SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4329

Raspoređivanje s ograničenjima u okolini nesrodnih strojeva

Ivan Ćorić

Zagreb, svibanj 2016.

Sadržaj

Uvod

U stvarnom svijetu konstantno se susrećemo s mnogo raznovrsnih problema raspoređivanja, kao što su primjerice raspoređivanje u proizvodnim procesima, raspoređivanje procesa po računalima u *cluster* okruženjima i mnogim drugim. Cilj problema raspoređivanja jest izraditi raspored koji optimizira određene kriterije (npr. ukupnu duljinu izvođenja rasporeda). Nažalost, većina problema raspoređivanja spada u kategoriju NP-teških problema, pa posljedično ne postoji algoritam koji u „razumnom“ vremenu može pronaći optimalno rješenje za navedene probleme. Zbog tog razloga, problemi raspoređivanja se najčešće rješavaju korištenjem raznih heurističkih metoda. Iako heurističke metode ne garantiraju pronalazak optimalnog rješenja za probleme raspoređivanja, one su ipak u stanju pronaći rješenja dovoljne kvalitete u prihvatljivom vremenskom periodu.

U ovom radu definiran je problem raspoređivanja s ograničenjima u okolini nesrodnih strojeva te je opisano rješavanje istog metodom genetskog programiranja. Naime, genetsko programiranje koristi se da bi se automatski razvila konkretna heuristika koja se može koristiti za rješavanje različitih instanci problema raspoređivanja, pri čemu se optimizira određeni kriterij. S obzirom da se u stvarnom svijetu mnogi problemi raspoređivanja pojavljuju s raznim ograničenjima, u ovom radu stavljen je naglasak na prilagodbu genetskog programiranja za rješavanje upravo takvih problema. U sklopu ovog rada omogućit će se rješavanja problema raspoređivanja s ograničenjima poput vremena postavljanja (koliko vremena je potrebno poslu da se pripremi nakon što je prethodno neki drugi posao završio s izvođenjem na određenom stroju) i međuovisnosti poslova (neki poslovi se ne mogu krenuti izvršavati prije nego drugi poslovi završe s izvođenjem).

U prvom poglavlju opisan je problem raspoređivanja, dana je njegova definicija, podjela te su navedeni kriteriji koji se koriste za ocjenu rasporeda. U drugom poglavlju definiran je meta-algoritam kojim će se vršiti raspoređivanje uz danu funkciju prioriteta te objašnjeno na koji se način traži funkcija prioriteta pomoću genetskog programiranja. U trećem poglavlju opisana su ograničenja koja ovaj rad proučava. U posljednjem poglavlju prikazani su rezultati ostvareni ovim radom.

1. Raspoređivanje

Raspoređivanje je problem donošenja odluke u kojem trenutku i na koji stroj je potrebno rasporediti posao na izvođenje. S obzirom na to da je ovo vrlo općenit problem, mnogi primjeri iz stvarnog svijeta (primjerice raspoređivanje pošiljki na kamione ili raspoređivanje dretvi na procesore) mogu se svesti upravo na različite probleme raspoređivanja. Iz tog razloga može se uočiti i velika mogućnost primjene u industriji.

U ovom poglavlju bit će dana formalna definicija problema raspoređivanja. Za početak ćemo se ograničiti na poseban slučaj raspoređivanja u kojem su strojevi nesrodni tj. nezavisni. To, naravno, ne mora uvijek biti slučaj (npr. ako znamo da se svi poslovi na nekom stroju izvode za određeni faktor sporije nego na nekom drugom stroju), no okolina nesrodnih strojeva je jedna od općenitijih okolina te se razmatra upravo iz tog razloga. Također ćemo, na početku, pretpostaviti da su i svi poslovi nezavisni te ćemo kasnije uvesti ograničenja među njima.

1.1. Formalna definicija

Neka je dano n poslova i m strojeva, gdje su n i m konačni prirodni brojevi. Poslovi i strojevi jednoznačno su određeni prirodnim brojem, manjim ili jednakim n odnosno m. Neka se indeks j odnosi na posao, a i na stroj. Za svaki posao definirana su sljedeća svojstva:

- Trajanje izvođenja posla j na stroju i – p_{ij}
- Vrijeme pripravnosti posla j – r_j
- Vrijeme željenog završetka posla – d_j
- Težina posla – w_j

Rješenje ovog problema mora davati odgovor na pitanje u kojem trenutku i kojem stroju je potrebno rasporediti pojedini posao.

1.2. Podjela raspoređivanja

Ovisno o trenutku u kojem doznajemo karakteristike poslova razlikujemo dvije vrste raspoređivanja: statičko i dinamičko.

Ako su karakteristike poslova poznate prije nego poslovi postanu spremni za izvođenje, tada možemo odrediti raspored prije izvođenja sustav. Ovakvo raspoređivanje naziva se statičko (engl. *off-line*) raspoređivanje. Za ovaj tip raspoređivanja, najčešće su korištene metode bazirane na pretraživanju, kao primjerice genetski algoritmi.

S druge strane, ako karakteristike poslova nisu poznate prije nego poslovi postanu spremni za izvršavanje, tada se gotovo uvijek koriste heurističke metode te metode genetskog programiranja. Ovaj tip raspoređivanja naziva se dinamičko (engl. *on-line*) raspoređivanje. Kako u ovom tipu raspoređivanja često dolazi do promjena u sustavu, metode bazirane na pretraživanje se najčešće ne mogu primijeniti zbog vremena koje im je potrebno da pronađu kvalitetna rješenja.

1.3. Kriterij raspoređivanja

U literaturi se mogu pronaći različiti načini koji određuju kvalitetu rasporeda ([2] Pinedo, 2012.). Jedan od kriterija koji se odmah nameće može biti ukupno kašnjenje koje je uzrokovao dani raspored. No onda možemo postaviti pitanje što ako dva rasporeda nemaju kašnjenje, kako onda odrediti koji je raspored bolji. U ovom potpoglavlju dani su kriteriji po kojima se može ocijeniti kvaliteta rasporeda.

1.3.1. Težinsko zaostajanje (Twt)

Neka je zaostajanje posla definirano kao:

$$T_j = max\{C_j - d_j, 0\} \tag{1}$$

gdje je C_j vrijeme u kojem je posao *j* završio izvođenje. Sada definiramo kriterij težinskog zaostajanja (engl. *weighted tardiness*) kao:

$$T = \sum_j w_j T_j. \tag{2}$$

1.3.2. Težinsko protjecanje (Ft)

Neka je vrijeme posla u sustavu definirano kao:

$$F_j = E_j - r_j. \tag{3}$$

Definiramo kriterij težinskog protjecanja (engl. *weighted flowtime*) kao:

$$F = \sum_j w_j F_j. \tag{4}$$

1.3.3. Težinski broj zaostalih poslova (Nwt)

Neka je funkcija koja određuje je li posao kasnio dana s:

$$U_j = \begin{cases} 1, T_j > 0 \\ 0, inače \end{cases} \tag{5}$$

Definiramo kriterij težinskog broja zaostalih poslova (engl. *weighted number of tardy jobs*) kao:

$$U = \sum_j w_j U_j. \tag{6}$$

1.3.4. Ukupna duljina rasporeda (Cmax)

Ovaj kriterij je definiran, kako i samo ime govori, kao ukupno trajanje potrebno da se izvrši čitav raspored (engl. *makespan*).

$$C = \max_j(C_j) \tag{7}$$

1.4. Okolina nesrodnih strojeva

Kao što je već navedeno ovaj rad bavi se isključivo problemom raspoređivanja u okolini nesrodnih strojeva. U takvoj okolini svaki stroj izvodi posao proizvoljno definiranom brzinom *pij*.

2. Raspoređivanje pomoću genetskog programiranja

U ovom poglavlju opisat će se primjena genetskog programiranja kako bi se automatski razvila nova pravila raspoređivanja. Ovako razvijen postupak raspoređivanja sastoji se od dva dijela: funkcije prioriteta i meta-algoritma. Funkcija prioriteta za dani posao i stroj određuje vrijednost koliko je poželjno rasporediti dani posao na dani stroj. Formalnije, funkciju prioriteta možemo definirati kao:

$$\pi_{ij} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R} \tag{7}$$

No sama funkcija prioriteta nije sama po sebi dovoljna te je potrebno dodatno definirati *meta-algoritam* koji na temelju dane funkcije prioriteta određuje koji posao će se rasporediti na koji stroj. Sama prioritetna funkcija razvit će se korištenjem genetskog programiiranja, dok je meta-algoritam potrebno definirati ručno. U nastavku poglavlja detaljnije je opisan sam postupak raspoređivanja pomogu genetskog programiranja, no prije toga dan je kratki uvod u genetsko programiranje.

2.1. Genetsko programiranje

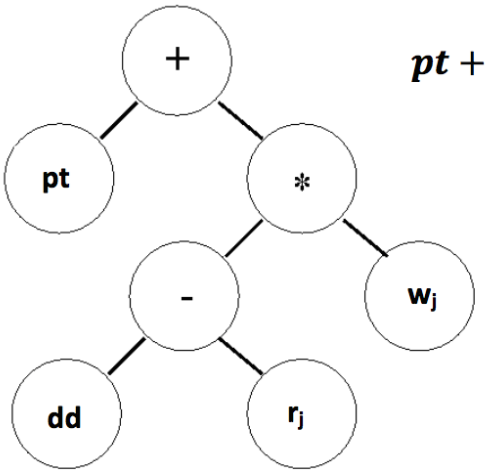
Genetsko programiranje (GP) je postupak u kojem su računalni programi predstavljeni kao geni koji se modificiraju određenim evolucijskim algoritmom te nakon evolucije uspješno rješavaju dani problem. Kao evolucijski algoritam često se koristi genetski algoritam. Takav algoritam simulira prirodnu evoluciju na način da obavlja operacije poput mutacije (stvara se novi program nasumičnim mijenjanjem slučajno odabranog dijela programa „roditelja“) te križanja (stvara se novi program koristeći nasumično izabrane dijelove od dva programa „roditelja“) nad samim programima.

Prvo se nasumično generira populacija jedinki računalnih programa. Svaka jedinka se izvodi nad nekoliko instanci problema kako bi se izračunala njezina funkcija dobrote (engl. *fitness function*), tj. vrijednost koja nam govori koliko dobro jedinka izvršava svoju

zadaću. Nakon generiranja početne populacije stvara se nova populacija koju čine odabrane jedinke iz stare populacije te određeni broj jedinki koje su nastale primjenom genetskih operatora nad starom populacijom. Jedinke nad kojima će se provoditi genetski operatori odabiru se korištenjem turnirske selekcije. Turnirska selekcija simulira turnire nad čijim pobjednicima se provode genetski operatori. Navedeni postupak se ponavlja s novom populacijom dok se ne zadovolji određeni kriterij zaustavljanja, kao primjerice: dostizanje maksimalnog broja generacija ili evaluacija, broj evaluacija bez poboljšanja u kvaliteti jedinki i slično.

2.2. Razvoj funkcije prioriteta

Funkcija prioriteta (svaki gen pa i konačno rješenje) imat će oblik stabla. Na slici Slika 1. nalazi se primjer funkcije prioriteta. Stablo se sastoji od dva tipa čvorova: funkcijskih i terminalnih. Terminalni čvorovi predstavljaju listove stabla. U njima se nalaze podaci koji su važni u odlučivanju prioriteta za pojedini posao, poput: p_{ij} i d_j. U funkcijskim čvorovima nalaze se funkcije koje se mogu obavljati nad terminalnim i drugim funkcijskim čvorovima (npr. zbrajanje ili množenje). Na slici Slika1. prikazan je primjer stabla te izraz koji ono predstavlja. Korišteni funkcijski i terminalni čvorovi dani su u tablicama Tablica 1. i Tablica 2. Bitno je za naglasiti kako je odabir skupa terminalnih čvorova koji će se koristiti tijekom evolucije iznimno bitan, jer će o tom skupu uvelike ovisiti ekspresivnost razvijenih funkcija prioriteta.



$$pt + (dd - r_j) * w_j$$

Slika 1. Primjer funkcije prioriteta

Funkcijski čvorovi	Objašnjenje
+	binarni operator zbrajanja
-	binarni operator oduzimanja
*	binarni operator množenja
/	binarni operator sigurnog dijeljenja : $\frac{a}{b} = \begin{cases} 1 \ ako \ \lvert b \rvert < 0.000001 \\ \frac{a}{b} \ inače \end{cases}$
POS	unarni operator, POS(x) = max(x,0)

Tablica 1. Korišteni funkcijski čvorovi

Terminalni čvorovi	Objašnjenje
pt	vrijeme izvođenja
pmin	minimalno (po svim strojevima) vrijeme izvođenja
pavg	prosječno (po svim strojevima) vrijeme izvođenja
age	vrijeme provedeno u sustavu
w	težina određenog posla
dd	trenutak do kojeg je poželjno da posao završi
Terminalni čvorovi korišteni kod ograničenja vremena postavljanja	
savg	prosječno (po svim strojevima) vrijeme postavljanja određenog posla
Terminalni čvorovi korišteni kod ograničenja u redoslijedu izvođenja	
nblk	broj svih poslova koji moraju čekati na završetak posla *j*
nwait	broj svih poslova na čiji završetak izvođenja posao *j* mora čekati

Tablica 2. Korišteni terminalni čvorovi

2.3. Meta-algoritam

Nakon što je u prošlom poglavlju opisano kako se pomoću genetskog programiranja evolura prioritetna funkcija, u ovom poglavlju opisat će se meta-algoritam koji korištenjem navedenih prioritetnih funkcija stvara raspored. Znajući funkciju prioriteta želimo napisati meta-algoritam kojim ćemo rasporediti poslove po strojevima. Koristit ćemo meta-algoritam korišten u radu [1] (Đurasević et al., 2015). Pseudo-kôd meta-algoritma prikazan je u programskom isječku 1.

```
dok postoje neraspoređeni poslovi:
      čekaj dok posao ne postane spreman ili završi;
      za sve spremne poslove i sve strojeve:
            izračunaj π_ij;
      za sve spremne poslove:
            odredi najbolji stroj;
      dok postoji posao čiji je najbolji stroj slobodan:
            rasporedi posao najvećeg prioriteta na njegov
            najbolji stroj;
```

Programski isječak 1. Meta-algoritam korišten za raspoređivanje

Najbolji stroj nekog posla je onaj na kojem funkcija prioriteta daje najveću vrijednost pa se na njega želi rasporediti posao. Nakon što uvedemo ograničenja, ovaj će meta-algoritam biti prilagođen svakom od ograničenja u sljedećem poglavlju.

Ovo nije jedini pristup rješavanju ovoga problema. Naime, kao što se evoluira funkcija prioriteta, tako se i meta-algoritam mogao evoluirati. . Nažalost evoluiranje samog meta-algoritma je u suštini mnogo teže nego evoluiranje prioritetnih funkcija te se stoga u ovom radu isključivo koristi meta-algoritam čiji je pseudokod dan u programskom isječku 1, dok je drugi pristup naveden samo radi postpunosti i neće se razmatrati u ovom radu.

3. Raspoređivanje s ograničenjima pomoću GP-a

U ovom poglavlju poopćit ćemo dosadašnje razmatranje raspoređivanja na način da u sustav uključimo dodatna ograničenja. Proučit će se dva bitna ograničenja: vrijeme postavljanja (engl. *setup time*), te ograničenja u redoslijedu (engl. *precedence constraints*).

3.1. Ograničenje u redoslijedu izvođenja

Ograničenja u redoslijedu izvođenja su ograničenja u kojima poslovi mogu ovisiti o drugim poslovima na način da se ne smiju krenuti izvršavati dok se određen skup poslova ne izvrši ([2] Pinedo, 2012.). Za svaki posao *j* definiramo skup $cekaj_j$ i $blokiraj_j$ koji govore završetak kojih poslova posao *j* mora čekati tj. koji poslovi moraju čekati završetak posla *j*.

U ovom slučaju, kada imamo samo ograničenja u redoslijedu, prije nego što posao postane spreman za izvođenje, moramo provjeriti jesu li završili svi poslovi na koje treba čekati. U slučaju da svi prethodnici nisu završili s izvođenjem, posao nije spreman, već je u stanju pripravnosti. Kako bismo ga onda postavili kao spremnog kada svi poslovi koji ga blokiraju završe s izvođenjem, svaki posao kada završi s izvođenjem, oslobađa sve poslove koje je blokirao do toga trenutka.

Pseudo-kôd modificiranog meta-algoritma prikazan je u programskom isječku 2. Ovdje je u originalni meta-algoritam dodano ograničenje u redoslijedu izvođenja.

```
dok postoje neraspoređeni poslovi:
      čekaj dok posao ne postane pripravan ili završi;
      ako je posao postao pripravan:
            ako posao ne čeka nikoga:
                  stavi posao u spremne poslove;
            inače:
                  posao ostaje pripravan;
      ako je posao završio:
            oslobodi sve poslove koje blokira;
      za sve spremne poslove i sve strojeve:
            izračunaj π_ij;
      za sve spremne poslove:
            odredi najbolji stroj;
      dok postoji posao čiji je najbolji stroj slobodan:
            rasporedi posao najvećeg prioriteta na njegov
            najbolji stroj;
```

Programski isječak 2. Meta-algoritam korišten za raspoređivanje uz ograničenje redoslijeda

Kako bismo omogućili funkciji prioriteta da se prilagodi na sustav s ograničenjima, pri njezinoj konstrukciji, trebamo joj dati mogućnost da koristi određene informacije o ograničenjima. Iz tog razloga za ovo ograničenje definirani su sljedeći terminali: ukupan broj poslova na koje posao čeka (*nwait*), ukupan broj poslova koje posao blokira (*nblk*).

3.2. Vrijeme postavljanja

Vrijeme postavljanja s_{ijk} je vrijeme koje je potrebno da se posao *j* postavi na stroj *i* neposredno nakon izvođenja posla *k*. Kao primjer možemo uzeti prilagođavanje stroja za tiskanje ili rezanje između obavljanja dva različita posla. Poseban slučaj vremena postavljanja je vrijeme potrebno da se posao postavi na stroj ako se prije njega na stroju nije izvodio niti jedan posao, to vrijeme označit ćemo sa s_{ijj}.

Kada je na poslove nametnuto samo ograničenje vremena postavljanja tada se trebamo pobrinuti oko situacije u kojoj se posao izvršava neposredno nakon drugog posla, na istome stroju. U toj situaciju vrijeme koje trebamo pričekati prije izvršavanje posla *i* iznosi s_{ijk}.

Pseudo-kôd modificiranog meta-algoritma prikazan je u programskom isječku 3. Ovdje je u originalni meta-algoritam dodano ograničenje u vremenu izvođenja.

```
dok postoje neraspoređeni poslovi:
      čekaj dok posao ne postane spreman ili završi;
      za sve spremne poslove i sve strojeve:
            izračunaj π_ij;
      za sve spremne poslove:
            odredi najbolji stroj;
      dok postoji posao čiji je najbolji stroj slobodan:
            rasporedi posao najvećeg prioriteta na njegov
            najbolji stroj, nakon s_ijk;
```

Programski isječak 3. Meta-algoritam korišten za raspoređivanje uz ograničenje vremena postavljanja

Varijabla *i* u programskom isječku 3. predstavlja najbolji stroj, za posao *j*, a *k* je posao koji se izvršavao na stroju *i* neposredno prije. U slučaju da se nitko nije izvršavao na stroju *i*, tada *k* = *j*.

Kako bi se omogućilo funkciji prioriteta da koristi podatke o ovoj vrsti ograničenja, u skup terminala dodano je prosječno vrijeme postavljanja posla na svim strojevima (*savg*).

3.3. Kombinacija ograničenja

Ako želimo imati obje vrste ograničenja u sustavu, tada je potrebno modificirati meta-algoritam na oba načina kako je opisano u prošla dva potpoglavlja. To možemo napraviti jer su dijelovi kôda, koji su mijenjani pri dodavanju oba ograničenja, nezavisni.

Pseudo-kôd modificiranog meta-algoritma prikazan je u programskom isječku 4.

```
dok postoje neraspoređeni poslovi:
      čekaj dok posao ne postane pripravan ili završi;
      ako je posao postao pripravan:
            ako posao ne čeka nikoga:
                  stavi posao u spremne poslove;
            inače:
                  posao ostaje pripravan;
      za sve spremne poslove i sve strojeve:
            izračunaj π_ij;
      za sve spremne poslove:
            odredi najbolji stroj;
      dok postoji posao čiji je najbolji stroj slobodan:
            rasporedi posao najvećeg prioriteta na njegov
            najbolji stroj, nakon s_ijk;
```

Programski isječak 4. Meta-algoritam korišten za raspoređivanje uz kombinaciju ograničenja

Prvi dio je potreban kako bi se osiguralo da se posao ne izvodi dok nisu završili s izvođenjem svi poslovi koje mora čekati te drugi koji bi osigurao da stroj pričeka vrijeme postavljanja između dva posla.

Kod kombinacije ograničnja, uz terminale koje koristimo kod problema bez ograničenja, dodatno koristimo i terminale definirane u potpoglavljima 3.1. i 3.2.

3.4. Izrada ispitnih primjera

Kako bi bilo moguće testirati rad genetskog programiranja pri rješavanju problema raspoređivanja s ograničenjima, potrebno je izraditi primjere koji će se koristiti u procesu učenja i ocjene genetskog programiranja.

Ograničenja u redoslijedu izvođenja generiramo parametrizirano. Parametri koji se koriste su: maksimalan broj neposrednih prethodnika, maksimalan broj neposrednih sljedbenika, udio poslova koji sigurno neće imati prethodnike te udio poslova koji mogu imati prethodnike.

Ograničenja u redoslijedu zadana su listom prethodnika tj. poslova koji se trebaju izvršiti prije određenog posla. Navedena ograničenja generirana su na način da se svim poslovima, iz dijela poslova koji smiju imati prethodnike, odredi razina (broj koji određuje o kojim poslovima dani posao može ovisiti). Neka je posao koji smije imati prethodnike razine *k*, tada taj posao smije ovisiti samo o poslovima manje razine. Poslovi koji ne ovise niti o jednom drugom poslu su nulte razine. Svim poslovima razine veće od nula, nasumično generiramo ovisnosti o poslovima koji su manje razine od njih.

Svim poslovima nulte razine generiramo r_j uniformnom razdiobom za vrijeme cijelog rada sustava. Nakon toga generiramo r_j za sve ostale poslove, pazeći pritom da za poslove *j* i *k* mora vrijediti $r_j < r_k$, ako posao *k* mora čekati na završetak izvođenja posla *j*

Vremena postavljanja također generiramo parametrizirano. Uvest ćemo parametar η koji nam predstavlja omjer srednjeg vremena postavljanja *s* i prosječnog trajanja izvođenja poslova na svim strojevima $\bar{p}$.

$$\eta = \frac{s}{\bar{p}} \tag{8}$$

Iz η i $\bar{p}$ može se izračunati srednje vrijeme postavljanja te normalnom razdiobom mogu se generirati vremena postavljanja za sve kombinacije poslova i strojeva.

Rezultat generiranja vremena postavljanja je memorijski velika datoteka. Razlog tome je što za jedan problem, moramo generirati $n^2 m$ vremena postavljanja. Ovisno o konkretnoj instanci problema raspoređivanja, taj može postati izuzetno velik

3.5. Dodatna ograničenja

Sustav bi se mogao dodatno poopćiti dodavanjem različitih ovisnosti te omogućavanjem ovisnosti između strojeva. Neke od ovisnosti među poslovima mogle bi biti:

- Ograničenja u izvođenju poslova na strojevima - određeni poslovi mogu se izvoditi samo na određenom podskupu strojeva, a ne na svim strojevima.
- Nedostupnost strojeva - u određenim trenucima pojedini strojevi postaju nedostupni.

- Prekidivost poslova - određeni posao se može prekinuti, tako da se neki drugi posao može izvoditi, a ovaj može nastaviti kasnije (ili nastaviti s izvođenjem na nekom drugom stroju).

No, dodavanjem dodatnih ograničenja, trebalo bi dobro razmisliti na koji načni dodavati nova ograničenja te među njima birati ona za koja želimo da su uključena u sustav. Dodavanje odnosno biranje terminala nije tako velik problem, no problem je u tome što svako ograničenje uzrokuje malu promjenu u kôdu meta-algoritma. Ako bi se naivno krenulo za svako ograničenje pisati vlastitu inačicu meta-algoritma, kod kombiniranja ograničenja, te biranja onih koje želimo u sustavu, došli bismo do velikog problema. Smatram da ni dodavanje *if*-ova u kôd isto tako nije opcija jer brzo dolazimo do nepreglednog, krutog te viskoznog kôda. Navedeni problem bi se vjerojatno mogao riješiti primjenom oblikovnog obrasca *dekorator*.

4. Rezultati

U ovom poglavlju prikazani su rezultati dobiveni primjenom genetskog programiranja nad problemima raspoređivanja s ograničenjima. Konačni rezultati prikazani su u tablici Tablica 3. U narednim potpoglavljima komentirani su rezultati svakog od posebnih slučajeva ograničenja: samo vrijeme postavljanja, samo ograničenje redoslijeda izvođenja, vrijeme postavljanja i ograničenje redoslijeda izvođenja te sustav bez ograničenja. Sustav bez ograničenja uključen je kako bi se dobile referentne vrijednosti za usporedbu s ostalim rezultatima.

Testiranje je napravljeno na način da se za svaki meta-algoritam, koristeći genetsko programiranje, tražila funkcija prioriteta koja optimira određeni kriterij. U tablici Tablica 3. opisani su parametri koji su korišteni za optimizaciju. Svaki test rađen je na skupu od 10 instanci problema. Broj poslova u problemima može biti 12, odnosno 25, dok je broj strojeva bio 3. Svaki test pokrenut je 10 puta.

Parametar	Vrijednost
Veličina turnira	3
Najmanja dubina stabla funkcije prioriteta	1
Najveća dubina stabla funkcije prioriteta	5
Najveći broj uzastopnih iteracija bez promjene	10

Tablica 3. Prikaz parametara korištenih za optimizaciju

Za svaki eksperiment istaknute su najmanja, prosječna te najveća vrijednost postignuta optimiziranjem određenog kriterija. Također dana je i standardna devijacija.

4.1. Ograničenje u redoslijedu izvođenja

Iz Tablice 3. može se zaključiti da, kao što je i bilo očekivano, ograničenje u redoslijedu izvođenja doprinosi dužem trajanju ukupnog izvođenja. Zaostajanje je veće i

od referentnog mjerenja, kao i od ograničenja u vremenu postavljanja. Također i prosječno vrijeme provedeno u sustavu opet je veće od ograničenja vremena postavljanja te referentnog mjerenja bez ograničenja.

Ovakvi rezultati posljedica su činjenice da neki poslovi bivaju blokirani od strane poslova o kojima ovise. Tek se nakon njihova završetka blokirani poslovi mogu krenuti izvršavati, što rezultira većim vremenom provedenim u sustavu, većim zaostajanjem te većim većim ukupnim trajanjem.

4.2. Vrijeme postavljanja

Ograničenje u vremenu postavljanja, kao što vidimo iz Tablice 3., rezultiralo je dužim izvođenjem te većim zaostajanjem. U usporedbi sa sustavom bez ograničenja vidimo da je kašnjenje čak dvostruko veće. Ukupna duljina rasporeda veća je od referentnog mjerenja, također je i veća od sustava s ograničenjem u redoslijedu izvođenja.

Rezultate objašnjavamo time da se vrijeme izvođenja posla povećava zbog toga što osim vremena izvođenja, između svaka dva posla, na svakom stroju imamo i vrijeme koje treba proteći između uzastopnog izvršavanja ta dva posla.

4.3. Vrijeme postavljanja i ograničenje u redoslijedu izvođenja

Ako promatramo sustav u kojem postoje i ograničenje u redoslijedu izvođenja i ograničenje vremena postavljanja, tada rezultate objašnjavamo kao što je to već napravljeno u zadnja dva potpoglavlja.

Ovdje imamo duže ukupno trajanje rasporeda iz razloga što se neki poslovi ne mogu krenuti izvoditi dok ne završe oni koji se moraju izvršiti prije njih, no i zbog toga što sustav treba pričekati određeno vrijeme između izvršavanja dva uzastopna posla.

Meta-algoritam	Kriterij															
	Twt				Nwt				Ft				Cmax			
	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev
Bez ograničenja	2.03	2.74	5.67	1.10	0.48	0.54	0.84	0.12	1.68	1.72	2.01	0.11	0.51	0.69	0.86	0.12
Ograničenje u redoslijedu	2.05	2.82	4.84	0.83	0.61	0.69	0.83	0.05	2.35	3.30	3.70	0.39	0.63	0.76	0.96	0.11
Vrijeme postavljanja	4.56	5.25	7.73	1.01	0.76	0.87	0.93	0.05	3.22	3.25	3.58	0.09	1.42	1.50	1.54	0.05
Kombinacija ograničenja	5.94	6.10	6.32	0.12	0.77	0.86	0.94	0.06	4.14	5.12	7.55	1.18	1.45	1.53	1.56	0.02

Tablica 4. Prikaz rezultata mjerenja

Zaključak

Ovaj rad opisuje na koji je način u sustav, koji obavlja dinamičko (engl. *on-line*) raspoređivanje poslova u okolini nesrodnih strojeva, moguće uklopiti različita ograničenja. Kako u ovom tipu raspoređivanja često dolazi do promjena u sustavu, metode bazirane na pretraživanju se najčešće ne mogu primijeniti zbog vremena koje im je potrebno da pronađu kvalitetna rješenja. Iz tog se razloga ovaj problem rješava genetskim programiranjem.

Nakon definiranja problema i uvođenja ključnih pojmova, obrađene su dvije vrste ograničenja: vrijeme postavljanja te ograničenja u redoslijedu izvođenja. Uklapanje ograničenja u ovakav sustav jest jednostavno dok je tih ograničenja malo. Rezultati pokazuju kako ograničenja utječu na konačan raspored uspoređujući ih s rasporedom sustava bez ograničenja. Izradom ovoga rada došao sam do novoga pitanja na koje trenutno ne znam odgovor: „Na koji način dinamički mijenjati male dijelove velikog baznog kôda tako da su zadovoljena sva oblikovna načela?“

Literatura

[1] ĐURASEVIĆ, JAKOBOVIĆ, KNEŽEVIĆ, Adaptive scheduling on unrelated machines with genetic programming. Sveučilište u Zagrebu, 2015.

[2] PINEDO, M.L. Scheduling Theory, Algorithms, and Systems, New York University, 2012.

Sažetak

Raspoređivanje s ograničenjima u okolini nesrodnih strojeva

U ovom radu definirani su osnovni pojmovi vezani uz raspoređivanje koristeći genetsko programiranje. Izrađen je sustav za raspoređivanje u okolini nesrodnih strojeva koji metodom genetskog programiranja rješava problem raspoređivanja. U takav sustav uključena su ograničenja: ograničenje u redoslijedu izvođenja te ograničenje vremena postavljanja. Konačno, prikazani su postignuti rezultati u odnosu na sustav za raspoređivanje bez ograničenja.

Ključne riječi

Raspoređivanje u okolini nesrodnih strojeva, ograničenje u redoslijedu izvođenja, vrijeme postavljanja, genetsko programiranje

Summary

Constrained scheduling for the unrelated machines environment

This paper investigates the use of genetic programming in constrained scheduling for unrelated machines. We have built a system for scheduling in the environment of unrelated machines, which uses genetic programming to solve the scheduling problem. In this system we have included the following constraints: precedency constraint and setup time constraint. Achieved results were presented and compared to the system without constraints.

Ključne riječi

Scheduling for unrelated machines, constraints, precedence constraint, setup time, genetic programming

Skraćenice

GP	*Genetic programming*	Genetsko programiranje
Twt	Weighted tardiness	Težinsko zaostajanje
Nwt	Weighted number of tardy jobs	Težinski broj zaostalih poslova
Ft	Weighted flowtime	Težinsko protjecanje
Cmax	Makespan	Ukupna duljina rasporeda