimalnoj dubini stabla mogu početi pojavljivati završni čvorovi.

Ramped half-and-half – prilikom generiranja jedinki se unaprijed definira samo maksimalna dubina stabla (*md*). Kreiranje jedinki se obavlja na sljedeći način:

- populacija se podijeli na *md-1* dijelova
- polovica svakog dijela generira se metodom *grow*, a druga polovica metodom *full*

3.3 Računanje dobrote jedinki

Dobrota (kvaliteta) pojedine jedinke je mjera kvalitete te jedinke u zadanom prostoru rješenja (jedinka koja je bolje obavila zadani posao ima veću dobrotu). Za neke je probleme jednostavno izračunati dobrotu jedinki (npr. računanje maksimuma funkcije – veća vrijednost funkcije označava veću dobrotu), dok se kod složenijih problema dobrota može izraziti kao, na primjer, vrijeme trajanja projekta ili vrijeme koje je potrebno da se obavi simulacija sustava koji optimiramo.

Dobro definirano računanje dobrote jedinki je jedna od ključnih stvari za uspješnost genetskog programa jer direktno utječe na izbor jedinki koje prelaze u iduću generaciju.

3.4 Selekcija

Selekcija je proces kojim se osigurava prenošenje boljeg genetskog materijala iz generacije u generaciju i stoga direktno utječe na samu kvalitetu genetskog programa. Postupci selekcije se međusobno razlikuju po načinu odabira jedinki koje će se prenijeti u sljedeću generaciju.

Prema *načinu prenošenja genetskog materijala* selekcija se dijeli na:

- **Generacijske selekcije**: proces odabire najbolje jedinke i od njih kreira novu generaciju
- **Eliminacijske selekcije**: proces selekcije eliminira najgore jedinke iz populacije

Generacijske selekcije iz populacije odabiru određen broj najboljih jedinki i od njih stvaraju međupopulaciju. Broj jedinki koje prežive selekciju je manji od veličine populacije pa se jedinke nadomještaju kopiranjem već selektiranih. Postupak može rezultirati nekvalitetnim genetskim materijalom jer se iste jedinke višestruko kopiraju te se gubi genetska raznolikost.

Eliminacijske selekcije odstranjuju slučajno odabrane jedinke iz populacije pri čemu lošije jedinke imaju veću vjerojatnost eliminacije. Eliminirane jedinke se nadomještaju križanjem postojećih.

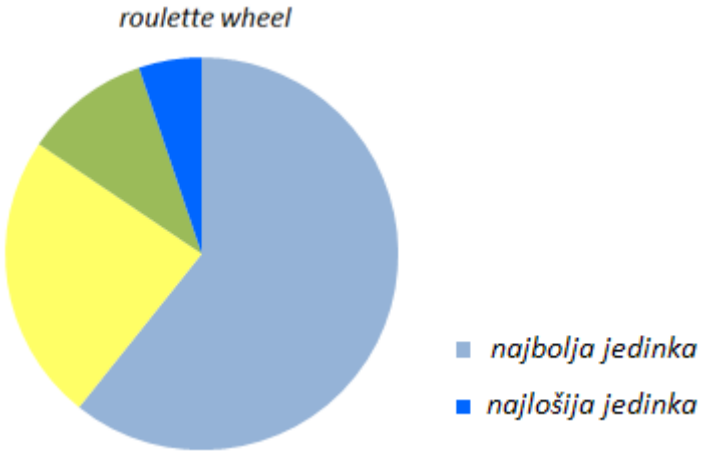
Prema načinu odabira pojedinih jedinki iz skupa svih jedinki na temelju njihove dobrote selekcija može biti na primjer:

- **Jednostavna proporcionalna selekcija** *(roulette-wheel selection)*
- **K-turnirska selekcija**

Jednostavna proporcionalna selekcija se najjednostavnije može prikazati kao kotač ruleta podijeljen na dijelove različitih veličina (slika 3.3). Za sve jedinke unutar populacije definira se njihova dobrota te se svakoj jedinki pridružuje dio kotača ruleta proporcionalan njenoj dobroti (podrazumijeva se da u ovom slučaju dobrote jedinki ne smiju biti negativni brojevi). Nakon toga odabiremo jedinke koje će činiti sljedeću generaciju na sljedeći način:

- slučajno odabiremo jednu jedinku iz populacije, pri čemu jedinke s većom dobrotom imaju veću vjerojatnost odabira
- postupak odabira ponavljamo *N* puta (*N* označava veličinu populacije)

Osim već spomenutog nedostatka da dobrote jedinki ne smiju biti negativni brojevi, glavni nedostatak ovakvog načina odabira jedinki je zagušenje populacije zbog intenzivnog kopiranja dobrih jedinki (program može „zaglaviti“ u lokalnom optimumu).



Slika 3.3 – Roulette-wheel selection

K-turnirska selekcija odabire *k* članova populacije koji sudjeluju u turniru te uspoređuje njihove dobrote (*k* označava veličinu turnira, npr. tri). Jedinke s lošijom dobrotom ispadaju iz generacije te se zamjenjuju novima koje nastaju križanjem jedinki koje su prošle turnirski odabir. Svojstvo takvog načina odabira je ***elitizam***, tj. očuvanje najboljih jedinki u generaciji (trenutno najbolja jedinka se nikad ne će eliminirati).

Algoritam k-turnirskog odabira je sljedeći (Jakobović, 2007):

Početak (GA s k-turnirskim odabirom)

Stvori populaciju P(0)

Ponavljaj

Odaberi k jedinki iz populacije

Pronađi i obriši najlošiju od k odabranih

Generiraj novu jedinku ponoću genetskih operatora

Dok nije zadovoljen uvjet zaustavljanja

Kraj

3.5 Križanje

Križanje jedinki je proces u kojem se rekombinira genetski materijal dvaju roditelja. Rezultat križanja su dvije jedinke koje od svakog roditelja nasljeđuju dio genetskog materijala. Princip križanja (kod jedinki prikazanih u obliku stabla) je sljedeći:

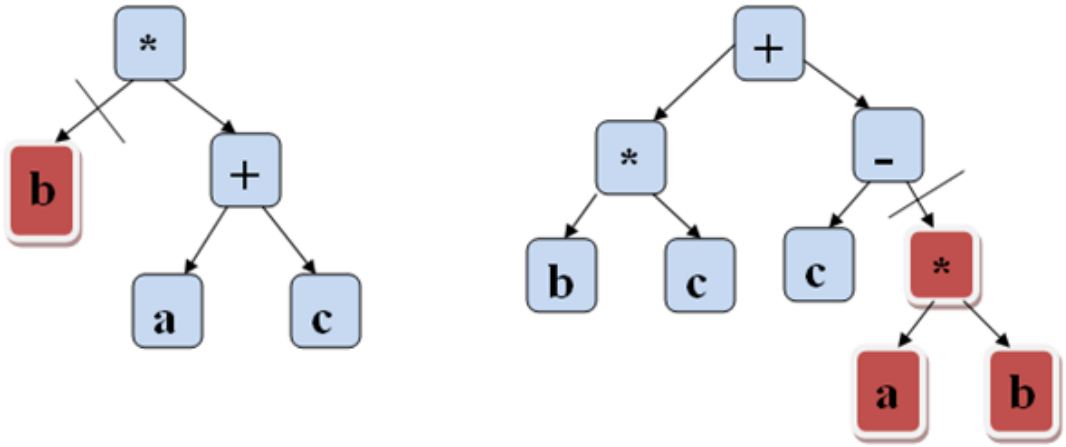
- Slučajno se odabiru točke prekida za oba roditelja (proizvoljna grana stabla)
- Podstabla ispod točke prekida se odvajaju od roditeljskog stabla
- Na praznu vezu koja je nastala micanjem podstabla iz pojedinog roditelja stavlja se podstablo koje je odvojeno od drugog roditelja

Roditeljske jedinke prikazane na slici 3.4 izvršavaju funkcije:

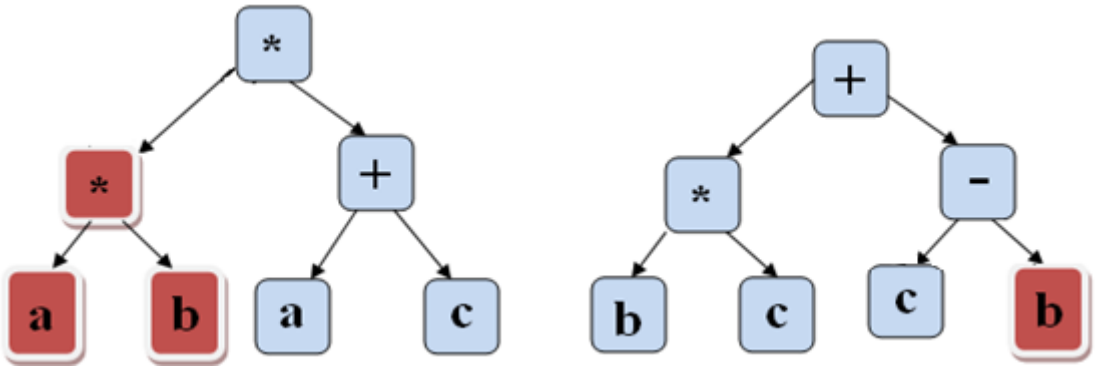
$$f = b \cdot (a + c) \tag{3.1}$$

$$g = (b \cdot c) + (c - (a \cdot b)) \tag{3.2}$$

Križenjem roditeljskih jedinki (podstabla koja se izmjenjuju označena su drugom bojom) dobivene su dvije nove jedinke (slika 3.5) koje ne moraju nužno biti bolje od svojih roditelja. Ako je novostvorena jedinka bolja od roditeljskih, njezin će se genetski materijal iskoristiti u daljnjoj evoluciji, dok će se jedinke koje su lošije od roditeljskih nakon određenog broja koraka izbaciti iz populacije.



Slika 3.4 – Roditeljske jedinke



Slika 3.5 – Rezultat križanja

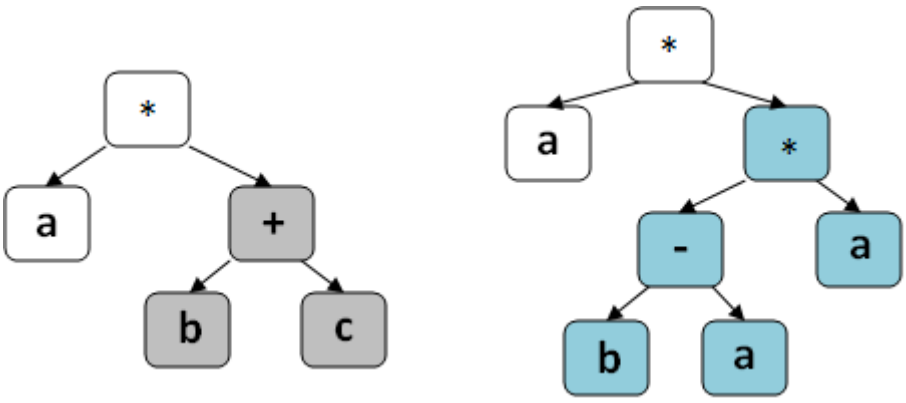
3.6 Mutacija

Mutacija u genetskom programiranju uvodi u nove jedinke neke gene koji će (možda) poboljšati dobrotu jedinke. Jedinke sa „smrtonosnim“ mutacijama će vrlo brzo biti eliminirane u postupku selekcije tako da ne će dugotrajno narušiti prosječnu dobrotu jedinki unutar populacije.

Mutacija se u genetskom programiranju izvodi na sljedeći način:

- Iz populacije se odabere jedinka na kojoj želimo izvršiti mutaciju
- U stablu koje reprezentira jedinku se nasumce odabere jedan čvor (ili list)
- Odabrani čvor se zajedno sa svojim podstablom briše iz stabla
- Na tom se mjestu zatim slučajnim odabirom stvara novo podstablo

Princip mutacije jedinki prikazan je na slici 3.6.



Slika 3.6 – Mutacija jedinke

Osnovna uloga mutacije je izbjegavanje lokalnih optimuma unutar kojih može „zaglaviti“ genetski program te obnavljanje izgubljenog genetskog materijala (koji je možda dobar, ali je bio iskorišten na loš način).

4 UGRADNJA SEMANTIČKE INFORMACIJE U GENETSKO PROGRAMIRANJE

Cilj ovog rada je ispitivanje utjecaja semantičke informacije na kvalitetu i konvergenciju dobivenog rješenja. U ovom poglavlju opisana je osnovna ideja te uloga semantike u gentskom programu, dok je o samoj implementaciji semantike više rečeno u sljedećem poglavlju.

S obzirom na leksiku, sintaksu i semantiku, postoje četiri vrste rješenja:

- Leksički neispravno rješenje
- Leksički ispravno, ali sintaksno neispravno rješenje
- Sintaksno ispravno, ali semantički neispravno rješenje
- Semantički ispravno rješenje

Pritom treba uočiti da rješenje ne može biti sintaksno ispravno ukoliko nije leksički ispravno, kao i da rješenje ne može biti semantički ispravno ukoliko nije sintaksno ispravno. Budući da je rješenje u genetskom programu reprezentirano stablom, u daljnjem će tekstu pojmovi „rješenje“ i „stablo“ biti ekvivalentni.

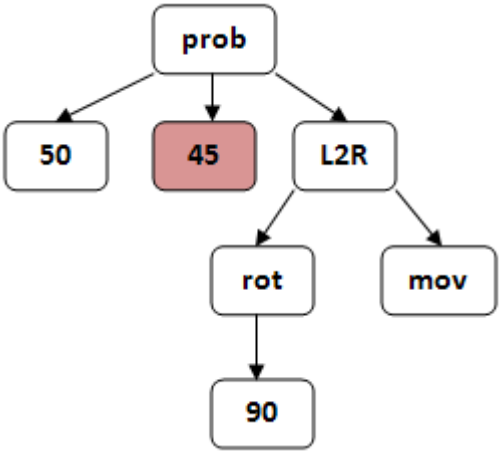
Leksički neispravna stabla mogu se dobiti, na primjer, dodavanjem pogrešnih čvorova. Ako se stablo gradi od unaprijed definiranih čvorova, sva rješenja su leksički ispravna pa leksički neispravna stabla ne će dalje biti razmatrana.

Primjeri sintaksno ispravnog i neispravnog stabla dani su u poglavlju 4.1. Sintaksno neispravno stablo nije od koristi za genetski program jer se ne može evaluirati pa je potrebno svaki genetski program modelirati tako da poštuje zadana sintaksna pravila. Primjeri semantički ispravnog i neispravnog stabla dani su u poglavlju 4.2. Semantički neispravno stablo može se evaluirati, ali ne pruža korisnu informaciju o rješenju, odnosno vrijednosti dobivene semantički neispravnim rješenjima nemaju značenje.

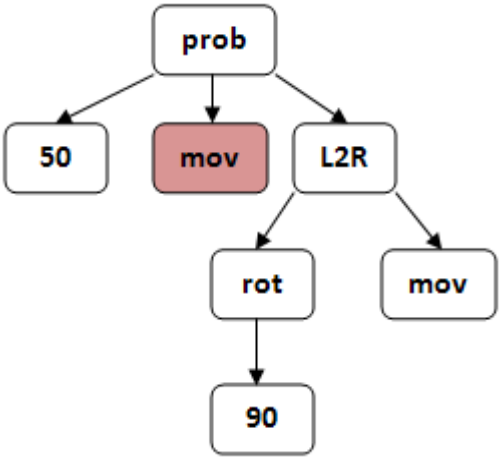
4.1 Primjer sintaksno neispravnog rješenja

Neka je dan sljedeći skup čvorova:

- **prob** – izvršni vjerojatnosni čvor; ima troje djece, od kojih prvo predstavlja vjerojatnost *p* u postocima, a drugo i treće su izvršni čvorovi koji se izvršavaju s vjerojatnošću *p*, odnosno (*1-p*) respektivno (izvršava se samo jedan čvor)
- **num** – numerički čvor (umjesno *num* prikazana je konkretna vrijednost)
- **L2R** – izvršni blok čvor; izvršavaju se sva podstabla s lijeva na desno
- **rot** – izvršni čvor; rotiranje
- **mov** – izvršni čvor; pomak



Slika 4.1 – Sintaksno neispravno stablo



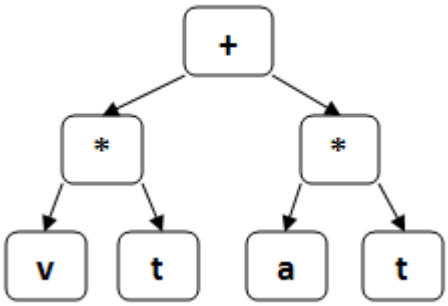
Slika 4.2 – Sintaksno ispravno stablo

Stablo na slici 4.1 je sintaksno neispravno jer je drugo dijete vjerojatnosnog čvora (prikazan drugom bojom) numerički, a ne izvršni čvor. Sintaksno neispravno stablo ne prikazuje moguće rješenje problema pa se stoga ne bi smjelo pojavljivati u genetskom programu. Stablo na slici 4.2 je sintaksno ispravno. Numerički čvor zamijenjen je (sintaksno ispravnim) izvršnim čvorom te dobiveno stablo predstavlja moguće rješenje problema.

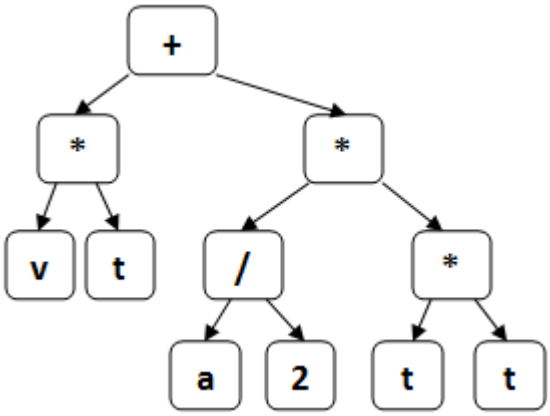
4.2 Primjer semantički neispravnog rješenja

Neka je dan sljedeći skup čvorova s pripadajućim semantičkim pravilima:

- **add (+)** – operator zbrajanja; djeca moraju imati istu jedinicu kao i rezultat
- **mul (*)** – operator množenja; jedinica rezultata je umnožak jedinica djece
- **t** – podatkovni čvor (vrijeme); jedinica: [s]
- **v** – podatkovni čvor (brzina); jedinica: [m/s]
- **a** – podatkovni čvor (ubrzanje); jedinica: [m/s^2]



Slika 4.3 – Semantički neispravna jedinka



Slika 4.4 – Semantički ispravna jedinka

Na slici 4.3 dan je primjer semantički neispravnog stabla (izraz $v*t + a*t$). Stablo je semantički neispravno jer djeca čvora *add* (*+*) nemaju jednake jedinice – jedinica lijevog djeteta je [m], a desnog [m/s]. Ako se od rješenja (stabla) zahtijeva da poštuje navedena semantička pravila, tada stablo na slici 4.3 ne pruža nikakvu korisnu informaciju te predstavlja redundantnu jedinku. Na slici 4.4 prikazano je semantički ispravno stablo. Stablo poštuje sva zadana semantička pravila te predstavlja potencijalno rješenje.

5 IMPLEMENTACIJA

U ovom poglavlju opisan je način implementacije genetskog programa za raspoređivanje u proizvoljnoj obradi. Dan je pregled osnovnih klasa i njihovih ovisnosti, korištenih čvorova te najvažnijih parametara genetskog programa (selekcija, križanje i sl.). Također, opisana je implementacija te uloga semantičke informacije u genetskom programu.

5.1 Struktura programa

Za implementaciju je korišten programski jezik *C++*. Programsko okruženje sastoji se od glavnog potprograma i skupa klasa s odgovarajućim metodama. Uloga glavnog potprograma te najvažnijih klasa, kao i njihova međuovisnost, opisana je u nastavku.

Glavni potprogram (main) – ima ulogu inicijalizacije osnovnih komponenti genetskog programa: skupa osnovnih čovorova (klasa *PrimitiveSet*), algoritma (*Algorithm*) te funkcije evaluacije dobrote jedinki (*MyEval*). O navedenim komponentama više će biti riječi kasnije u ovom poglavlju.

Algorithm – predstavlja osnovnu klasu genetskog programa. Uloga klase je simuliranje procesa evolucije (inicijalizacija populacije, selekcija, reprodukcija, križanje, mutacija) te su u skladu s tim razvijene odgovarajuće metode.

Tree - predstavlja reprezentaciju stabla, odnosno jedinke genetskog programa. Stablo je predstavljeno vektorom čvorova (*Node*) što omogućuje brz pristup pojedinom elementu, ali sporije ubacivanje ili izbacivanje pojedinog elementa. Posljedica toga je nešto sporije izvođenje križanja i mutacije. Čvorovi su u vektoru zapisani u *preorder* poretku. Za manipulaciju nad stablima razvijene su sljedeće osnovne metode:

- *Create* – stvara se novo stablo s nasumično odabranim čvorovima
- *Update* – osvježavaju se informacije u čvorovima stabla
- *Cross* – križanje dvaju odabranih stabala

- *Mutate* – mutacija odabranog stabla
- *Calculate* – evaluacija stabla
- *PrintTree* – ispis stabla u *preorder* poretku
- *LoadTree* – rekonstrukcija stabla iz skupa čvorova u *preorder* poretku

Prim – abstraktna klasa koju nasljeđuju klase završnih i nezavršnih čvorova (*Add, Mul, Jw, ...*). Njena uloga je implementacija osnovne funkcionalnosti koju čvorovi stabla moraju imati. Klasa sadrži članske varijable *des* i *nArgIn* koje definiraju opis i broj argumenata čvora, te virtualne funkcije *Execute* (obavlja funkciju čvora), *SetVal* i *GetVal* koje postavljaju, odnosno vraćaju vrijednost čvora.

PrimitiveSet – služi za pohranu i dohvat čvorova koji se koriste u genetskom programu. Čvorovi se dodaju pozivom metode *AddPrim*, a dohvaćaju metodama *RetRandFunc*, *RetRandTerm* i *RetRandPrim*, ovisno o tome želi li se dohvatiti funkcijski, završni ili proizvoljan čvor. Također, definirana je metoda *RetByDes* koja vraća pokazivač na čvor čiji opis (*string des*) odgovara željenom.

Node – predstavlja jedan čvor jedinke. Svaki čvor sadrži podatke o svojoj *veličini* (broj čvorova u podstablu čiji je korijen dotični čvor), *dubini* te *maksimalnoj dubini čvora u* nekom *podstablu*. Posljednji se podatak koristi pri ograničavanju maksimalne dubine stabla.

MyEval – služi za računanje dobrote jedinke. Sadrži osnovne metode *Initialize,* koja služi za inicijalizaciju potrebnih podataka, te *Execute,* koja simulira izvođenje jedinke i vraća njenu dobrotu.

5.2 Završni i nezvršni čvorovi

Kao što je već rečeno u poglavlju o genetskom programiranju, poželjno je da čvorovi stabla sadržavaju samo osnovne informacije o problemu koji se rješava. U skladu s tim, definiran je relativno malen broj čvorova koji predstavljaju osnovne matematičke operacije (nezavršni čvorovi), odnosno osnovne karakteristike poslova koje treba obaviti (završni čvorovi). Pregled čvorova koji su odabrani za rješavanje problema raspoređivanja u proizvoljnoj obradi dan je u tablici 5.1.

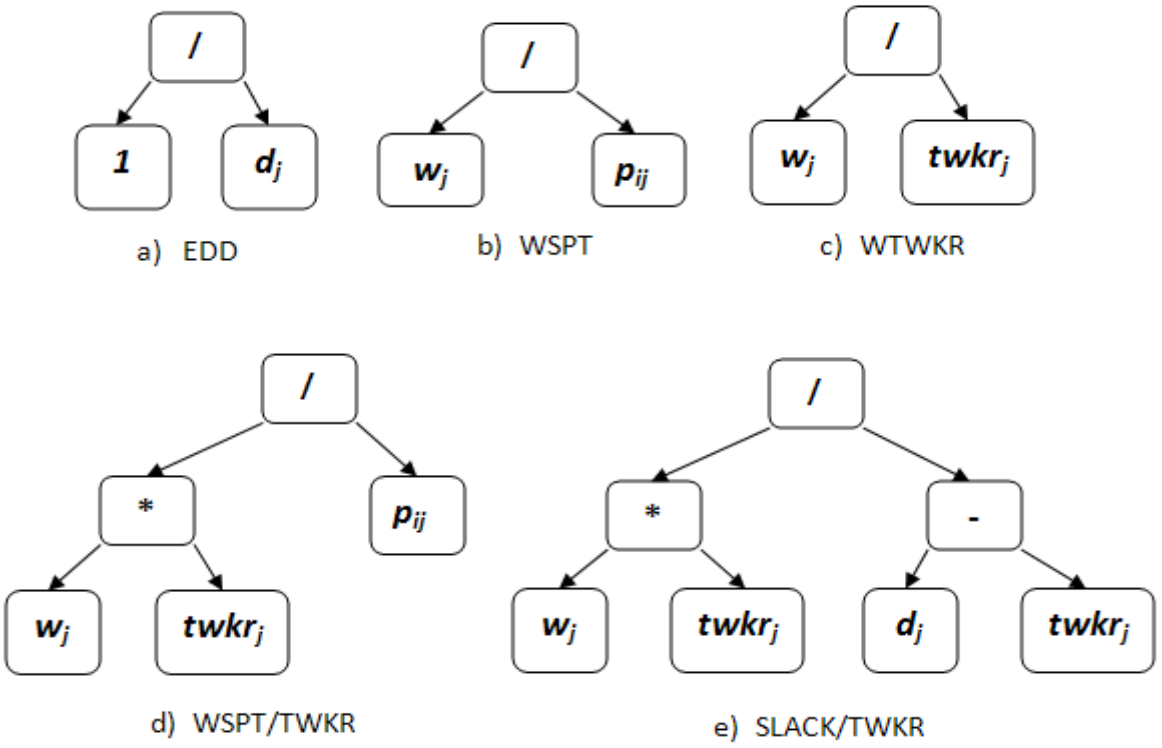
Tablica 5.1 – Korišteni čvorovi

Oznaka funkcijskog čvora (opis)	Definicija
Add (+)	Binarni operator zbrajanja
Sub (-)	Binarni operator oduzimanja
Mul (*)	Binarni operator množenja
Div (/)	Zaštićeno dijeljenje: Div(a, b) = $\begin{cases} 1, & \lvert b \rvert < 0.000001 \\ \frac{a}{b}, & inače \end{cases}$
Pos(pos)	Unarni operator „+“: Pos(a) = max{a, 0}
Oznaka podatkovnog čvora	Definicija
twkrj	Preostalo trajanje obrade posla
twkj	Ukupno trajanje obrade posla
wj	Težina posla
wj_av	Prosječna težina posla
dj	Željeno vrijeme završetka posla
JI	Pozitivna dopuštena odgoda: *max*{*d – p – time*, 0}
avjt	Prosječno trajanje obrade cijelih poslova
avjt_r	Prosječno trajanje obrade preostalih poslova
pij	Trajanje sljedeće operacije posla

Programska implementacija problema omogućuje jednostavno dodavanje novih tipova čvorova, pa je navedeni popis čvorova fleksibilan.

5.3 Prikaz osnovnih funkcija prioriteta binarnim stablom

Pomoću prethodno navedenih čvorova je, između ostalog, moguće implementirati većinu osnovnih funkcija računanja prioriteta operacije navedenih u poglavlju 2.3.2. Na slci 5.1. dani su primjeri kako bi izgledala takva stabla. Potrebno je napomenuti da je, uz dodavanje odgovarajućih čvorova, moguće implementirati proizvoljnu funkciju računanja prioriteta operacije, ali umetanje velikog broja čvorova u genetski program često nije preporučljivo zbog povećanja prostora mogućih rješenja te ubacivanja “viška informacija“ u program (čvorovi moraju sadržavati samo osnovna svojstva problema).



Slika 5.1 – prikaz funkcija prioriteta pomoću jedinki genetskog programa

Na slici 5.1 vidljivo je da implementacija osnovnih funkcija računanja prioriteta pojedinog posla pomoću binarnog stabla zahtijeva relativno malen broj čvorova, pa se može pretpostaviti da će se veći dio tih funkcija u postupku izvođenja genetskog programa doista i pojaviti. Zbog toga se očekuje da će genetski program na skupu primjera za učenje pronaći rješenje koje je bolje ili barem jednako dobro kao najbolja od dotičnih funkcija računanja prioriteta posla.

5.4 Evaluacija jedinke i funkcija dobrote

Poslovi koji služe za evaluaciju jedinke (primjeri za učenje) te dodatni podaci o poslovima (*vrijeme željenog završetka*, *vrijeme dolaska*, *težina operacije*) učitavaju se iz ulaznih datoteka te se spremaju u odgovarajuće podatkovne strukture klase „*MyEval*“. Zbog bolje generalizacije, svaka se jedinka (pravilo raspoređivanja) testira nad svim primjerima za učenje te se naknadno na odgovarajući način računa ukupna dobrota jedinke (budući da u problemu raspoređivanja veća „*dobrota*“ zapravo označava veću

zakašnjelost poslova, jedinka s manjom „*dobrotom*“ je bolja pa se može upotrebljavati i termin „*kazna*“). Osnovni princip evaluacije jedinke za jedan primjer za učenje prikazan je algoritmom 5.1.

```
dok (postoje neizvedene operacije) {
     čekaj dok stroj za koji postoje operacije ne postane
     raspoloživ;
     odredi prioritete svih operacija koje čekaju na stroj;
     rasporedi operaciju najvećeg prioriteta;
     odredi sljedeću operaciju posla;
     obnovi vrijeme raspoloživosti stroja i sljedeće
     operacije posla;
  }
```

Algoritam 5.1 – Postupak raspoređivanja

Za svaki primjer za učenje računa se težinsko zaostajanje, te se normalizira prema formuli (5.1):

$$\hat{T}_w = \frac{T_w}{n \cdot \overline{w} \cdot \overline{p}} \tag{5.1}$$

gdje je *n* broj poslova u primjeru za učenje, $\overline{w}$ je srednja vrijednost težina poslova, a $\overline{p}$ je srednje trajanje poslova. Vrijednost težinskog zaostajanja se za pojedini primjer za učenje dijeli s brojem poslova u dotičnom primjeru kako bi se postigla podjednaka težina za primjere s različitim brojem poslova. Također, obavlja se dijeljenje sa srednjom težinskom vrijednošću i srednjim trajanjem obrade.

Ukupna dobrota jedinke (stabla) dobiva se zbrajanjem normiranih težinskih zakašnjelosti za sve primjere za učenje:

$$F = \sum_i \hat{T}_{w,i} \tag{5.2}$$

gdje je *F* ukupna dobrota jedinke, a $\hat{T}_{w.i}$ normirano težinsko zaostajanje *i*-tog primjera za učenje.

5.5 Parametri evolucijskog procesa:

Parametri evolucijskog procesa (selekcija, križanje, mutacija, uvjet zaustavljanja) mogu imati velik utjecaj na kvalitetu konačnog rješenja pa ih je potrebno detaljnije opisati.

Kao postupak ***selekcije*** jedinki je odabran *turnirski eliminacijski odabir***:**

- Slučajno se odabiru četiri jedinke iz populacije
- Eliminiraju se dvije najlošije

Postupak se ponavlja dok se ne eliminira dovoljan broj jedinki. Turnirski eliminacijski odabir ima (važno) svojstvo *elitizma* – najbolja jedinka u generaciji (ili u ovom slučaju dvije najbolje) nikad ne će biti izbačena.

Načini ***križanja*** i ***mutacije*** jedinki jednaki su načinima opisanim u poglavlju o genetskom programiranju, ali se, ako je uključena semantika, mora paziti da čvorovi koji se križaju ili mutiraju imaju jednaku semantičku informaciju. Također je omogućena mutacija jedinke tako da se proizvoljan čvor zamijeni *„sličnim“* čvorom (metoda *RetSimNode* u klasi *PrimitiveSet*). Sličan čvor je u ovom slučaju onaj koji ima jednak broj djece i jednaku semantičku informaciju (uz uključenu semantiku).

Kao ***uvjet zaustavljanja programa*** odabrana su dva kriterija od kojih barem jedan mora biti zadovoljen:

- *unaprijed definirani broj iteracija* koje program treba izvesti
- *određeni broj generacija bez poboljšanja*

Parametri genetskog programa te njihove pretpostavljene vrijednosti pregledno su zapisani u tablici 5.2.

Tablica 5.2 – Pretpostavljeni parametri genetskog programa

Parametar/operator	Opis
Semantika	Uključena
Veličina populacije	256
Način odabira	Eliminacijski
Operator odabira	Turnirski
Uvjet zaustavljanja	300 generacija ili 50 generacija bez poboljšanja
Mutacija	Vjerojatnost 0.5%
Inicijalizacija	*full metoda*, minimalna dubina 2, maksimalna dubina 6
Konačno rješenje	Maksimalna dubina 15

5.6 Semantička informacija

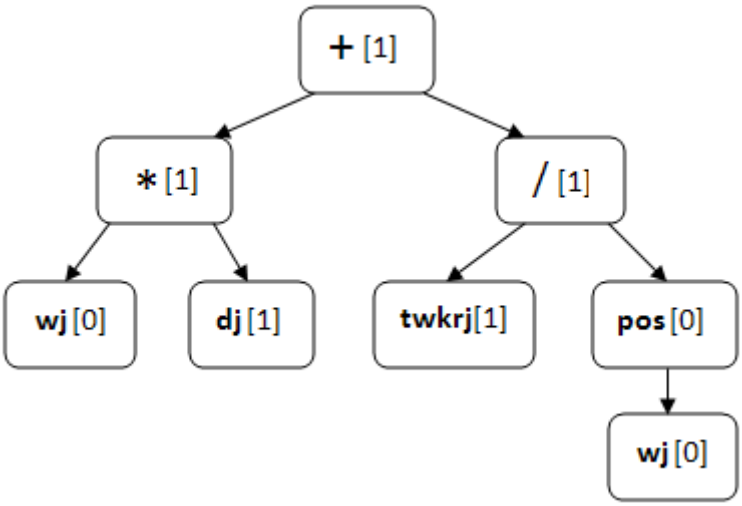
Uloga semantičke informacije je smanjivanje prostora mogućih rješenja. Ako su poznata semantička pravila izgradnje željene jedinke, tada semantička informacija omogućuje brže pronalaženje željene ili dovoljno dobre jedinke.

Semantička informacija je u programu zapisana u obliku vektora potencija (*char*) unutar svakog čvora (dva čvora imaju jednaku semantičku informaciju ako su im vektori potencija jednaki). Za problem raspoređivanja u proizvoljnoj obradi jedina jedinica koja se koristi je *vrijeme* pa se vektor potencija sastoji samo od jednog elementa. Također u programu je ograničen raspon pojedinih potencija (unutar [-4, 4]).

Semantička informacija čvora koristi se prilikom *stvaranja novog stabla*, *križanja* i *mutacije*:

Stvaranje novog stabla – budući da za problem raspoređivanja u proizvoljnoj obradi nije poznata potencija željene jedinice, prilikom inicijalizacije početne populacije korijenskom se čvoru pridružuje slučajno odabrana potencija (unutar definiranog raspona). Primjer semantički ispravnog stabla dan je na slici 5.2. Korijenski čvor ima

proizvoljno definiranu potenciju, a ostali čvorovi se generiraju u skladu sa semantičkim pravilima (potencija svakog čvora prikazana je u uglatoj zagradi u samom čvoru).



Slika 5.2 – Primjer izgradnje semantički ispravne jedinke

Križanje – semantički ispravno križanje svodi se na pronalaženje čvorova s jednakom semantičkom informacijom. Kako bi se taj proces ubrzao, čvorovi stabala koja su prošla turnirski eliminacijski odabir sortiraju se prema potenciji jedinice. Time se složenost pronalaženja čvorova za križanje smanjuje s kvadratne na logaritamsku.

Mutacija – semantički ispravna mutacija je jednaka mutaciji bez semantike, pri čemu treba paziti da novo podstablo ima jednaku semantičku informaciju kao i izbačeno.

6 ANALIZA REZULTATA

Za potrebe ocjenjivanja kvalitete genetskog programa definiran je skup primjera za učenje te skup primjera za ocjenjivanje. Također, za usporedbu s postojećim strategijama raspoređivanja korišten je skup primjera iz literature (*Taillard*). Kvaliteta generiranih jedinki (odnosno pravila raspoređivanja) uspoređuje se s najčešće korištenim pravilima raspoređivanja (navedeni u poglavlju 2.3.2) pomoću mjerila vrednovanja kvalitete rasporeda (navedena u poglavlju 2.3.3).

Nad primjerima za učenje genetskim je programom generirano 24 semantički ispravne jedinke te isti broj jedinki uz isključenu semantiku. Najbolje jedinke uz uključenu i isključenu semantiku dobivene su testiranjem generiranih jedinki nad skupom primjera za ocjenu (pritom je moguće zanemariti minimalni pomak u kvaliteti jedinke ako je „lošija“ jedinka bitno manja). Najbolje jedinke uz uključenu (6.1) i isključenu (6.2) semantiku su sljedeće (zbog veličine jedinki i jednostavnosti zapisa, jedinke nisu prikazane u obliku stabla):

$$\frac{wj}{\frac{avjt\_r^2 \cdot pij \cdot twkrj}{wj} + twkj \cdot pij \cdot (JI \cdot avjt\_r + JI^2)} \tag{6.1}$$

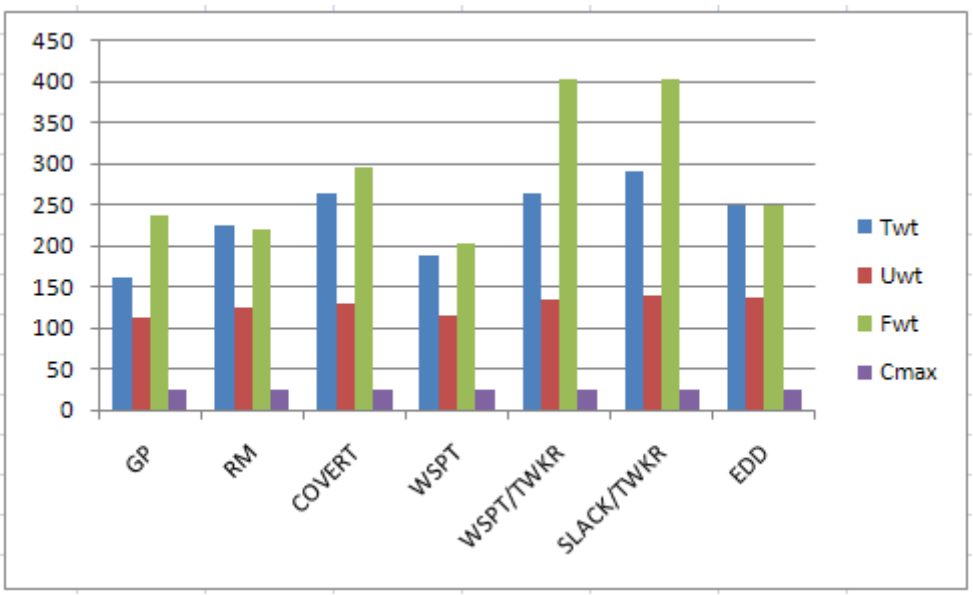
$$\frac{Pos(wj)^2 \cdot (Pos(wj) + twkrj + pij + wj) \cdot twkrj}{twkj \cdot pij \cdot (2 \cdot avjt_r + twkrj + 2 \cdot JI) \cdot (JI^2 + wj_{av} + twkrj^2 + JI^2)} \tag{6.2}$$

Rezultati za svaki skup primjera dani su u dva dijela. U prvom dijelu prikazuje se iznos funkcije dobrote za različita pravila raspoređivanja (manja dobrota označava bolju jedinku). U drugom se dijelu za svaki ispitni primjer pronalazi najbolje rješenje (dobiveno od bilo kojeg pravila), te se za svako pravilo određuje postotak primjera u kojima pravilo daje rješenje jednako najboljem, odnosno određuje se u kolikom postotku dotično pravilo dominira nad drugim pravilima raspoređivanja.

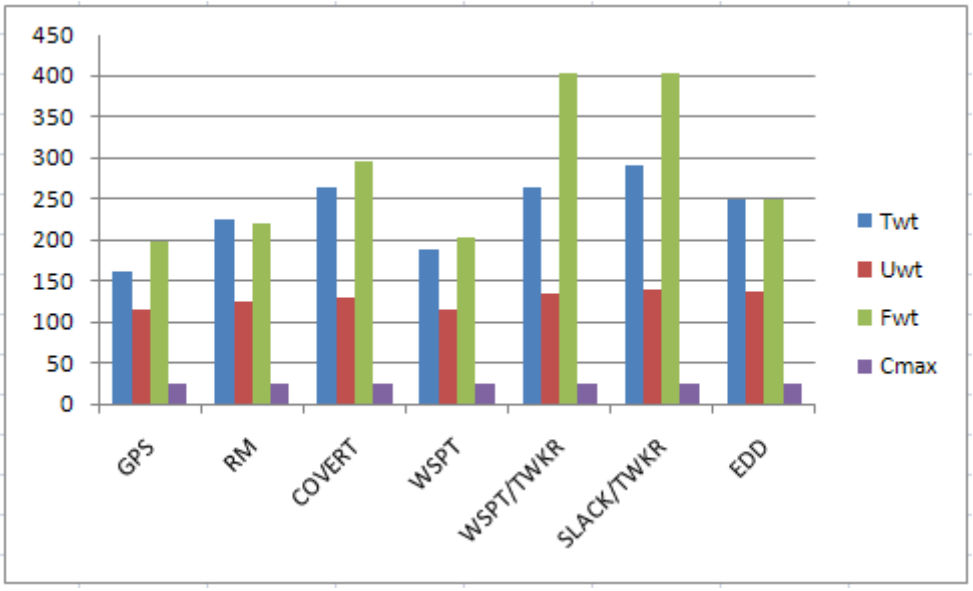
6.1 Primjeri za učenje

Za pronalaženje funkcije prioriteta genetskim programiranjem definirano je 160 primjera za učenje. Primjeri su statičkog tipa, odnosno svi poslovi koje treba rasporediti dostupni su od početka rada sustava. Iako se prilikom učenja pravila raspoređivanja genetskim programom kao mjerilo vrednovanja rasporeda koristi samo težinsko zaostajanje (*Twt*), zbog potpunijeg uvida u učinkovitost pojedine metode prikazani su i rezultati za težinsku zakašnjelost (*Uwt*), ukupnu duljinu rasporeda (C_{max}) i težinsko protjecanje (*Fwt*). Treba također napomenuti da su rezultati za težinsko protjecanje zbog znatno veće „dobrote" podijeljeni s odgovarajućim faktorom (faktor tri za primjere iz literature ili pet za primjere za učenje i ocjenu), kako bi se povećala preglednost grafa. Time nije narušen relativan odnos kvalitete različitih pravila raspoređivanja.

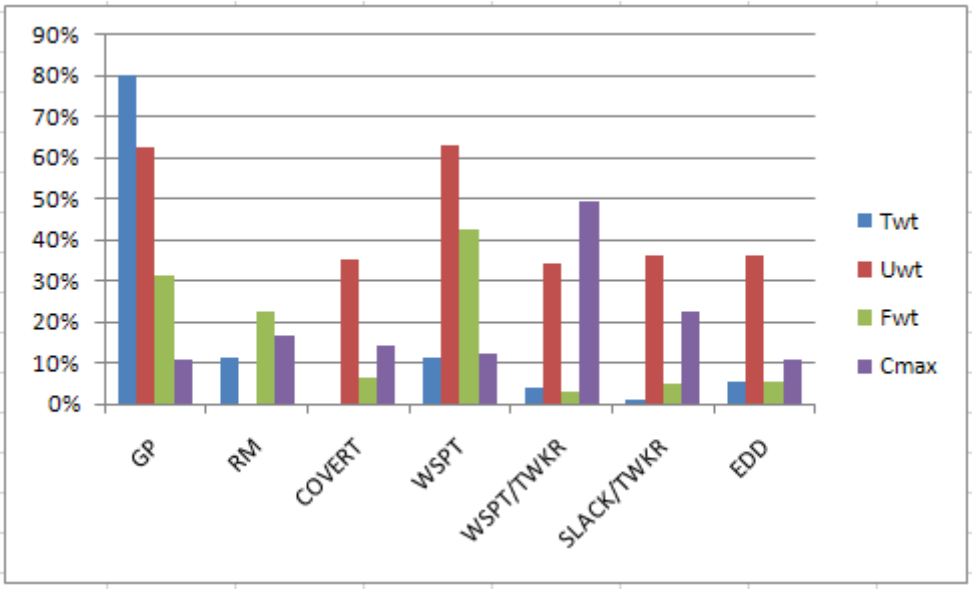
Iz prikazanih rezultata (slike 6.1, 6.2, 6.3 i 6.4) vidljivo je da rješenja dobivena genetskim programom (uz uključenu i isključenu semantiku) dominiraju nad ostalim pravilima uz mjerilo vrednovanja *Twt* (težinsko zaostajanje). To je očekivano jer je genetski program koristio upravo tu funkciju prilikom učenja pravila raspoređivanja. Također, generirane jedinke daju zadovoljavajuće rezultate za pravila *Uwt* (težinska zakašnjelost) i *Fwt* (težinsko protjecanje). Jedino mjerilo vrednovanja za koje genetski program daje loše rezultate je C_{max} (ukupna duljina rasporeda), ali to nije neočekivano jer ta funkcija ne sadrži informacije o željenom završetku i težini poslova, koje koristi više čvorova genetskog programa.



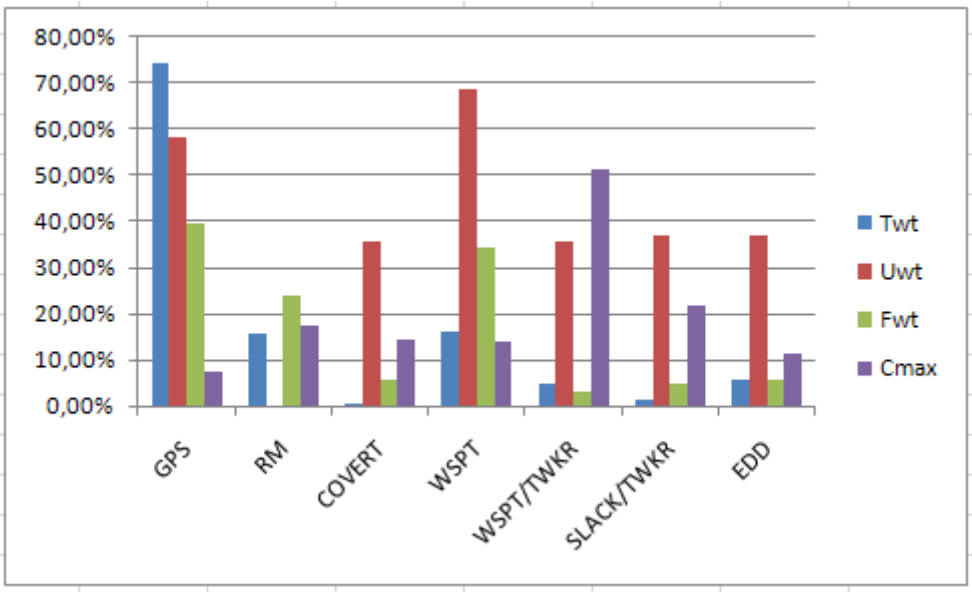
Slika 6.1 – Optimiranje – skup primjera za učenje, isključena semantika



Slika 6.2 – Optimiranje – skup primjera za učenje, uključena semantika



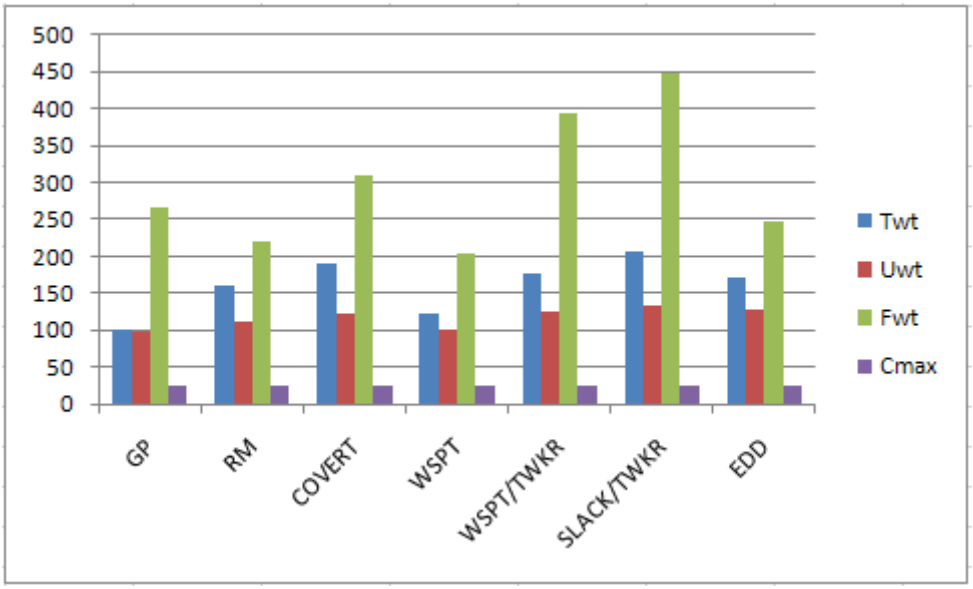
Slika 6.3 – Postoci dominacije – skup primjera za učenje, isključena semantika



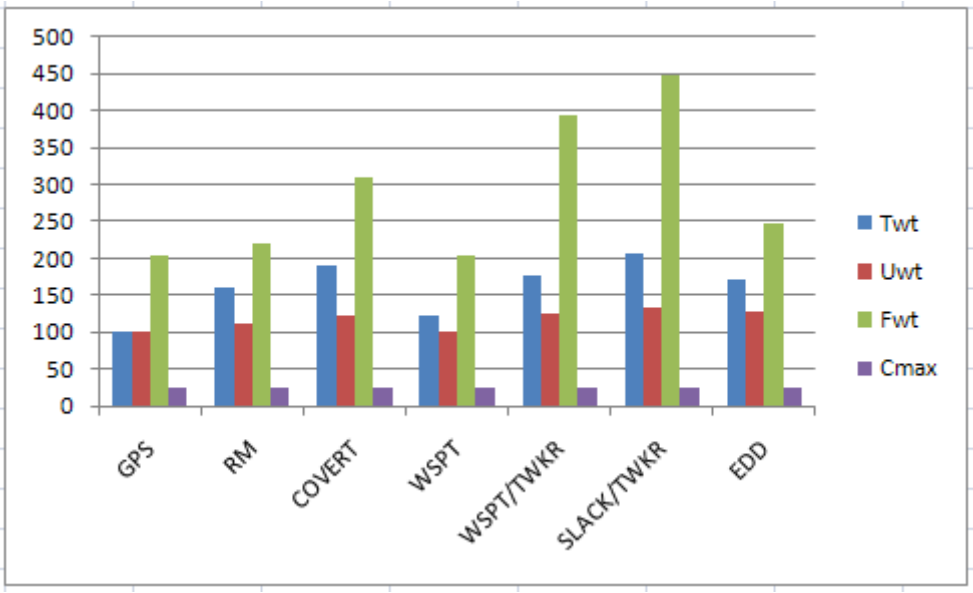
Slika 6.4 – Postoci dominacije – skup primjera za učenje, uključena semantika

6.2 Primjeri za ocjenu

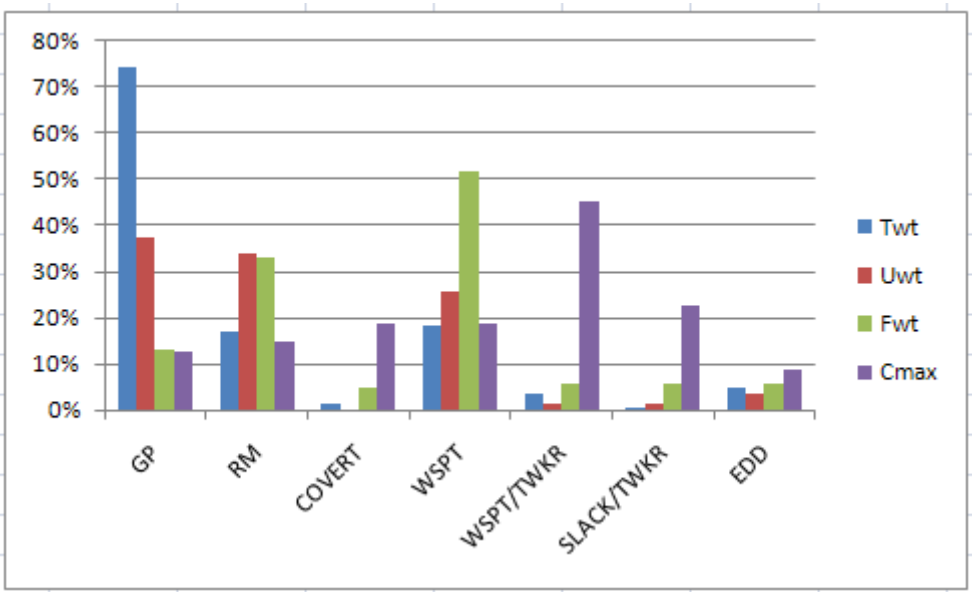
Za ocjenu kvalitete jedinki (pravila raspoređivanja) definirano je 160 primjera. Primjeri su ekvivalentni primjerima za učenje, ali nisu dostupni genetskom programu za vrijeme učenja pravila raspoređivanja. Dobiveni rezultati prikazani su na slikama 6.5, 6.6, 6.7 i 6.8.



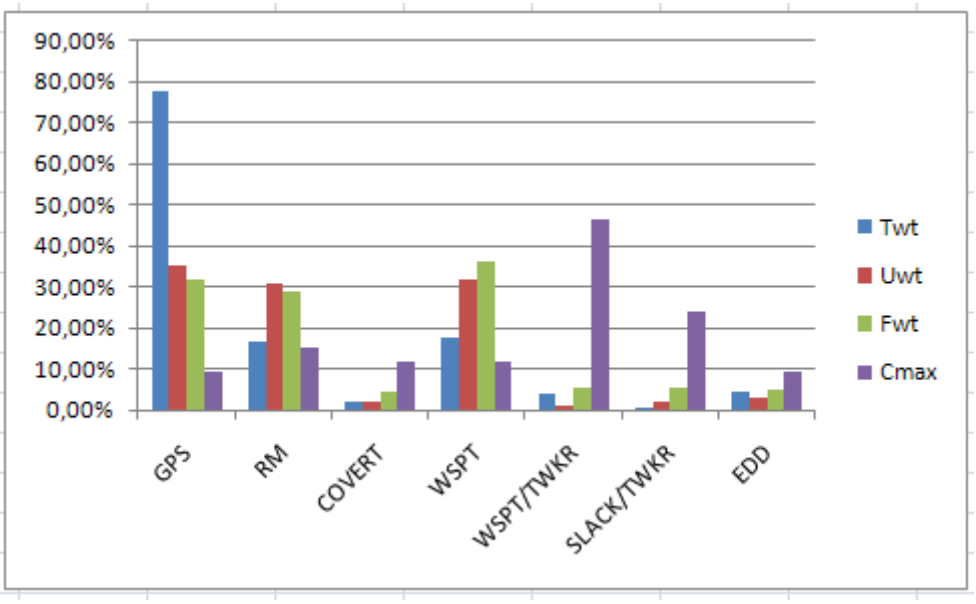
Slika 6.5 – Optimiranje - skup primjera za ocjenu, isključena semantika



Slika 6.6 – Optimiranje – skup primjera za ocjenu, uključena semantika



Slika 6.7 – Postoci dominacije – skup primjera za ocjenu, isključena semantika

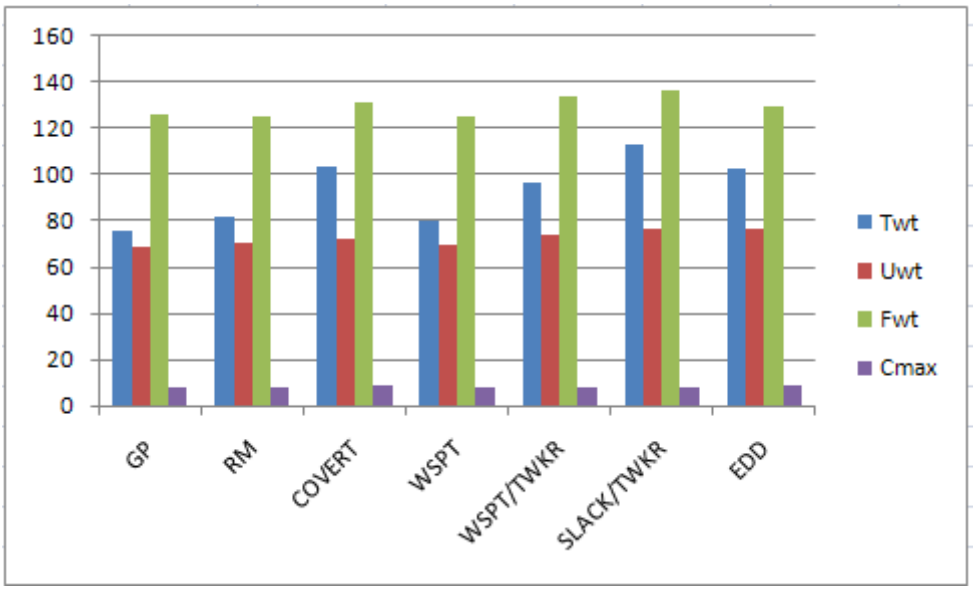


Slika 6.8 – Postoci dominacije – skup primjera za ocjenu, uključena semantika

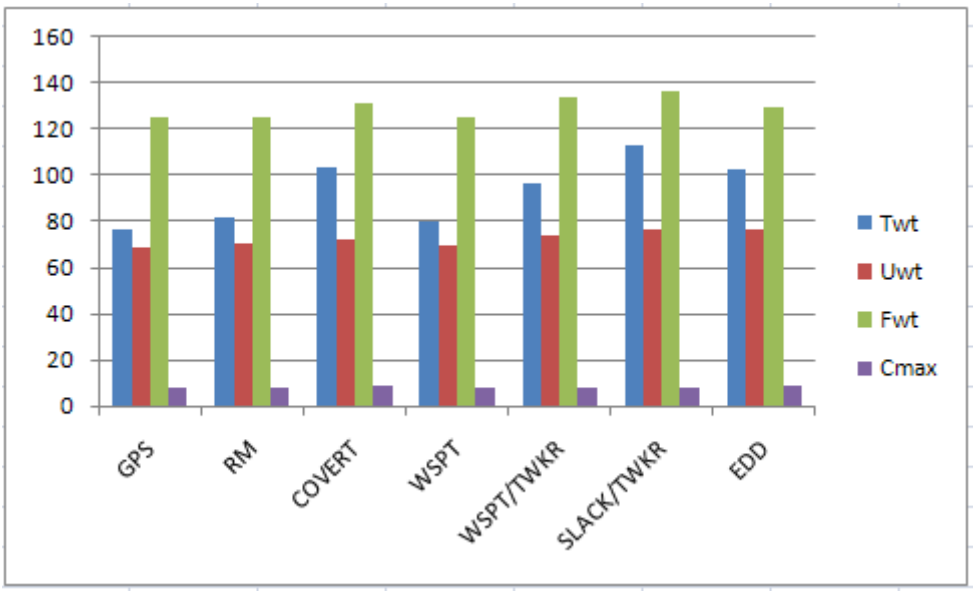
Rezultati nad skupom za učenje pokazuju da generirana pravila raspoređivanja ponovno dominiraju nad ostalim pravilima uz mjerilo vrednovanja *Twt* (težinsko zaostajanje) te daju loše rezultate za vrednovanje ukupne duljina rasporeda (C_{max}). Međutim, zanimljivo je uočiti da semantički ispravna jedinka daje znatno bolje rezultate za težinsko protjecanje (*Fwt*) od jedinke generirane uz isključenu semantiku.

6.3 *Primjeri iz literature*

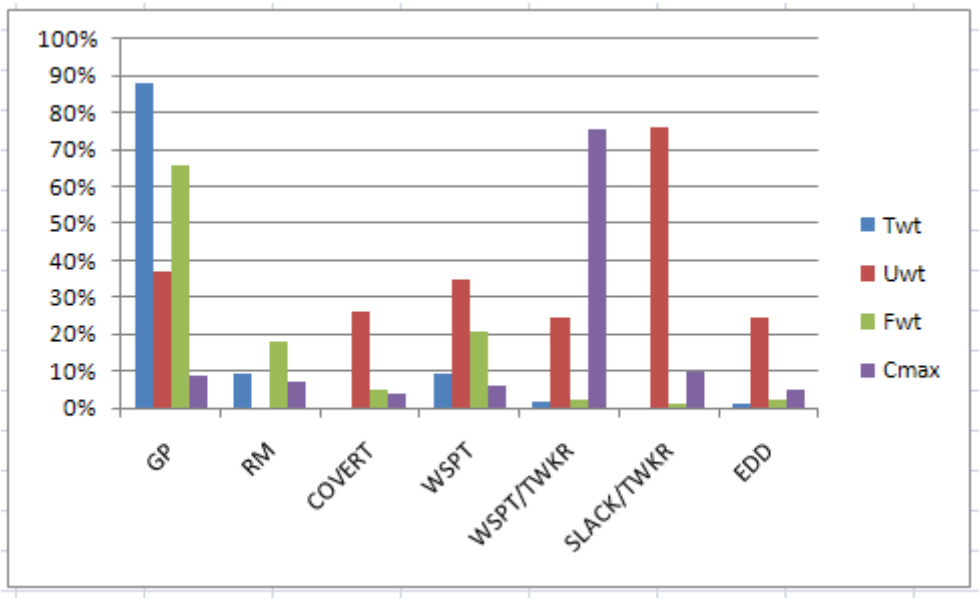
Dodatna (i realnija) procjena kvalitete generiranih pravila raspoređivanja dobivena je primjerima iz literature (*Taillard*). Korišteno je 80 primjera, a vrednovanje kvalitete rasporeda jednako je kao i u primjerima za učenje i ocjenu. Dobiveni rezultati prikazani su na slikama 6.9, 6.10, 6.11 i 6.12.



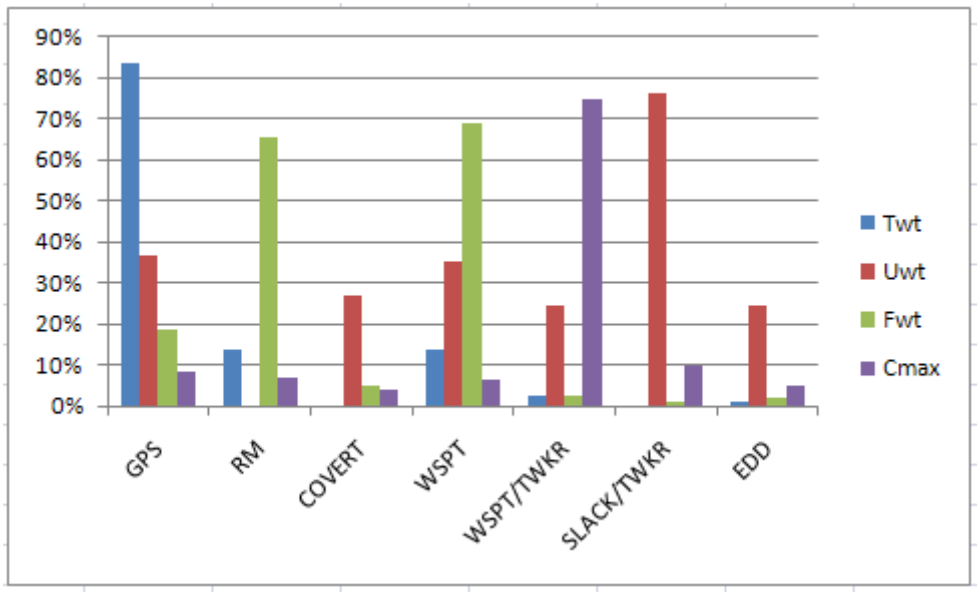
Slika 6.9 – Optimiranje – skup primjera iz literature, isključena semantika



Slika 6.10 – Optimiranje – skup primjera iz literature, uključena semantika



Slika 6.11 – Postoci dominacije – skup primjera iz literature, isključena semantika



Slika 6.12 – Postoci dominacije – skup primjera iz literature, uključena semantika

Iz prikazanih rezultata vidljivo je da jedinke (pravila) generirane pomoću genetskog programa ponovno dominiraju nad ostalim pravilima uz mjerilo vrednovanja *Twt* (težinsko zaostajanje). To pokazuje dobro svojstvo generalizacije dobivenih pravila. Također, pravilo izvedeno uz isključenu semantiku dominira nad ostalim pravilima za mjerilo vrednovanja *Fwt* (težinsko protjecanje), dok semantički ispravno pravilo daje dobre rezultate za težinsku zakašnjelost (*Uwt*).

6.4 Usporedba rješenja sa semantikom i bez semantike

Utjecaj semantičke informacije na kvalitetu rješenja ispitan je pomoću 48 jedinki (24 s uključenom i 24 s isključenom semantičkom informacijom) dobivenih pokretanjem genetskog programa nad primjerima za učenje. Nad generiranim jedinkama proveden je statistički *t-test*.

Statistička razlika za skup ispitnih primjera je $t = 6.1877$ u korist analitički neispravnih rješenja uz P vrijednost $1.52 * 10^{-6}$. Uz nivo značajnosti od $\alpha = 0.05$ null-hipoteza da su distribucije jednake ne može se prihvatiti, odnosno semantički ispravne jedinke nad skupom primjera za učenje daju nešto lošije rezultate.

Jedan od razloga takvog ponašanja je nemogućnost točnog definiranja semantičke informacije u problemu raspoređivanja u proizvoljnoj obradi, odnosno ne može se sa sigurnošću reći koja će pravila poštivati optimalna jedinka (npr. koji vektor potencija korijenskog čvora treba uzeti, zbrajaju li se samo čvorovi s istim vektorom potencija i sl.). To uzrokuje pretraživanje nepotrebnih prostora stanja (s neispravnom semantičkom informacijom). Također, program može pronaći relativno dobru jedinku čiji korijenski čvor ima vektor potencija različit od optimalnog, te takva jedinka može odvesti pretragu u krivom smjeru.

Radi potpunosti rezultata, *t*-test je proveden i nad primjerima za ocjenu. Statistička razlika za skup primjera za ocjenu je 2.4238 u korist analitički neispravnih rješenja uz P vrijednost 0.02496. Uz nivo značajnosti od $\alpha = 0.05$ null-hipoteza da su distribucije jednake ne može se prihvatiti, odnosno semantički ispravne jedinke nad skupom primjera za ocjenu daju nešto lošije rezultate.

7 ZAKLJUČAK

Za velik broj problema ne postoji egzaktan algoritam koji će, uz dostupna sredstva, u zadovoljavajućem vremenu pronaći željeno rješenje. Zbog toga se razvijaju algoritmi koji ne traže optimalno rješenje, već koriste određenu heuristiku za pronalaženje „dovoljno dobrog“ rješenja. Jedna od takvih metoda koja se u novije vrijeme počela intenzivno razvijati je i genetsko programiranje.

U ovom radu istražuje se utjecaj semantičke informacije na kvalitetu rješenja dobivenih genetskim programiranjem. Kao optimizacijski problem odabran je problem raspoređivanja poslova u proizvoljnoj obradi (*job-shop scheduling problem*). Nad skupom primjera za učenje generiran je skup jedinki korištenjem težinskog zaostajanja (*Twt*) kao mjerila vrednovanja rasporeda, pri čemu svaka jedinka genetskog programa predstavlja funkciju prioriteta pomoću koje se generira raspored poslova. Između generiranih jedinki odabrane su najbolje jedinke uz uključenu i isključenu semantičku informaciju, a zatim je njihova kvaliteta testirana nad primjerima iz literature (usporedbom s često korištenim strategijama).

Iako se testiranjem generiranih jedinki nad primjerima za ocjenjivanje te primjerima iz literature ne uočava značajnija razlika između semantički ispravne i semantički neispravne jedinke, nad skupom za učenje semantički neispravne jedinke u prosjeku daju nešto bolje rezultate. Jedan od razloga takvog ponašanja je nemogućnost točnog definiranja ispravne semantičke informacije u problemu raspoređivanja u proizvoljnoj obradi, što uzrokuje pretraživanje nepotrebnih prostora stanja (s neispravnom semantičkom informacijom).

Uz korištenje težinskog zaostajanja kao mjerila za vrednovanje rasporeda, pravila dobivena genetskim programom dominiraju nad ostalim pravilima raspoređivanja, te se može pretpostaviti da će genetski program za proizvoljnu mjeru vrednovanja rasporeda generirati pravilo koje je bolje od postojećih.

LITERATURA

Dragoljević, O. »Semantičko genetsko programiranje.« Zagreb: FER, 2008.

Jakobović, D. »Genetski algoritmi - predavanje.« 2007.

Jakobović, Domagoj. »Raspoređivanje zasnovano na prilagodljivim pravilima, Doktorska disertacija.« FER, 2005.

Jones, A., i J. C. Rabelo. »Survey of Job Shop Scheduling Techniques.« Gaithersburg, MD.: NISTIR, National Institute of Standards and Technology, 1998.

Leung, J. »Handbook of scheduling.« Chapman & Hall/CRC Press, 2004.

Morton, T.E., i D.W. Pentico. »Heuristic Scheduling Systems.« John Wiley & Sons, Inc., 1993.

Taillard, E. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop2.txt (pokušaj pristupa 1. 7 2010).

Yamada, T., i R. Nakano. »A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems.« Kyoto: Elsevier Science Publishers, 1992.

SAŽETAK

Naslov: Semantičko genetsko programiranje

U radu je obrađen problem raspoređivanja u okruženju proizvoljne obrade. Dana su najčešće korištena pravila raspoređivanja poslova te mjerila vrednovanja uspješnosti rasporeda. Postupak raspoređivanja definiran je u dva dijela: meta-algoritam koji koristi prioritete elemenata u sustavu za pridruživanje aktivnosti sredstvima, te funkciju koja određuje prioritete elemenata. Prioritetna funkcija dobiva se pomoću genetskog programiranja. Također, istražen je utjecaj semantičke ispravnosti jedinke na kvalitetu dobivenog rješenja.

Za dani problem definirani su skupovi ispitnih primjera za učenje i ocjenu te su dobiveni algoritmi raspoređivanja uspoređeni s postojećima. Algoritmi raspoređivanja izvedeni pomoću genetskog programiranja daju značajno bolje rezultate od postojećih uz vrednovanje pomoću težinskog zaostajanja, dok uz ostala mjerila vrednovanja daju rezultate koji su bliski rezultatima najčešće korištenih pravila.

Ključne riječi: raspoređivanje u okruženju proizvoljne obrade, pravila raspoređivanja, mjerila vrednovanja, meta-algoritam, funkcija prioriteta, genetsko programiranje, semantička ispravnost, težinsko zaostajanje

ABSTRACT

Title: Semantic genetic programming

This paper describes the *job shop scheduling* problem. The common used scheduling rules, and methods for evaluation are described. Scheduling algorithms are defined with two components: one component represents meta-algorithm which operates in scheduling enviroment, and the other represents an apropriate scheduling policy which derives job or machine priorities. The scheduling policy is evolved with genetic programming. This paper also studies the effect of semantic correctnes of the entitiy on the quality of obtained solutions.

For the given scheduling enviroment, a set of learning and a set of evaluation examples is defined. The scheduling policies derived with genetic programming are compared with existing algorithms. The derived polices show significantly better results then other known priority funciotns if weighted tardiness is chosen as the evaluation method. For different evaluation methods, the derived policies show similar results as the commonly used policies.

Key words: job-shop scheduling problem, scheduling rules, evaluation methods, meta-algorithm, scheduling policy, genetic programming, semantic correctnes, weighted tardiness

DODATAK A – TABLICE REZULTATA TESTIRANJA

Tablica A.1 – Optimiranje – primjeri za učenje, isključena semantika

	GP	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	160,33	224,61	263,62	187,37	265,04	291,67	248,45
Uwt	113,333	123,83	130,46	114,45	133,45	138,59	136,59
Fwt	236,73	220,01	295,53	201,87	403,07	402,42	248,63
Cmax	24,69	24,45	24,65	24,52	23,97	24,46	24,81

Tablica A.2 – Optimiranje – primjeri za učenje, uključena semantika

	GPS	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	161,43	224,61	263,62	187,37	265,04	291,67	248,45
Uwt	115,58	123,83	130,46	114,45	133,45	138,59	136,59
Fwt	197,54	220,01	295,53	201,87	403,07	402,42	248,63
Cmax	24,79	24,45	24,65	24,52	23,97	24,46	24,81

Tablica A.3 – Postoci dominacije – primjeri za učenje isključena semantika

	GP	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	80,0%	11,25%	0,0%	11,25%	3,75%	1,25%	5,63%
Uwt	62,5%	0,0%	35,0%	63,13%	34,38%	36,25%	36,25%
Fwt	31,25%	22,5%	6,25%	42,5%	3,13%	5,0%	5,63%
Cmax	10,63%	16,88%	14,38%	12,5%	49,38%	22,5%	10,63%

Tablica A.4 – Postoci dominacije – primjeri za učenje uključena semantika

	GPS	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	74,38%	15,63%	0,63%	16,25%	5,0%	1,25%	5,63%
Uwt	58,13%	0,0%	35,63%	68,75%	35,63%	36,88%	36,88%
Fwt	39,38%	23,75%	5,63%	34,38%	3,13%	5,0%	5,63%
Cmax	7,5%	17,5%	14,38%	13,75%	51,25%	21,88%	11,25%

Tablica A.5 – Optimiranje – Primjeri za ocjenu, isključena semantika

	GP	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	99,72	160,03	189,55	121,92	177,98	206,47	171,38
Uwt	98,96	112,56	122,59	101,32	124,09	132,55	127,54
Fwt	264,98	221,45	309,0	204,16	394,39	447,75	248,32
Cmax	24,35	24,24	24,54	24,14	23,69	24,1	24,93

Tablica A.6 – Optimiranje – Primjeri za ocjenu, uključena semantika

	GPS	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	100,1	160,03	189,55	121,92	177,98	206,47	171,38
Uwt	101,54	112,56	122,59	101,32	124,09	132,55	127,54
Fwt	202,76	221,45	309,0	204,16	394,39	447,75	248,32
Cmax	24,42	24,24	24,54	24,14	23,69	24,1	24,93

Tablica A.7 – Postoci dominacije – Primjeri za ocjenu, isključena semantika

	GP	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	74,38%	16,88%	1,25%	18,13%	3,75%	0,63%	5,0%
Uwt	37,5%	33,75%	0,0%	25,63%	1,25%	1,25%	3,75%
Fwt	13,13%	33,13%	5,0%	51,88%	5,63%	5,63%	5,63%
Cmax	12,5%	15,0%	11,88%	11,88%	45,0%	22,5%	8,75%

Tablica A.8 – Postoci dominacije – Primjeri za ocjenu, uključena semantika

	GPS	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	77,5%	16,88%	1,88%	17,5%	3,75%	0,625%	4,38%
Uwt	35,0%	30,63%	1,88%	31,88%	1,25%	1,88%	3,13%
Fwt	31,88%	28,75%	4,38%	36,25%	5,63%	5,63%	5,0%
Cmax	9,38%	15,0%	11,88%	11,88%	46,25%	23,75%	9,38%

Tablica A.9 – Optimiranje – Primjeri iz literature, isključena semantika

	GP	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	76,02	81,62	103,67	80,01	96,17	113,04	102,48
Uwt	69,11	70,04	72,34	69,19	74,17	76,73	76,51
Fwt	125,71	125,41	131,13	125,07	133,66	136,58	139,68
Cmax	8,34	8,31	8,44	8,33	8,09	8,3	8,49

Tablica A.10 – Optimiranje – Primjeri iz literature, uključena semantika

	GPS	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	76,31	81,62	103,67	80,01	96,17	113,04	102,48
Uwt	68,47	70,04	72,34	69,19	74,17	76,73	76,51
Fwt	125,71	125,41	131,13	125,07	133,66	136,58	139,68
Cmax	8,33	8,31	8,44	8,33	8,09	8,3	8,49

Tablica A.11 – Postoci dominacije – Primjeri iz literature, isključena semantika

	GP	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	88,13%	9,38%	0,0%	9,38%	1,88%	0,0%	1,25%
Uwt	36,88%	0,0%	26,25%	35,0%	24,38%	76,25%	24,38%
Fwt	65,63%	18,13%	5,0%	20,63%	2,5%	1,25%	2,5%
Cmax	8,75%	6,88%	3,75%	6,25%	75,63%	10%	5%

Tablica A.12 – Postoci dominacije – Primjeri iz literature, uključena semantika

	GPS	RM	COVERT	WSPT	WSPT/TWKR	SLACK/TWKR	EDD
Twt	83,75%	13,75%	0,0%	13,75%	2,5%	0,0%	1,25%
Uwt	36,88%	0,0%	26,88%	35,0%	24,38%	76,25%	24,38%
Fwt	18,75%	65,63%	5,0%	68,75%	2,5%	1,25%	1,88%
Cmax	8,13%	6,88%	3,75%	6,25%	75,0%	10,0%	5,0%