SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1482

Hiperheurističke metode rješavanja problema raspoređivanja u okruženju nesrodnih strojeva

Hrvoje Backović

Zagreb, lipanj 2017.

Zahvaljujem mentoru izv.prof.dr.sc. Domagoju Jakobovi\'cu i asistentu mag.ing. Marku Đurasevi\'cu na podršci i vodstvu tijekom izrade ovog rada.

SADRŽAJ

1. Uvod

Raspoređivanje se može definirati kao optimizacijski problem gdje je skup poslova potrebno rasporediti na dostupni skup resursa, pri čemu se minimiziraju određeni kriteriji (npr. kriterij da svi poslovi budu što prije gotovi) [2] [3]. Resursi mogu biti strojevi u tvornici, procesori u računalu, ljudi itd. Problem se može dodatno otežati uvođenjem raznih svojstava i ograničenja sustava (npr. ograničenje u redoslijedu izvođenja poslova). Većina poznatih problema raspoređivanja spada u NP-teške probleme, što znači da ne postoje efikasni algoritmi za njihovo rješavanje. S obzirom na tu činjenicu, u praksi se vrlo često koriste razne heuristike koje u razumnom vremenskom periodu pronalaze rješenje koje nije optimalno, ali je prihvatljivo. Te heuristike se prema načinu na koji pronalaze rješenje mogu podijeliti u one koje pretražuju prostor rješenja i one koje iterativno grade rješenje.

Heuristike koje pretražuju prostor rješenja se koriste kada su informacije o problemu unaprijed poznate i nepromjenjive, ali su obično vremenski vrlo zahtjevne pa se primjenjuju samo u statičkom okruženju raspoređivanja. Primjeri takvih postupaka su tabu pretraživanje, simulirano kaljenje, genetski algoritmi itd. S druge strane, heuristike koje iterativno grade rješenje su jednostavnije i brže pa se mogu koristiti u dinamičkom okruženju raspoređivanja ili u okruženju s promjenjivim parametrima, ali zato često daju lošije rezultate.

U heuristike koje iterativno izgrađuju rješenja spadaju i takozvana pravila raspoređivanja (engl. *dispatching rules*). Iako postoje mnoga pravila raspoređivanja koja za specifična okruženja i kriterije daju dobra rješenja, za mnoge probleme raspoređivanja još uvijek nisu razvijena prikladna pravila raspoređivanja. Zbog toga se u ovom radu opisuje hiperheuristički pristup gdje se uz pomoć metoda strojnog učenja dobivaju algoritmi koji su najpogodniji za raspoređivanje sustava s jedinstveno definiranim svojstvima. Dobiveni algoritmi koriste postupak prioritetnog raspoređivanja pri čemu se međusobno razlikuju po prioritetnoj funkciji, koja se koristi za određivanja redoslijeda raspoređivanja[6]. U ovom radu prikazane su metode kartezijskog genetskog

programiranja, gramatičke evolucije i neuronskih mreža. Ove metode su prilagođene tako da koriste mehanizme evolucijskih algoritama kako bi pronašle najbolju funkciju u prostoru prioritetnih funkcija, zato je za svaki postupak potrebno definirati odgovarajući prikaz rješenja te pripadne operatore križanja i mutacije. Cilj rada jest definirati i isprobati odabrane hiperheurističke metode i usporediti ih s postojećim metodama kako bi se ustanovilo je li njihovim korištenjem moguće izraditi kvalitetnija pravila raspoređivanja.

Rad je organiziran na sljedeći način: u 2. poglavlju je detaljno opisan problem raspoređivanja u okruženju nesrodnih strojeva s posebnim naglaskom na prioritetno raspoređivanje. Nakon toga su u poglavljima 3,4 i 5 slijedno opisani pristupi rješavanju tog problema koji koriste kartezijsko genetsko programiranje, gramatičku evoluciju i neuronske mreže. Za svaku metodu je naveden prikaz rješenja te operatori križanja i mutacije. U 6. poglavlju je opisano okruženje u kojem se provodilo testiranje te su izneseni i analizirani rezultati. Na samom kraju je dan kratak zaključak koji predlaže i moguća poboljšanja ovih pristupa.

2. Raspoređivanje u okruženju nesrodnih strojeva

Problem raspoređivanja u okruženju nesrodnih strojeva definiran je s n poslova i m strojeva, pri čemu je za svaki posao potrebno odrediti na koji će se stroj rasporediti, kao i redoslijed izvođenja poslova na svakom od strojeva [2]. Jednom kada je neki posao dodijeljen određenom stroju, on se na njemu izvodi do završetka, dakle nije ga moguće prekinuti i rasporediti na drugi stroj. Također, svaki stroj može u nekom trenutku izvoditi samo jedan posao.

Svakom poslu dodijeljen je indeks iz intervala $[0, n-1]$, dok je svakom stroju dodijeljen indeks iz intervala $[0, m-1]$. Prilikom raspoređivanja na raspolaganju su svojstva specifična za sam problem i okruženje. Ta svojstva su nabrojana i opisana u tablici 2.1. Kada se određuje raspored poslova po strojevima, s obzirom na navedena svojstva, optimira se jedan od kriterija raspoređivanja opisanih u nastavku.

Oznaka	Opis
p_{ij}	vrijeme izvođenja posla j na stroju i
r_j	vremenski trenutak kada posao j dolazi u sustav
d_j	rok, vremenski trenutak kada posao j treba biti gotov, u suprotnom dolazi do određenog gubitka zbog kašnjenja
w_j	težina, važnost posla

Tablica 2.1: Svojstva problema raspoređivanja

Oznaka	Opis
C_j	vremenski trenutak kada posao j završi s izvođenjem
F_j	tok, vrijeme koje je posao j proveo u sustavu. Definira se kao $F_j = C_j - r_j$
T_j	zakašnjelost, vrijeme koliko se posao j izvršavao nakon roka $T_j = max(C_j - d_j, 0)$
U_j	je li posao j zakasnio, $U_j = \begin{cases} 1 & \text{if } T_j > 0 \\ 0 & \text{if } T_j = 0 \end{cases}$

Tablica 2.2: Kriteriji za pojedinačne poslove

Oznaka	Opis
$Cmax$	maksimalno vrijeme završetka izvođenja. $Cmax = max(Cj)$
Ft	ukupni tok. $Ft = \sum_{j=1}^{n} F_j$
Twt	ukupna težinska zakašnjelost. $Twt = \sum_{j=1}^{n} w_j T_j$
Uwt	težinski broj zakašnjelih poslova. $Uwt = \sum_{j=1}^{n} w_j U_j$

Tablica 2.3: Kriteriji po kojima se provodi optimizacija

2.1. Kriteriji raspoređivanja

Kvalitetu nekog rasporeda možemo definirati preko brojnih kriterija raspoređivanja. Navedeni kriteriji se zatim mogu koristiti kao funkcija dobrote (engl. *fitness function*) u genetskom algoritmu i po njima se može optimirati rješenje. Kako bi se jednostavnije iskazali kriteriji za čitav raspored, u tablici 2.2 su prikazani kriteriji za pojedinačne poslove, koji se koriste pri definiranju kriterija za cjelokupni raspored [3]. U tablici 2.3 navedena su 4 glavna kriterija koji će se optimirati u ovom radu.

2.2. Pristupi rješavanju problema raspoređivanja

Problem raspoređivanja može se podijeliti u nekoliko klasa s obzirom na pojedina svojstva problema. Na primjer, ako su sve informacije o problemu poznate odmah na početku izvođenja, onda se takav problem naziva *offline* raspoređivanje. S druge strane,

problem se može zadati na način da informacije o pojedinom poslu postaju dostupne tek kada taj posao uđe u sustav. U tom slučaju se taj problem naziva *online* raspoređivanje. Nadalje, problem raspoređivanja možemo podijeliti i po načinu na koji se konstruira rješenje. Ako se cjelokupno rješenje konstruira prije početka izvođenja sustava, onda se to naziva statičko raspoređivanje. Kada se rješenje konstruira paralelno s izvođenjem sustava, tada se to naziva dinamičko raspoređivanje.

U ovom radu koncentriramo se na *online* dinamičko raspoređivanje, koje je najzastupljenije u problemima raspoređivanja iz stvarnosti. Često kod problema raspoređivanja nisu poznate sve informacije o sustavu te je potrebno donositi odluke za vrijeme izvođenja sustava te se prilagođavati na nagle promjene u sustavu. Algoritmi koji rješavaju ovakve probleme moraju imati kratko vrijeme izvođenja i spadaju u skupinu heuristika koje grade rješenje iterativno. Takvi algoritmi najčešće su oblikovani na način da reagiraju na događaje u sustavu i kao izlaz daju sljedeće stanje sustava (npr. pridruživanje zadatka procesoru). U nastavku je opisano prioritetno raspoređivanje u kojem je detaljnije opisan način na koji se donosi odluka o sljedećem stanju.

2.3. Pravila raspoređivanja

Raspoređivanje uz pomoć pravila ili prioritetno raspoređivanje je oblik raspoređivanja gdje se raspored dinamički gradi za vrijeme izvođenja sustava na način da se u ključnim trenucima donosi odluka o sljedećem stanju sustava. Ovakvo raspoređivanje koristi se u uvjetima raspoređivanja u stvarnom vremenu gdje se sustav mijenja s vremenom te je potrebno brzo reagirati i donijeti odluku. Izlaz takvih algoritama nije cjelokupni raspored od početka rada sustava do njegovog eventualnog završetka već samo informacija uz pomoć koje sustav prelazi u sljedeće stanje. Opisani način primjene prioritetnog raspoređivanja u dinamičkoj okolini prikazan je algoritmom 1.

Algoritam 1: Postupak raspoređivanja po pravilima za dinamičke uvjete

```
1 postoje neraspoređeni poslovi čekaj dok stroj ne postane slobodan;
2 odredi prioritete svih dostupnih poslova;
3 odaberi posao s najboljim prioritetom;
4 rasporedi posao s najboljim prioritetom na stroj;
```

Pravila raspoređivanja se obično opisuju funkcijom koja za dane ulazne elemente sustava, na izlazu daje numeričku vrijednost koja označava prioritet. Ulaz u prioritetnu

funkciju je svaki par posao/stroj koji su dostupni u tom trenutku te se računa njihov prioritet. Na kraju se odabire par s najmanjim prioritetom te se provodi pridruživanje posla stroju. U ovom radu se ulazi u prioritetnu funkciju oblikuju kao 9 terminala koji opisuju razne karakteristike poslova i čitavog sustava. Terminali koji se koriste kao ulazi prikazani su u tablici 2.4.

Oznaka	Opis
pt	vrijeme izvođenja posla j na stroju i (p_{ij})
$pmin$	minimalno vrijeme izvođenja na svim strojevima
$pavg$	srednje vrijeme izvođenja na svim strojevima
PAT	strpljivost - vrijeme dok stroj s minimalnim vremenom izvođenja ne postane slobodan
MR	vrijeme dok trenutni stroj ne postane slobodan
age	vrijeme koje je posao proveo u sustavu
dd	vremenski trenutak kad posao mora završiti (d_j)
w	težina (prioritet) posla
SL	dopuštena odgoda uz brzinu dotičnog stroja - $max(d_j - p_{ij} - time, 0)$

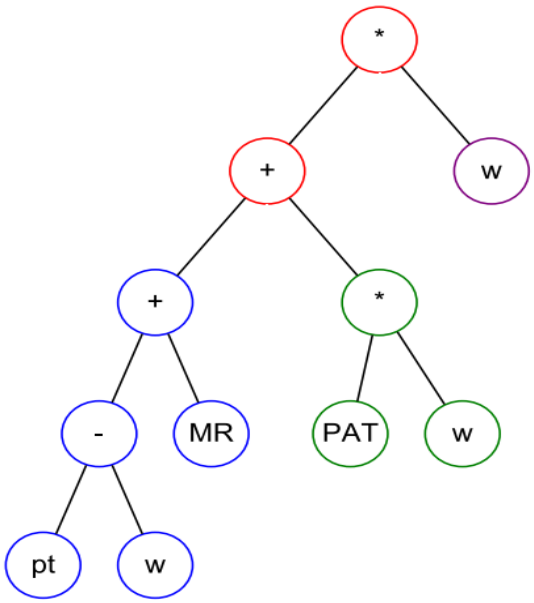
Tablica 2.4: Ulazni terminali prioritetne funkcije

Prednost ovakvog pristupa raspoređivanju je brzina izvođenja i jednostavnost primjene. Pravila koja se koriste su uglavnom jednostavna za implementaciju i razumljiva su za čovjeka jer prate ljudski način razmišljanja. Također je kod prioritetnog raspoređivanja dobro to što se vrlo brzo prilagodi na promjene u sustavu jer ih odmah u sljedećem koraku može uzeti u obzir pri računanju prioriteta. Prioritetno raspoređivanje je fleksibilno te se može koristiti i u statičkom okruženju s offline načinom raspoređivanja. Loša strana raspoređivanja uz pomoć pravila je kvaliteta rezultata koja je često lošija od dugotrajnijih statičkih postupaka temeljenih na pretraživanju stanja, ali ti postupci ponekad nisu primjenjivi. U nastavku je dan pregled postupaka koji uz pomoć evolucijskih algoritama i neuroloških sustava pronalaze odgovarajuće prioritetne funkcije koje zatim možemo koristiti u prioritetnom raspoređivanju.

3. Kartezijsko genetsko programiranje

3.1. Genetsko programiranje

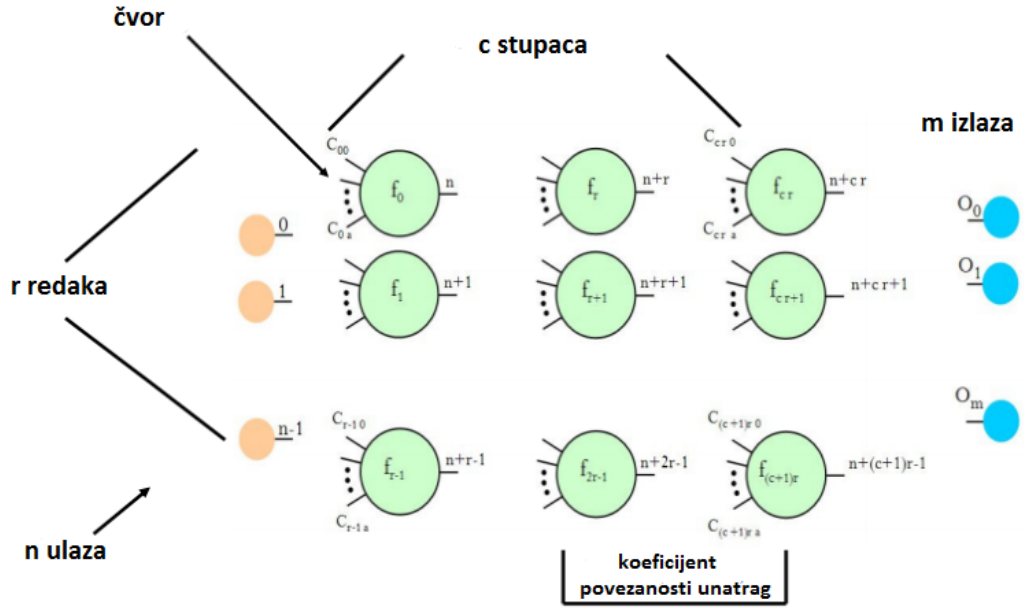
Genetsko programiranje je optimizacijski postupak koji stvara programe ili funkcije za rješavanje nekog konkretnog problema. Koristeći evolucijske algoritme, pretražuje se prostor računalnih programa kako bi se pronašao prikladan program koji daje najbolje rješenje danog problema. Jedna od najčešćih primjena genetskog programiranja je simbolička regresija gdje je potrebno simulirati nepoznatu funkciju tj. pronaći matematički izraz koji opisuje dan skup empirijskih podataka. U genetskom programiranju programi ili funkcije se kodiraju u obliku stabla gdje listovi predstavljaju ulazne parametre, a unutarnji čvorovi predstavljaju matematičke operacije. Primjer jedne funkcije prikazan je na slici 3.1.



Slika 3.1: Funkcija zapisana u obliku stabla. $f = (((pt - w) + MR) + (PAT \cdot w)) \cdot w$

3.2. Struktura genotipa

Kartezijsko genetsko programiranje je fleksibilna i efikasna varijanta genetskog programiranja gdje se programi kodiraju kao Kartezijev produkt jednostavnih funkcija koje su međusobno povezane[8]. Kartezijsko genetsko programiranje osmislio je Julian Miller 1999. godine. Struktura genotipa prikazana je na slici 3.2. Primijetimo da struktura više nije nužno stablo već graf kojem je čak moguće definirati više izlaza. Unutarnji čvorovi koji predstavljaju funkcije složeni su u matricu kojoj unaprijed definiramo broj redaka i stupaca. Svi unutarnji čvorovi imaju unaprijed definiran broj ulaza. Funkcije kao ulaze mogu uzimati samo čvorove koji se nalaze u prethodnim stupcima s time da se može dodatno definirati parametar koji određuje maksimalnu udaljenost indeksu stupca iz koje funkcija smije uzimati podatke za svoje ulaze (koeficijent povezanosti unatrag).

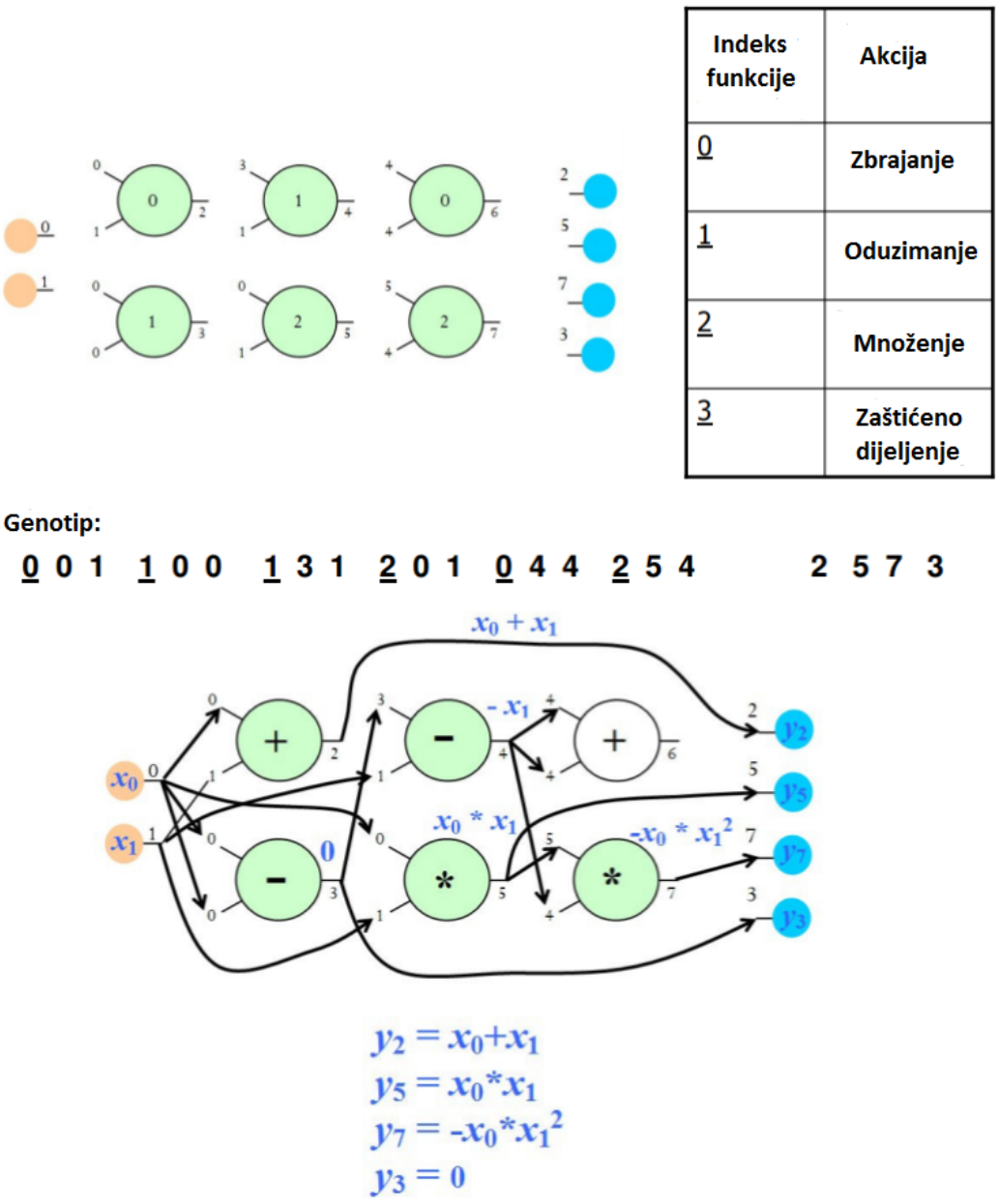


Slika 3.2: Struktura genotipa [9]

Potrebno je primijetiti da je veličina genotipa fiksna i definira se unaprijed prije izvođenja evolucijskog algoritma. Ovime se sprječava poznati problem napuhavanja (engl. *bloat*) rješenja koji se često javlja kod genetskog programiranja zbog varijabilne veličine genotipa. Napuhavanje rezultira velikim funkcijama koje su nepregledne i računalno složenije te često sadrže nepotrebne redundantne dijelove koje ne doprinose značajno kvaliteti čitavog rješenja.

3.3. Prikaz rješenja

Kako bi mogli primijeniti evolucijske operatore na genotipu, potrebno je definirati kvalitetan prikaz rješenja. Genotip kod kartezijskog genetskog programiranja definira se kao niz cijelih brojeva fiksne veličine. Brojevi u prikazu definiraju tip funkcija koje se izvršavaju u pojedinom unutarnjem čvoru te veze između njih. Primjer takvog prikaza rješenja prikazan je na slici 3.3.



Slika 3.3: Prikaz rješenja [9]

U primjeru se koriste 2 ulaza, 4 izlaza i 6 unutarnjih čvorova. Svi ulazi i unutarnji čvorovi su numerirani. Svi unutarnji čvorovi imaju po dva ulaza. Genotip takve strukture sastoji se od 3 vrijednosti za svaki unutarnji čvor nakon čega slijede vrijednosti za svaki izlaz. Za svaki unutarnji čvor definira se tip funkcije te indeksi čvorova koji se koriste kao ulazi u čvor, a za svaki izlaz definira se čvor koji je izravno spojen na njega.

Izraz za svaki izlaz može se efikasno izračunati prolaskom kroz graf počevši od izlaznih čvorova i prateći njihove veze. Time se osigurava iteriranje samo po aktivnim čvorovima dok se čvorovi koji nisu povezani s izlazima preskaču.

3.4. Parametri algoritma

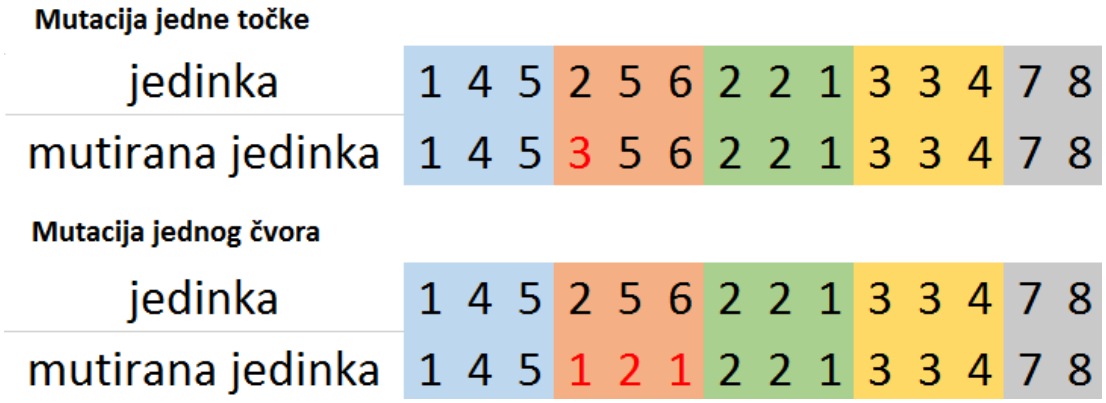
Iz definicije strukture genotipa kartezijskog genetskog programiranja može se vidjeti veliki broj parametara koje je potrebno odrediti unaprijed i koji imaju veliki utjecaj na performanse algoritma. Kako bi se lakše provela optimizacija parametara, u ovom radu se koriste preporučene vrijednosti određenih parametara[8]. U nastavku koristimo strukturu koja ima samo jedan redak, ali veći broj stupaca. Također se za koeficijent povezanosti unatrag koristi broj stupaca tako da svaki čvor za ulaze može koristiti sve prethodne čvorove. Broj ulaza u sustav je jednak broju terminala definiranih u tablici 2.4, a koristi se jedan izlaz s obzirom da se traži prioritetna funkcija koja ima samo jedan izlaz. Funkcije unutarnjih čvorova definirane su u tablici 3.1. Po uzoru na ove funkcije, koriste se dva ulaza za svaki unutarnji čvor.

Oznaka	Opis
$+$	operator binarnog zbrajanja
$-$	operator binarnog oduzimanja
$*$	operator binarnog množenja
$/$	sigurno dijeljenje: $/(a,b) = \begin{cases} 1 & \text{if } \lvert b\rvert < 0.000001 \\ \text{a / b} & \text{else} \end{cases}$
POS	unarni operator: $POS(a) = max(a, 0)$

Tablica 3.1: Funkcije unutarnjih čvorova

3.5. Evolucijski operatori

Kako bi evolucijski algoritam bio kvalitetan, potrebno je odabrati dobre operatore križanja i mutacije. Operatori koji se koriste moraju brinuti o tome da se ne prekrše neka od pravila strukture genotipa (npr. unutarnji čvor ne smije kao ulaz uzeti čvor koji nije u rasponu koji određuje koeficijent povezanosti unatrag). Za mutaciju se koristi mutacija jedne točke i mutacija jednog unutarnjeg čvora, čije je djelovanje prikazano na slici 3.4. Operatori križanja koji se koriste su križanje s jednom točkom prekida na cijelom polju i križanje s jednom točkom prekida na razini unutarnjih čvorova. Prikaz ovih operatora nalazi se na slici 3.5.



Slika 3.4: Operatori mutacije

Slika 3.5: Operatori križanja

4. Gramatička evolucija

Gramatička evolucija (engl. *grammatical evolution*) je evolucijski algoritam pretraživanja, sličan genetskom programiranju. Najčešće se koristi kako bi se generirali programi čija je sintaksa zadana gramatikom[7]. Koristeći evolucijski algoritam (npr. genetski algoritam), pretražuje se prostor svih programa koje je moguće definirati zadanom gramatikom kako bi se pronašao program koji daje najbolje rezultate za neki problem. U ovom radu uz pomoć gramatičke evolucije traži se najbolja prioritetna funkcija za dani problem raspoređivanja.

4.1. Gramatika prioritetnih funkcija

Gramatička evolucija koristi kontekstno-slobodnu gramatiku kako bi opisala sintaksu programa. Takva gramatika je neovisna o kontekstu rečenice i definira se skupom završnih simbola, nezavršnih simbola, produkcijskih pravila te početnim simbolom. Gramatičku evoluciju osmislili su Conor Ryan, JJ Collins i Michael O'Neill 1998. godine. Na slici 4.1 prikazana je gramatika koja opisuje prioritetnu funkciju u BNF (Backus-Naurova forma) obliku. Prvi redak prikazuje početni nezavršni simbol, a nakon njega slijede produkcijska pravila. Završni simboli su zapisani u uglatim zagradama. Iz gramatike se može vidjeti skup korištenih funkcija koji odgovara onima iz tablice 3.1. Također, ulazni argumenti funkcija odgovaraju terminalima iz tablice 2.4.

```
<expr>
<expr> : <expr> <op> <expr> | <subexpr> | ( <expr> )
<subexpr> : <func> ( <expr> ) | <var>
<func> : [pos]
<op> : + | - | * | /
<var> : [T_pt] | [T_dd] | [T_w] | [T_SL] | [T_pmin] | [T_pavg] | [T_PAT] | [T_MR] | [T_age]
```

Slika 4.1: Gramatika za generiranje prioritetnih funkcija

4.2. Prikaz rješenja

Za korištenje evolucijskog algoritma potrebno je definirati prikaz rješenja koji će biti efikasan i prilagođen za evolucijske operatore križanja i mutacije. U ovom radu se prioritetne funkcije mapiraju u niz prirodnih brojeva fiksne veličine koji opisuje proces kojim se iz gramatike generira određena prioritetna funkcija. Primjer takvog prikaza nalazi se na slici 4.2. Dobiveni izraz se uz pomoć poznatog algoritma skretničkog kolosijeka (engl. *shunting-yard algorithm*) pretvori u prefiksni sustav oznaka (poljsku notaciju) koji je pogodan za brzo izračunavanje rezultata funkcije[12].

prikaz	izraz
	<expr>
0	<expr> <op> <expr>
7	<subexpr> <op> <expr>
5	<var> <op> <expr>
1	[T_dd] <op> <expr>
4	[T_dd] + <expr>
4	[T_dd] + <subexpr>
2	[T_dd] + <func> (<expr>)
3	[T_dd] + pos (<expr>)
0	[T_dd] + pos (<expr> <op> <expr>)
1	[T_dd] + pos (<subexpr> <op> <expr>)
3	[T_dd] + pos (<var> <op> <expr>)
0	[T_dd] + pos ([T_pt] <op> <expr>)
1	[T_dd] + pos ([T_pt] - <expr>)
7	[T_dd] + pos ([T_pt] - <subexpr>)
5	[T_dd] + pos ([T_pt] - <var>)
3	[T_dd] + pos ([T_pt] - [T_SL])

Slika 4.2: Prikaz rješenja

Algoritam mapiranja kreće od početnog nezavršnog simbola i primjenjuje produkcijska pravila kako bi došao do završnog matematičkog izraza. Brojevi u prikazu rješenja čitaju se slijedno te određuju indeks produkcijskog pravila koje će se primijeniti na najlijevijem nezavršnom simbolu u izrazu. U slučaju kad indeks iz prikaza premašuje broj produkcijskih pravila za neki nezavršni simbol, uzima se ostatak pri dijeljenju s brojem produkcijskih pravila. Ako se nakon prolaska kroz čitav izraz ne dobije izraz koji sadrži samo završne simbole, tada se ta jedinka proglašava nevažećom te joj se daje najgora ocjena dobrote.

4.3. Evolucijski operatori

Zbog jednostavnosti prikaza rješenja, moguće je koristiti postojeće evolucijske operatore križanja i mutacije nad nizom prirodnih brojeva. Za mutaciju se koristi mutacija jedne točke koja je prikazana na slici 4.3. U prikazu se koristi maksimalna vrijednost 8 iz razloga što u gramatici nema nezavršnog simbola s više od 9 produkcijskih pravila, čime se ponešto pojednostavljuje prikaz. Kao operator križanja koristi se slučajno križanje gdje se svaki broj u prikazu rješenja djeteta uzima nasumično od jednog ili drugog roditelja. Primjer ovakvog križanja vidljiv je na slici 4.4.

jedinka	1	0	3	4	7	5	8	1	2
mutirana jedinka	1	0	3	4	7	4	8	1	2

Slika 4.3: Mutacija jedne točke

1. roditelj	6	6	4	2	7	3	1	3	4
2. roditelj	1	0	8	8	0	0	2	4	5
dijete	1	0	4	8	7	3	1	4	4

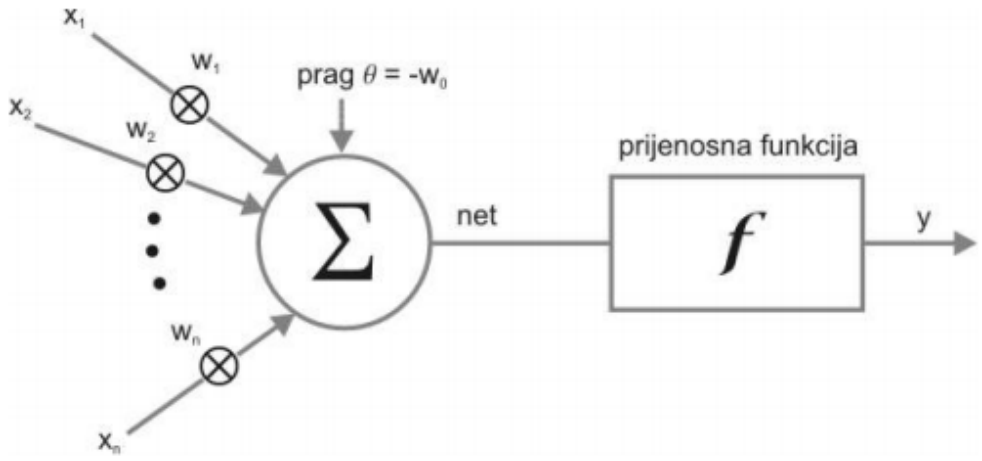
Slika 4.4: Nasumično križanje

5. Neuronske mreže

Umjetne neuronske mreže su matematički model u računarskoj znanosti kojeg su definirali McCulloch i Pitts 1940. godine i koji je baziran na radu neuroloških sustava u živim bićima. Model pronalazi brojne primjene u umjetnoj inteligenciji i strojnom učenju te u današnje vrijeme predstavlja nezaobilazan koncept u razvoju inteligentnih sustava[11]. Jedna od glavnih primjena umjetnih neuronskih mreža je aproksimacija funkcija (regresija) što je pogodno za hiperheuristički model opisan u ovom radu gdje je cilj pronaći odgovarajuću prioritetnu funkciju za dani problem raspoređivanja. Učenje neuronske mreže je u ovom radu ostvareno uz pomoć genetskog algoritma pa je za mrežu potrebno definirati odgovarajući prikaz rješenja i evolucijske operatore [10].

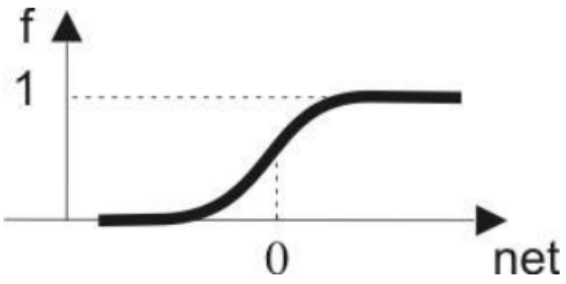
5.1. Struktura neuronske mreže

Umjetna neuronska mreža se sastoji od većeg broja jednostavnih jedinica (neurona) koje su međusobno povezane. Struktura neurona može se vidjeti na slici 5.1. Svaki neuron je definiran vektorom težina (wi) i prijenosnom funkcijom. Izlaz neurona se računa po formuli $y = f(\sum_{j=1}^{n} w_j x_j + w_0)$. Prijenosne funkcije koje se koriste u ovom radu su sigmoidalna (prikazana na slici 5.2) i linearna.



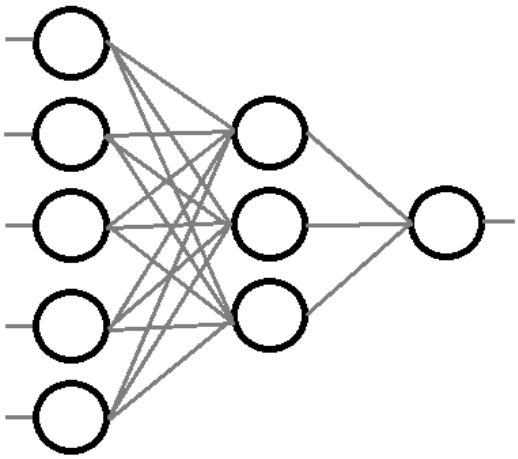
Slika 5.1: Građa neurona

$$f(net) = \frac{1}{1 + e^{-a \cdot net}}$$



Slika 5.2: Sigmoidalna prijenosna funkcija

Međusobnim povezivanjem neurona dobiva se neuronska mreža. Postoji više vrsta neuronskih mreža, a u ovom radu se koriste potpuno povezane unaprijedne neuronske mreže. Kod takvih neuronskih mreža neuroni su složeni po slojevima gdje su svi neuroni jednog sloja povezani sa svakim neuronom iz sljedećeg sloja. Primjer takve mreže prikazan je na slici 5.3. Neuroni u prvom sloju se zovu ulazni neuroni koji nemaju težine već samo prosljeđuju ulaze prema neuronima iz sljedećeg sloja. S druge strane, neuroni u zadnjem sloju su izlazni neuroni. Oni imaju težine, ali im je prijenosna funkcija linearna. Svi ostali neuroni se nazivaju skriveni neuroni koji imaju definirane težine te koriste sigmoidalnu prijenosnu funkciju.

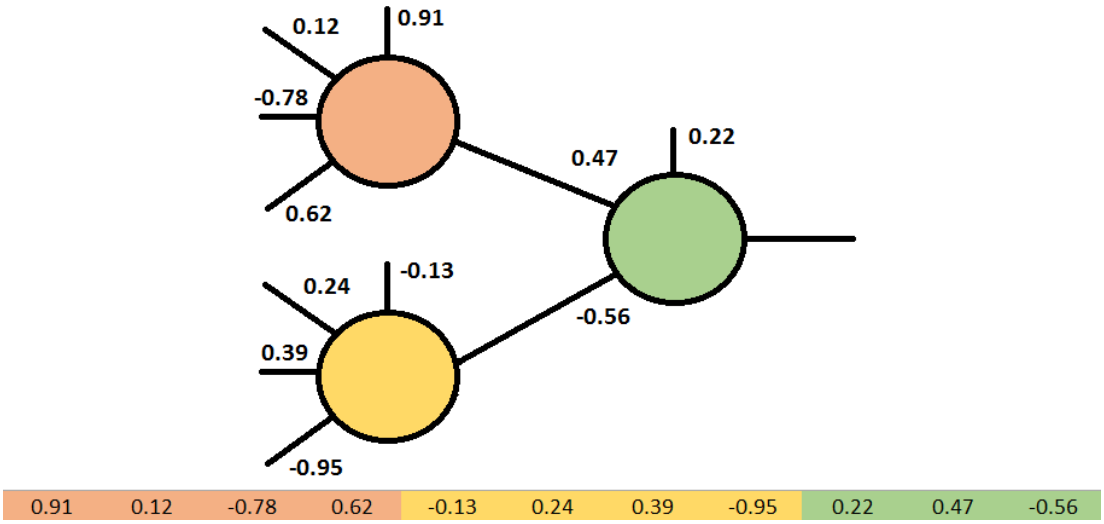


Slika 5.3: Potpuno povezana unaprijedna mreža

Kako bi se mreža prilagodila prioritetnim funkcijama u raspoređivanju, koristi se 9 ulaznih neurona koji odgovaraju terminalima iz tablice 2.4. U izlaznom sloju se koristi jedan neuron koji označava prioritet, a za skrivene neurone testirat će se više različitih struktura.

5.2. Prikaz rješenja

U ovom radu za neuronske mreže koristi se prikaz rješenja u obliku niza decimalnih brojeva. S obzirom da je neurone moguće opisati nizom decimalnih brojeva koji označavaju težine, svi neuroni mreže mogu se slijedno zapisati u jedan niz čime se dobiva jednostavan i efikasan zapis neuronske mreže. Ako je unaprijed poznata struktura mreže i redoslijed zapisivanja neurona, moguće je iz niza decimalnih brojeva jednostavno rekonstruirati cijelu mrežu. Primjer mapiranja neuronske mreže u niz decimalnih brojeva prikazan je na slici 5.4.



Slika 5.4: Prikaz rješenja

5.3. Evolucijski operatori

Kako bi efikasno učili umjetnu neuronsku mrežu uz pomoć genetskog algoritma, potrebno je definirati operatore križanja i mutacije. Za prethodno definirani prikaz neuronske mreže koriste se sljedeći operatori križanja:

- **križanje slojeva** - nova jedinka nasumično odabire cijele slojeve od jednog ili drugog roditelja
- **križanje neurona** - nova jedinka svaki neuron nasumično odabire od jednog ili drugog roditelja
- **uniformno križanje neurona** - slično kao križanje neurona, ali je broj neurona odabranih od jednog roditelja uvijek približno jednak broju neurona uzetih od drugog roditelja

Korišteni operatori mutacije su:

- **skaliranje težine** - nasumično odabrana težina se množi s nasumičnim faktorom uzetim iz intervala [0.5, 1.5]
- **skaliranje neurona** - provođenje skaliranja težine nad svim težinama nasumično odabranog neurona
- **mutiranje neurona** - nasumično odabranom neuronu se sve težine mijenjaju u slučajno odabrane vrijednosti iz dozvoljenog intervala
- **mutiranje N neurona** - provođenje mutiranja neurona nad N nasumično odabranih neurona pri čemu je N također slučajno odabran.

– **Gaussovo mutiranje težine** - nasumično odabrana težina se mijenja tako da joj se pridoda vrijednost iz normalne razdiobe oko nule sa standardnom devijacijom jednakoj desetini vrijednosti gornje granice dozvoljenog raspona.

Za neuronsku mrežu je moguće postaviti gornju i donju granicu težina pa se operatori s obzirom na to brinu da vrijednosti težina ne izlaze van iz danog intervala. U slučaju kad neka težina izađe iz dozvoljenog raspona, njena se vrijednost promijeni u najveću odnosno najmanju dozvoljenu vrijednost.

6. Rezultati i analiza

U ovom poglavlju opisano je okruženje u kojem se provodilo testiranje te na koji način su se odabrali parametri algoritama kako bi se dobili što bolji rezultati. Nakon toga se iznose rezultati testiranja nad opisanim metodama. Uvodimo oznake CGP za kartezijsko genetsko programiranje, GE za gramatičku evoluciju te NN za neuronske mreže. Dobiveni rezultati se uspoređuju s onima iz [1] koje daju deterministične heuristike, metoda genetskog programiranja i GEP.

6.1. Okruženje za testiranje i implementacija

S obzirom da se u ovom radu uvelike oslanjamo na rezultate iz [1], testiranje provodimo na jednakom skupu primjera i to na isti način. Skup primjera se sastoji od 120 instanci problema raspoređivanja. Ovisno o konkretnoj instanci problema, ukupni broj poslova može iznositi između 12 i 100, dok broj raspoloživih strojeva može iznositi između 3 i 10. Opisani skup primjera podijeljen je na dva odvojena skupa: skup za učenje i skup za validaciju. Skup za učenje korišten je za vrijeme izvođenja evolucijskog algoritma, dok je skup za validaciju korišten za ispitivanje kvalitete dobivenih rješenja nakon završetka algoritma. Oba skupa sadrže po 60 instanci problema. Svi prikazani rezultati izmjereni su nad skupom za validaciju. Testiranje se provodi na način da se svaki eksperiment pokrene 30 puta nad svim problemima za učenje te se u svakom pokretanju odabire najbolje dobiveno rješenje. Za svaki eksperiment prikazuje se minimalna vrijednost dobrote, maksimalna vrijednost dobrote i medijan dobrote. Implementacija samih algoritama napisana je u jeziku C++ koristeći programsko okruženje ECF (Evolutionary Computation Framework [5]) koje se koristi za implementaciju raznih postupaka evolucijskog računanja.

6.2. Optimiziranje parametara

Kvaliteta rezultata koje daju opisani pristupi uvelike ovisi o odabranim vrijednostima parametara pa je zato potrebno u fazi testiranja pronaći optimalan skup vrijednosti parametara. U nastavku se opisuje optimiziranje parametara za svaki od tri pristupa opisanih u ovom radu. Za optimiranje parametara koristi se Twt kriterij dok se rezultati za ostale kriterije izračunavaju samo za najbolji dobiveni skup vrijednosti parametara. Kod testiranja kriterija Ft i Cmax izbacuju se terminali dd, w i SL jer u tim slučajevima nisu relevantni.

6.2.1. Parametri evolucijskog algoritma

Kao bazni algoritam za tri opisana pristupa u ovom radu koristi se genetski algoritam. To je stohastički optimizacijski postupak koji se može primijeniti na razne optimizacijske probleme te se u današnje vrijeme intenzivno koristi [4]. Za prethodno opisane prikaze rješenja i operatore križanja i mutacije, algoritam provodi selekciju, rekombinaciju i mutaciju nad populacijom potencijalnih rješenja. Populacija prolazi kroz određeni broj generacija dok se naposljetku ne dobije kvalitetno rješenje. U ovom radu koristi se k-turnirska eliminacijska selekcija ($k = 3$) pri čemu se za kriterij zaustavljanja koristi 80000 evaluacija. K-turnirska selekcija opisana je algoritmom 2.

Algoritam 2: K-turnirska selekcija

1 zaberi slučajno k jedinki;
2 Ukloni najgoreg od k izabranih;
3 Dijete = križaj(najbolja dva od k izabranih);
4 S određenom vjerojatnošću mutiraj dijete;
5 Ubaci dijete u populaciju;

Osim genetskog algoritma, kod kartezijskog genetskog programiranja isprobana je i evolucijska strategija. Evolucijska strategija također spada u skupinu evolucijskih algoritama koji uz pomoć selekcije, križanja i mutacije traži najbolju jedinku u populaciji mogućih rješenja. Postupak evolucijske strategije opisan je algoritmom 3. Ovaj algoritam je neovisan o genotipu što znači da ga možemo primijeniti na prethodno definiran prikaz rješenja. U ovom radu koristi se $(1 + 4)$ evolucijska strategija ($\mu = 1, \rho = 1, \lambda = 4, selekcija = +$) s kriterijem zaustavljanja od 80000 evaluacija.

Algoritam 3: Jedna generacija evolucijske strategije

```
podpopulacija Dodaj μ jedinki u skup roditelja;
Generiraj λ djece koristeći slučajnih ρ roditelja za svako dijete;
if plus selekcija then
    Stvori novi skup roditelja s μ najboljih jedinki iz skupa djece i roditelja;
end
else
    Stvori novi skup roditelja s μ najboljih jedinki iz skupa djece;
end
```

Osim spomenutih parametara, vjerojatnost mutacije i veličina početne populacije će se u nastavku zasebno optimizirati s parametrima genotipa.

6.2.2. Parametri kartezijskog genetskog programiranja

Kao što je opisano u poglavlju 3, kod kartezijskog genetskog programiranja postoje brojni parametri koji se mogu optimirati. Koristeći savjete iz literature, taj broj parametara je smanjen. Na kraju je potrebno optimirati broj stupaca u genotipu C, veličinu populacije P te vjerojatnost mutacije p_m. Koristeći pretraživanje po rešetci (engl. *grid search*), testiraju se parovi veličina broja stupaca i vjerojatnosti mutacije dok veličina populacije ostaje konstantna. Tablica 6.1 prikazuje optimiranje parametara za k-turnirsku selekciju dok tablica 6.2 prikazuje optimiranje parametara za evolucijsku strategiju.

$C \backslash p_m$	0.3	0.5	0.7	0.9
100	13.38,**13.81**,**15.42**	**12.74**,14.21,16.28	13.13,13.84,15.60	13.53,14.92,17.03
300	13.50,14.18,15.63	13.30,14.17,15.59	13.69,14.52,16.40	13.17,14.91,16.74
500	12.75,14.24,16.52	13.43,14.31,16.36	13.24,14.37,17.08	13.34,14.85,17.09

Tablica 6.1: Rezultati testiranja za k-turnirsku selekciju uz $P = 500$. Svaka ćelija sadrži minimum,medijan i maksimum dobrote.

$C \backslash p_m$	0.3	0.5	0.7	0.9
100	13.57,14.09,15.56	13.32,14.29,15.83	13.53,14.19,15.37	13.44,**14.09**,**15.24**
300	13.51,14.14,16.10	13.30,14.35,15.73	**13.17**,14.21,15.99	13.38,14.21,15.79
500	13.23,14.33,16.66	13.28,14.33,15.49	13.53,14.45,15.84	13.34,14.48,15.79

Tablica 6.2: Rezultati testiranja za evolucijsku strategiju uz $P = 5$. Svaka ćelija sadrži minimum,medijan i maksimum dobrote.

Koristeći rezultate iz tablica 6.1 i 6.2, uzima se skup parametara koji ima najmanju vrijednost medijana dobrote te se s njim optimira veličina populacije. Tablice 6.3 i 6.4 prikazuju rezultate optimiranja veličine populacije za k-turnirsku selekciju i evolucijsku strategiju.

P	minimum	medijan	maksimum
500	13.38	**13.81**	**15.42**
1000	**12.90**	14.15	15.78

Tablica 6.3: Rezultati testiranja različitih veličina populacije za k-turnirsku selekciju uz $C = 100$ i $p_m = 0.3$

P	minimum	medijan	maksimum
5	13.44	**14.09**	**15.24**
20	**13.08**	14.29	16.07
50	13.33	14.23	15.63

Tablica 6.4: Rezultati testiranja različitih veličina populacije za evolucijsku strategiju uz $C = 100$ i $p_m = 0.9$

Za odabir najbolje konfiguracije uzima se u obzir medijan dobrote kao najstabilniji pokazatelj kvalitete. Koristeći najbolje skupove parametara iz prethodnih tablica, računaju se vrijednosti za sve kriterije raspoređivanja. Rezultati za k-turnirsku selekciju prikazani su u tablici 6.5 dok su za evolucijsku strategiju rezultati prikazani u tablici 6.6. Iz rezultata se vidi da je k-turnirska selekcija postigla bolje rezultate od evolucijske strategije za sve kriterije.

kriterij	minimum	medijan	maksimum
Cmax	37.88	38.27	38.71
Ft	153.72	154.93	158.62
Uwt	6.61	7.23	7.67
Twt	13.38	13.81	15.42

Tablica 6.5: Rezultati testiranja kartezijskog genetskog programiranja s k-turnirskom selekcijom nad svim kriterijima raspoređivanja uz C = 100, p_m = 0.3 i P = 500

kriterij	minimum	medijan	maksimum
Cmax	37.90	38.39	38.71
Ft	153.75	155.06	156.48
Uwt	6.77	7.19	7.42
Twt	13.44	14.09	15.24

Tablica 6.6: Rezultati testiranja kartezijskog genetskog programiranja s evolucijskom strategijom nad svim kriterijima raspoređivanja uz C = 100, p_m = 0.9 i P = 5

6.2.3. Parametri gramatičke evolucije

Parametar prikaza rješenja gramatičke evolucije koji se optimizira je veličina genotipa jer je prikaz definiran preko niza prirodnih brojeva fiksne veličine. Uz ovaj parametar, na performanse najviše utječe i vjerojatnost mutacije i veličina početne populacije. Kako bi se ovi parametri optimizirali, koristi se pretraživanje po rešetci gdje se isprobava više kombinacija veličine genotipa N i vjerojatnosti mutacije p_m, dok je veličina populacije jednaka za sve i iznosi 500. Rezultati ovog testa nalaze se u tablici 6.7.

$N \backslash p_m$	0.3	0.5	0.7	0.9
30	13.40,17.26,27.36	13.40,17.07,27.36	14.65,19.98,27.36	14.53,17.64,27.36
50	13.40,16.49,22.21	13.40,16.89,27.36	13.11,15.38,22.69	13.40,15.42,27.36
70	13.13,15.15,27.36	13.58,15.67,20.11	13.58,15.06,20.16	13.40,15.63,27.36
100	13.54,14.83,18.86	13.40,15.22,27.36	13.24,15.56,27.36	12.93,15.16,19.70
150	13.40,14.91,19.85	13.29,14.70,**18.40**	13.41,**14.37**,19.99	12.98,15.28,20.53
200	13.40,14.80,27.36	12.98,15.87,27,36	**12.80**,14.86,25.14	13.58,15.40,20.05
500	13.38,14.75,20.05	13.30,16.40,27.36	13.40,14.85,27.36	13.56,14.67,20.34
1000	13.40,15.68,23.95	13.40,14.83,18.79	13.36,14.95,27.36	13.44,15.10,27.36

Tablica 6.7: Rezultati testiranja različitih veličina genotipa i vjerojatnosti mutacije uz $P = 500$. Svaka ćelija sadrži minimum,medijan i maksimum dobrote.

U tablici 6.7 podebljane su najbolje vrijednosti za minimum, medijan i maksimum dobrote. Konfiguracije koje daju najbolje rezultate se zatim koriste u optimiranju veličine populacije P. Rezultati za različite veličine populacije prikazani su u tablici 6.8.

N,$p_m \backslash P$	100	200	500	1000	2000
150,0.5	14.17,18.76,27.36	13.58,17.83,27.36	13.29,14.70,**18.40**	13.56,16.49,21.14	13.43,16.46,20.88
150,0.7	13.40,18.78,396.36	13.18,15.08,27.36	13.41,**14.37**,19.99	13.40,15.85,20.99	14.65,16.76,22.81
200,0.7	13.84,17.81,27.36	13.89,16.99,27.36	**12.80**,14.86,25.14	13.40,16.47,23.08	14.01,16.93,21.99
500,0.9	13.30,17.63,27.36	13.58,16.26,20.99	13.56,14.67,20.34	13.40,15.36,19.80	14.03,15.74,20.03

Tablica 6.8: Rezultati testiranja različitih veličina populacije. Svaka ćelija sadrži minimum,medijan i maksimum dobrote.

Na osnovi rezultata iz tablice 6.8, uzima se skup parametara koji ima minimalnu vrijednost medijana dobrote. Koristeći ovaj skup parametara provode se eksperimenti za sve kriterije iz tablice 2.3. Rezultati ovih eksperimenata prikazani su u tablici 6.9.

kriterij	minimum	medijan	maksimum
Cmax	37.86	38.36	40.88
Ft	154.78	159.43	171.53
Uwt	6.65	7.35	7.93
Twt	13.41	14.37	19.99

Tablica 6.9: Rezultati testiranja gramatičke evolucije nad svim kriterijima raspoređivanja uz $N = 150$, $p_m = 0.7$ i $P = 500$

6.2.4. Parametri neuronskih mreža

Za neuronsku mrežu jedan od najbitnijih parametara je struktura mreže gdje se definira broj skrivenih slojeva te broj neurona u svakom sloju. Ako se izabere prejednostavna struktura, mreža neće imati ekspresivnost da točno izrazi nelinearnost kvalitetne prioritetne funkcije. S druge strane, presložena struktura može se prenaučiti na skupu podataka za učenje što će rezultirati slabom generalizacijom. Korištenjem pretraživanja po rešetci eksperimentira se s nekoliko parova strukture St i vjerojatnosti mutacije p_m dok je veličina početne populacije fiksirana i iznosi 100. Tablica 6.10 prikazuje rezultate ovih eksperimenata.

$St \backslash p_m$	0.3	0.5	0.7	0.9
9-3-1	13.94,15.80,19.60	14.49,15.70,18.16	14.47,15.52,19.18	14.17,15.61,18.80
9-7-1	14.58,15.46,18.01	14.37,16.02,18.57	14.71,16.11,19.82	14.46,15.55,20.59
9-10-1	14.17,16.23,18.89	14.19,15.69,19.10	14.52,15.77,18.83	14.57,15.62,19.31
9-15-1	14.43,16.02,20.72	14.75,15.91,18.75	14.50,15.51,19.55	**13.66**,16.23,19.73
9-5-3-1	13.73,15.79,28.15	13.89,15.14,19.49	13.74,**15.09**,20.89	14.04,15.34,17.96
9-7-3-1	13.98,15.91,24.35	14.39,15.70,20.15	13.81,16.35,21.46	13.74,15.29,**17.49**
9-7-5-1	13.49,15.83,20.37	13.46,15.78,21.25	13.54,15.42,24.77	13.69,15.13,19.54

Tablica 6.10: Rezultati testiranja različitih struktura mreže i vjerojatnosti mutacije uz $P = 100$. Svaka ćelija sadrži minimum,medijan i maksimum dobrote.

Iz tablice 6.10 vidi se da najmanji medijan daje struktura 9-5-3-1 i vjerojatnost mutacije 0.7. Ovi parametri se zatim koriste u optimiranju veličine populacije. Rezultati za različite veličine populacije prikazani su u tablici 6.11.

P	minimum	medijan	maksimum
50	13.71	15.96	21.48
100	13.74	15.09	20.89
200	13.93	15.29	21.91
500	**13.57**	**14.29**	**17.73**

Tablica 6.11: Rezultati testiranja različitih veličina populacije uz St = 9-5-3-1 i p_m = 0.7

Kao što je vidljivo u tablici 6.11, najbolje rezultate uvjerljivo daje veličina populacije 500. S tim skupom parametara provode se eksperimenti za sve kriterije raspoređivanja. Rezultati ovih eksperimenata prikazani su u tablici 6.12.

kriterij	minimum	medijan	maksimum
Cmax	40.41	41.25	42.44
Ft	171.23	172.56	177.13
Uwt	6.62	6.86	7.51
Twt	13.57	14.29	17.73

Tablica 6.12: Rezultati testiranja neuronskih mreža nad svim kriterijima raspoređivanja uz St = 9-5-3-1, p_m = 0.7 i P = 500

6.3. Usporedba hiperheurističkih postupaka

Kako bi ocijenili kvalitetu pristupa opisanih u ovom radu, rezultati se uspoređuju s postojećim metodama iz [1]. Tablica 6.13 objedinjuje najbolje rezultate svih metoda na svim kriterijima raspoređivanja. Osim metoda opisanih u ovom radu, prikazuju se i rezultati preuzeti iz [1] za genetsko programiranje (GP), GEP i determinističke heuristike min-min, max-min, min-max i sufferage. Svi rezultati su dobiveni optimizacijom parametara s Twt kriterijem.

metoda	Cmax	Ft	Uwt	Twt
min-min	38.31	157.20	7.14	16.72
max-min	38.84	195.89	8.14	22.07
min-max	38.07	167.30	7.79	17.49
sufferage	37.93	160.97	7.20	16.65
GP	37.95,38.26,38.87	153.64,155.31,158.84	**6.33**,7.06,7.84	**12.96**,13.60,14.62
GEP	37.95,38.22,38.73	**153.53**,154.83,158.09	6.44,6.93,7.55	13.06,13.69,15.14
CGP, k-tournament	37.88,38.27,38.71	153.72,154.93,158.62	6.61,7.23,7.67	13.38,13.81,15.42
CGP, evol. strat.	37.90,38.39,38.71	153.75,155.06,156.48	6.77,7.19,7.42	13.44,14.09,15.24
GE	**37.86**,38.36,40.88	154.78,159.43,171.53	6.65,7.35,7.93	13.41,14.37,19.99
NN	40.41,41.25,42.44	171.23,172.56,177.13	6.62,6.86,7.51	13.57,14.29,17.73

Tablica 6.13: Pregled rezultata svih metoda. Svaka ćelija sadrži minimum,medijan i maksimum dobrote.

Iz tablice 6.13 vidi se da CGP i GE nadmašuju determinističke heuristike u svim kriterijima. Iako su se CGP i GE pokazali uspješnima, ipak daju u većini kriterija ponešto slabije rezultate od GP-a i GEP-a. Iznenađenje u rezultatima su neuronske mreže koje nisu uspijele postići zadovoljavajuće rezultate. Neuronske mreže za kriterije Cmax i Ft čak daju lošije rezultate i od spomenutih deterministčkih heuristika. Mogući razlog za to je jednostavnost isprobanih struktura mreže. S druge strane, kompleksnije mreže bi iziskivale veću računalnu moć u izvođenju. Čak i jednostavnije mreže koje su isprobane su kompleksnije od ostalih metoda jer koriste složenije operacije kao što je potenciranje u sigmoidalnoj funkciji.

7. Zaključak

U ovom radu opisan je hiperheuristički model za evoluciju prikladnih heuristika raspoređivanja koje se zatim koriste u algoritmu prioritetnog raspoređivanja. U sklopu modela koristi se kartezijsko genetsko programiranje, gramatička evolucija i neuronske mreže gdje se za svaku od metoda definira prikaz rješenja i evolucijski operatori križanja i mutacije. Razvijen je programski sustav za ispitivanje različitih prikaza rješenja i ograničenja problema. Ispitana je učinkovitost razvijenog pristupa s obzirom na postojeće heuristike s naglaskom na dinamičke uvjete raspoređivanja.

Pokazano je da su metode gramatičke evolucije i kartezijskog genetskog programiranja nadmašile postojeće deterministčke heuristike te su bile konkuretne metodama klasičnog genetskog programiranja. S druge strane, neuronske mreže su se pokazale nedovoljno uspješne te za neke kriterije raspoređivanja ne uspijevaju nadmašiti niti postojeće deterministčke heuristike. Razlog za to može biti jednostavnost testiranih struktura mreža, ali s druge strane kompleksnije mreže općenito su vremenski i prostorno složenije od ostalih metoda. Postojeće metode je vjerojatno moguće poboljšati korištenjem različitih evolucijskih operatora od onih korištenih u ovom radu. Također bi se mogli dobiti bolji rezultati uvođenjem lokalnih operatora pretraživanja.

Problem raspoređivanja ima široku primjenu (npr. raspoređivanje dretvi na procesore u operacijskim sustavima) te smatram da je važno istraživati nove pristupe koji bi doveli do boljih rezultata jer, kao što je vidljivo iz rezultata, deterministčke heuristike koje se najčešće koriste ne daju dobre rezultate. Također, metode opisane u ovom radu vrlo su korisne za rješavanje raznih problema i mogu dati vrlo dobre rezultate u raznim primjenama pa se zato nadam da će ovo istraživanje potaknuti druge da ih koriste.

8. Literatura

[1] M. Đurasević, "Optimizacija raspoređivanja u okruženju nesrodnih strojeva", Zagreb, 2014.

[2] J. Y-T. Leung, “Handbook of Scheduling: Algorthms, Models, and Performance Analysis”, Chapman & Hall/CRC, 2004.

[3] M. P. Pinedo, “Scheduling: theory, algorithms, and systems" , Springer Science & Business Media, 2012.

[4] Z. Michalewicz, “Genetic Algorithms + Data Structures = Evolutionary Programs”, Springer-Verlag, Berlin, 1992.

[5] "ECF - Evolutionary Computation Framework", URL: http://ecf.zemris.fer.hr/, 18.7.2017.

[6] J. Branke, T. Hildebrandt, B. Scholz-Reiter, "Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations", Massachusetts Institute of Technology, 2014

[7] F. Noorian, A. M. deSilva, P. H. W. Leong, "Grammatical Evolution: A Tutorial using gramEvol", Massachusetts Institute of Technology, 2016

[8] "Cartesian Genetic Programming", URL: http://www.cartesiangp.co.uk/, 18.7.2017.

[9] J. F. Miller, "Cartesian Genetic Programming in a nutshell", University of York, UK, 2012

[10] Zhen-Guo Che, Tzu-An Chiang, Zhen-Hua Che, "Feed-forward neural networks training: a comparison between genetic algorithm and back-propagation learning algorithm", ICIC International, 2011

[11] K. Gurney, "An introduction to neural networks", University of Sheffield, 1997

[12] E. W. Dijkstra, "ALGOL-60 Translation", 1961

Hiperheurističke metode rješavanja problema raspoređivanja u okruženju nesrodnih strojeva

Sažetak

Problem raspoređivanja je vrlo poznat i spada u klasu NP teških problema, što znači da ne postoje efikasni algoritmi koji pronalaze optimalno rješenje unutar razumnih vremenskih ograničenja. Jedna od metoda rješavanja tog problema je prioritetno raspoređivanje gdje se raspored dinamički gradi uz pomoć predefinirane heurističke prioritetne funkcije. U ovom radu opisuje se hiperheuristički model za pronalaženje prioritetnih funkcija temeljen na evolucijskim algoritmima i neurološkim sustavima. U sklopu modela opisuje se kartezijsko genetsko programiranje, gramatička evolucija i neuronske mreže kao tri različita pristupa za evoluciju prioritetnih funkcija. Za svaki pristup se posebno definira struktura, prikaz rješenja i evolucijski operatori križanja i mutacije. Svaki od ta tri pristupa se testira te se analizira njihova efikasnost u usporedbi s postojećim rezultatima drugih metoda. Osim usporedbe kvalitete rezultata, analizira se i vremenska i prostorna složenost različitih pristupa te se daju prijedlozi za moguća poboljšanja.

Ključne riječi: raspoređivanje, prioritetno raspoređivanje, nesrodni strojevi, genetski algoritam, evolucijski algoritam, genetsko programiranje, kartezijsko genetsko programiranje, gramatička evolucija, umjetne neuronske mreže

Hyper-heuristic methods for solving job shop scheduling problems with unrelated machines

Abstract

Job shop scheduling is a well known NP-hard optimization problem, which means an algorithm that finds the optimal solution within reasonable time constraints does not exist. One of the methods for solving this problem is priority scheduling where the final schedule is dynamically constructed using a predefined priority function (heuristic). This paper describes a hyper-heuristic model for generating priority functions based on evolutionary algorithms and artificial neural networks. This model describes cartesian genetic programming, grammatical evolution and artifical neural networks as three different methods for evolving priority functions. Structure, solution representation and evolutionary operators are defined for each method. All the methods are tested and their efficiency is analyzed and compared to existing methods. Also, time and space complexity of the methods is analyzed and possible future improvements are proposed.

Keywords: job shop scheduling, priority scheduling, unrelated machines, genetic algorithm, evolutionary algorithm, genetic programming, cartesian genetic programming, grammatical evolution, artificial neural network