SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Rješavanje problema rasporeda poslova u višeprocesorskim sustavima uz pomoć genetskih algoritama

Ivana Stokić

Voditelj: doc. dr. sc. Domagoj Jakobović

Zagreb, svibanj, 2008.

Sadržaj

1. Uvod

Višeprocesorski sustav (engl. multiprocessor system) je računalni sustav koji sadrži više procesora. U tom se sustavu poslovi programa dijele između procesora i određuje se njihov redoslijed izvršavanja.

Problem raspoređivanja poslova pripada klasi NP problema. Za probleme koji pripadaju spomenutoj klasi problema još se uvijek ne zna postoji li algoritam polinomijalne složenosti koji će pronaći optimalno rješenje. Egzaktni algoritmi koji imaju eksponencijalno vrijeme traženja ne primjenjuju se za rješavanje NP – teških problema, već se za takve probleme koriste heurističke i metaheurističke metode, npr. genetski algoritmi. Od drugih metoda su tu i pohlepni algoritmi (engl. greedy algorithms) - zadani se problem rješava u koracima u kojima se bira lokalno optimalno rješenje i time se smanjuju prostor i složenost pretraživanja; lokalno pretraživanje (engl. *local search, LS*) kod kojeg se obavlja zamjena s najboljim susjedom; zatim tabu pretraživanje (engl. *tabu search*, *TS*) koji kombinira algoritam lokalnog pretraživanja, bijeg iz lokalnog optimuma i izbjegavanje alterniranja dvaju rješenja. U tabu listi pamte se rješenja prethodnih iteracija i ona se u sljedećem koraku više ne mogu upotrijebiti.

U ovom će se seminarskom radu opisati način rješavanja problema rasporeda poslova u višeprocesorskom sustavu (uz određena pojednostavljenja) uz pomoć genetskog algoritma.

2. Genetski algoritam

Genetski algoritam je heuristička (stohastička) metoda optimiranja koja oponaša proces evolucije iz prirode. Njih je 70 – ih godina predstavio John Henry Holland.

Charles Darwin je u 19. st. postavio teoriju evolucije. U prirodi se događa prirodna selekcija kojom preživljavaju najbolje i najsposobnije jedinke, dok ostale izumiru jer se nisu mogle dovoljno prilagoditi okolini i njenim uvjetima.

Okolinu u genetskom algoritmu predstavlja funkcija dobrote ili funkcija cilja, a svaka jedinka ima mjeru - dobrotu. Jedinke koje su loše se eliminiraju, a one bolje (koje imaju veću vjerojatnost preživljavanja, a time i veću dobrotu) ostaju i sudjeluju u reprodukciji.

Svako potencijalno rješenje problema u genetskom algoritmu predstavljeno je kromosomom (jednom jedinkom). Podaci koji obilježavaju jedinku mogu biti zapisani u kromosomu na različite načine ovisno o kakvom se problemu radi. Skup mogućih rješenja čini populaciju jedinki. Populacija se razvija iterativno, tj. kroz generacije kako bi se povećala dobrota jedinki. Postupci primjene genetskih operatora nad jedinkama populacije obavljaju se sve do isteka vremena ili zadovoljenja zadanog uvjeta (npr. određeni broj iteracija ili dostignuće unaprijed zadane vrijednosti dobrote) pri čemu najbolji član trenutne generacije predstavlja najbolje rješenje [1] .

2.1 Primjena genetskih algoritama

Genetski se algoritmi koriste za rješavanje mnogih problema, od jednostavnijih, poput pronalaženja minimuma ili maksimuma zadane funkcije do složenijih, npr. rješavanje problema rasporeda, traženje najkraćeg puta, rješavanje problema n kraljica; daje dobre rezultate pri strategiji igara, problemima sličnim transportnom problemu, problemima određivanja parametara sustava, optimiranju upita nad bazom podataka [1]. Koriste se i pri automatiziranom pregovaranju u elektroničkom poslovanju, optimizaciji portfelja u naftnoj industriji, terminskom planiranju pokretanja energetskih postrojenja, minimizaciji troškova za planiranje održavanja električnih transmisijskih mreža, procjeni rizika u poslovanju (marketing, banke...) u situacijama u kojima su podatci nepotpuni. Također se mogu koristiti pri dizajnu u tehnici, pri portretiranju osumnjičenika, dizajnu optičkih mreža [2]... Iako se ti problemi mogu riješiti i drugim heurističkim metodama, genetski su algoritmi jedna od boljih metoda jer uvijek daju skup dobrih rješenja (ne samo jedno), a postoji i velika sloboda pri uporabi (npr. mogućnost nadogradnje algoritma kako bi se povećala učinkovitost. Pokazano je kako pri rješavanju vrlo složenih problema genetski algoritmi pronalaze inovativne pristupe rješavanja istih, tj. one do kojih ljudi prije nisu došli [2]. Međutim, jedan od većih nedostataka u odnosu na druge metode je sporost izvršavanja (zbog korištenja velikog broja računskih operacija). Genetski algoritmi imaju mogućnost pronaći optimalno rješenje u velikim prostorima pretrage. Kako bi se riješio neki problem, on se može prilagoditi genetskom algoritmu ili se genetski algoritam može prilagoditi problemu pri čemu se njegov rad mijenja kako bi mogao raditi s veličinama koje su specifične tom problemu.

Rješenja problema se najčešće prikazuju u binarnom obliku kao niz bitova, ali postoje i vrlo složeni problemi čije se rješenje tako ne može prikazati, već se prikazuje u obliku dvodimenzionalnog polja, stabla...

Nakon što se odredi problem koji se mora riješiti pomoću genetskog algoritma potrebno je [1] odrediti funkciju dobrote (cilja), odabrati jednu vrstu selekcije koja najviše odgovara danom problemu i koja će sudjelovati u dobivanju najboljeg rješenja, odrediti u kojem će se obliku predstaviti rješenje, definirati uvjet završetka evolucijskog procesa, odrediti generacijski procjep (koliko se jedinki prenosi iz generacije u generaciju), odrediti koja će se vrsta križanja i mutacije upotrijebiti. Također je potrebno odrediti parametre genetskog algoritma.

2.2 Operatori genetskih algoritama

Dvije najvažnije karakteristike genetskog algoritma su selekcija i reprodukcija.

2.2.1 Selekcija

Selekcijom se dobra svojstva jedne jedinke prenose na buduće generacije čime se čuva dobar genetski materijal. Nakon mnogo iteracija postoji mogućnost gubitka dobrih svojstava jedinke - najboljeg mogućeg rješenja. Postupak zaštite najbolje jedinke zove se elitizam. Selekcijom se biraju dobre jedinke, a loše odumiru, nestaju Tri su osnovne vrste selekcija: jednostavna, turnirska i eliminacijska. S obzirom na vrstu selekcije genetski se algoritmi dijele na eliminacijske i generacijske.

Jednostavna selekcija (engl. *roulette-wheel selection)* generira novu populaciju koja ima isti broj jedinki kao i prethodna populacija. Odabiru se roditelji čija je vjerojatnost selekcije proporcionalna njihovoj dobroti. Operator odabira je kotač ruleta. Dobrota pojedine jedinke je D_i, a ukupna dobrota je D (suma D_i). Ova selekcija ima prednosti, ali i dosta nedostataka [3] .

Turnirska selekcija u svakom koraku generira novu populaciju iz stare tako da s jednakom vjerojatnošću odabere iz stare populacije k jedinki onoliko puta kolika je veličina te populacije; uspoređuje odabranih k jedinki i nad najboljom se dalje obavlja reprodukcija i djeluju genetski operatori (križanje ili mutacija).

Eliminacijska selekcija bira loše jedinke (kromosome) koje eliminira, a njih nadomještaju djeca preostalih jedinki nad kojima djeluju genetski operatori. Postoji i eliminacijska selekcija bez duplikata kojom se provjerava postoji li jedinka slična novonastaloj jedinci. Ako postoji, postupak provjeravanja nastavit će se sve do pojave jedinke koja više nije duplikat neke druge jedinke. Tako se može povećati učinkovitost genetskog algoritma.

Nakon što se nad jedinkama obavi selekcija, slijede križanje (binarni operator) i mutacija (unarni operator).

2.2.2 Križanje

Križanjem dvaju roditelja nastaju djeca koja imaju neka svojstva svojih roditelja. Križanjem se loša ili dobra svojstva roditelja prenose na djecu. Za binarni prikaz rješenja postoje križanja s jednom točkom prekida ili s više njih. Dijele se na jednoliko

(uniformno), segmentno i miješajuće. Uniformno križanje primjenjuje se kad je populacija mala. Segmentno i miješajuće (križanja s više točaka prekida) koriste se kada je populacija velika. Kod miješajućeg križanja pomiješaju se bitovi roditelja, nakon toga se obavlja klasično križanje s dvije ili više točaka prekida, a zatim se pomiješani bitovi roditelja vrate na mjesta na kojima su bili [1].

2.2.3 Mutacija

Mutacija se obavlja nad jednom jedinkom i njome nastaje nova jedinka s promijenjenim genima. Kao i u prirodi, vjerojatnost mutacije puno je manja od vjerojatnosti križanja. Neke vrste mutacija su jednostavna, miješajuća, potpuna miješajuća, invertirajuća miješajuća.

2.3 Parametri genetskih algoritama

Svaki genetski algoritam mora imati parametre, među kojima su veličina populacije, broj iteracija (generacija) i vjerojatnost mutacije.

Kod eliminacijskog genetskog algoritma zadaje se i broj jedinki za eliminaciju, a kod generacijskog vjerojatnost križanja. Poznate su preporučene vrijednosti tih parametara ovisno o tome radi li se o maloj ili velikoj populaciji. Imaju vrlo veliki utjecaj na učinkovitost algoritma [3].

2.4 Struktura jednostavnog genetskog algoritma

```
Genetski algoritam{
t=0;
generiraj početnu populaciju potencijalnih rješenja P(0);
sve dok nije zadovoljen uvjet završetka evolucijskog procesa {
t=t+1;
selektiraj P'(t) i djecu spremi u P(t);
mutiraj jedinke iz P(t); }
ispiši rješenje; }
```

Slika 2.1 Struktura jednostavnog genetskog algoritma

U početku se, najčešće slučajnim odabirom, generira početna populacija mogućih rješenja. Nad generiranom se populacijom obavlja selekcija kojom se izabiru najbolje jedinke koje dalje sudjeluju u procesu križanja kako je prikazano na slici 2.1. Mutacijom se, slučajnom promjenom gena, mijenjaju svojstva jedinke čime se postiže sve veća prosječna dobrota populacije. Taj se postupak obavlja sve dok ne istekne određeno vrijeme ili se ne zadovolji uvjet završetka evolucijskog procesa.

3. Raspoređivanje poslova u višeprocesorskim sustavima

Danas se, sve većim i bržim razvojem tehnologije, sve češće koriste višeprocesorski sustavi. Oni su u mnogo čemu bolji od sustava sa samo jednim procesorom; jedna od prednosti je brže izvođenje programa. Poslovi na koje je program podijeljen mogu se izvršavati paralelno (više poslova istodobno obavlja se na više procesora).

3.1 Metode rješavanja problema raspoređivanja poslova

Postoje mnogi algoritmi koji rješavaju problem rasporeda poslova. Već je prije spomenuto kako je taj problem u takvim sustavima deterministički NP – potpuni problem te se za njegovo rješavanje moraju koristiti heurističke i metaheurističke metode [8]. NP potpuni problemi imaju svojstvo da im se rješenja mogu brzo provjeriti, a trenutne metode pronalaženja rješenja nisu skalabilne. Kad se radi o heurističkim metodama, tada se program najčešće podijeli u više poslova kojima se odredi prioritet. Poslovi se poslože u listu ili red po prioritetu od najvećeg prema najmanjemu. Kad neki procesor završi s izvršavanjem posla, posao na početku reda, tj. posao s najvećim prioritetom dodjeljuje se tom slobodnom procesoru. U metaheurističke metode ubrajaju se i genetski algoritmi koji u potpunosti pretražuju prostor rješenja i pronalaze optimalno rješenje (ili bar rješenje blizu optimalnog) u svim situacijama [3]. To je usmjereno slučajno pretraživanje prostora rješenja (engl. *guided random search techniques).*

Osim toga postoje i deterministički algoritmi kod kojih su sve informacije o poslovima, njihovim međusobnim vezama, vremenima izvršavanja i broju procesora a priori poznate ili nedeterministički algoritmi kod kojih barem jedna od tih informacija nije unaprijed poznata (pa se određuje “u hodu“) i može se promijeniti. Za rješavanje ovog problema može se koristiti i A* algoritam, zatim simulirano kaljenje... [4] Također se koriste i različite vrste genetskih algoritama (ovisno o tome kako se prikazuju kromosomi. Kromosomi mogu biti predstavljeni u obliku binarne matrice fiksnih dimenzija. Rješenje pomoću takvog genetskog algoritma nije osobito uspješno, jer se može dogoditi da se neki poslovi uopće ne izvrše. Zatim je predstavljeno rješenje u kojem su kromosomi prikazani u obliku nizova znakova različite duljine koji prikazuju redoslijed izvršavanja poslova za svaki procesor. Problem se može riješiti i pomoću FSG-a (engl. *full-search genetic algorithm*) ili pomoću CGL-a (engl. combined-genetic list algorithm) [5] ili pomoću inačice genetskog algoritma *Partitioned Genetic Algorithm (PGA*) [12]. Struktura kromosoma i određene restrikcije vezane za njegovu strukturu bitno utječu na kompleksnost genetskih operatora i učinkovitost algoritma u konvergiranju prema optimalnom rasporedu.

3.2 Prikaz problema

Program se dijeli na niz poslova. Potrebno je obaviti dva zadatka: odrediti kojem će se procesoru dodijeliti koji posao (engl. *matching*) i odrediti redoslijed izvršavanja svakog pojedinog posla (engl. *scheduling*). Poslovi se dodjeljuju procesoru tako da svaki posao bude izvršen u najkraćem mogućem vremenu. Cilj je najčešće postići

najkraće moguće vrijeme izvršavanja programa u višeprocesorskom sustavu. Pretpostavlja se da procesori mogu međusobno komunicirati.

Slijedne ovisnosti između pojedinih poslova prikazane su u direktnom acikličkom grafu (engl. *directed acyclic graph (DAG))* G=(V, E)=(S, E). Pretpostavlja se da je program podijeljen na |S| poslova. U acikličkom grafu poslovi su prikazani kružićem (čvor u grafu) pri čemu je s_i i – ti posao, $S=\{s_i \mid 0 \leq i < |S|\}$. Redoslijed izvršavanja poslova je u grafu prikazan strjelicama (bridovi u grafu). Važno je da se poslovi izvršavaju slijedno, tj. ako su dva posla (čvora) u grafu povezana bridom to znači da jedan od njih prethodi drugome - da bi se jedan posao koji slijedi izvršio, prije njega se mora izvršiti njegov prethodnik; poslovi nisu jednako važni. Višeprocesorski sustav na kojem se taj program izvršava sastoji se od |M| identičnih procesora, što znači da su njihove brzine procesiranja jednake [6]. M_j je j – ti procesor, $M=\{m_j \mid 0 \leq j < |M|\}$. Vremena izvršavanja poslova ne moraju biti jednaka. Vrijeme izvršavanja posla s_i na procesoru m_j je Tij, pri čemu je $i \in [\,0, |S|\,\rangle$, a $j \in [\,0, |M|\,\rangle$. Svakom je procesoru dodijeljen točno jedan posao, tj. na jednom se procesoru u određenom trenutku može izvršavati samo jedan posao (dva posla na istom procesoru ne mogu se izvršavati paralelno). Komunikacijski trošak (engl. *communication cost*) između dva posla dodijeljenih istom procesoru je nula. Nije dopušteno prekidanje ili rastavljanje poslova (engl. *nonpreemption).* Procesori su potpuno povezani i među njima nema komunikacijskog kašnjenja (vrijeme kašnjenja zbog komunikacije između poslova koji se izvršavaju na različitim procesorima). Jedan posao izvršava se samo na jednom određenom procesoru (ne pojavljuje se slučaj da je jedan isti posao dodijeljen dvama procesorima, ili više njih). Ukupno vrijeme potrebno programu za izvršavanje naziva se ukupna duljina rasporeda (engl. *makespan).* Cilj je minimizirati ukupnu duljinu rasporeda određujući koji će se posao kojem procesoru dodijeliti i definirajući redoslijed izvršavanja. Složenost problema ovisi o broju procesora i načinu na koji su poslovi (čvorovi u grafu) povezani.

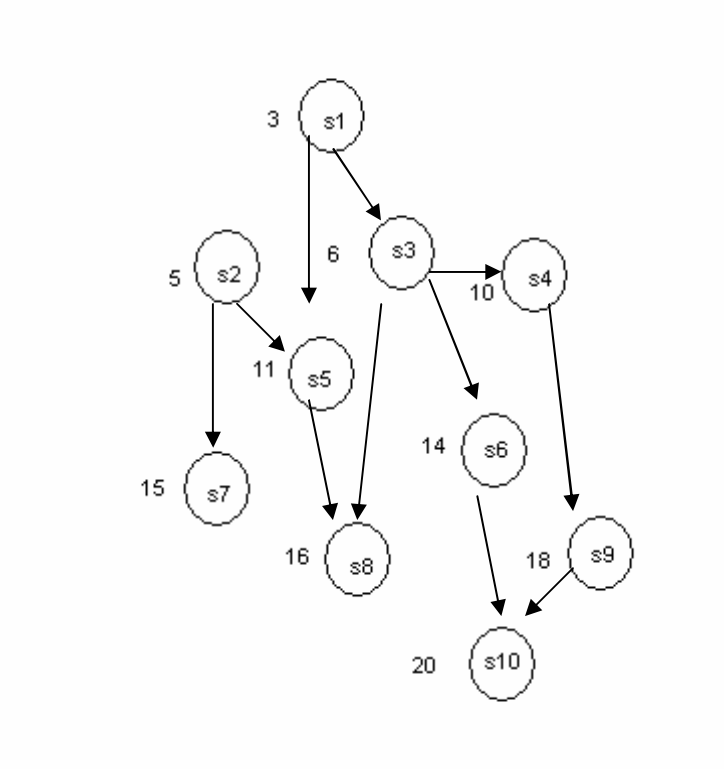
Tablica 3.1 Složenost problema rasporeda poslova

Broj procesora(***m***)	Vrijeme izvođenja posla (Ti)	Uvjet prethođenja	Složenost
Proizvoljan	Jednako	Stablo	O(***n***)
2	Jednako	Proizvoljan	O(***n^2***)
Proizvoljan	Jednako	Proizvoljan	NP-teško
Fiksan (***m***>=2)	T*i*=1 ili 2 za sve *i-eve*	Proizvoljan	NP-teško
Proizvoljan	***Proizvoljno***	***Proizvoljan***	***Izrazito NP-teško***

Iz tablice 3.1 vidi se složenost problema rasporeda poslova u ovisnosti o broju procesora, vremenu izvođenja i povezanosti poslova (uvjetu prethođenja) [7]. Ako je npr. broj procesora fiksan, a pri tome veći ili jednak dva te ako je vrijeme izvođenja

posla s_i T_i =1 ili T_i =2 za sve poslove i ako je način povezanosti (uvjet prednosti) poslova na grafu proizvoljan, tada se taj problem ubraja u NP-teške probleme. S druge strane, ako je broj procesora proizvoljan, ako je vrijeme izvođenja svih poslova jednako (što pojednostavljuje slučaj), složenost je O(n), gdje je n broj poslova.

Primjer: Sustav se sastoji od 5 procesora, a program je podijeljen na 10 poslova. Graf kojim bi se prikazala povezanost poslova prikazan je na slici 3.1. Radi se o ograničenjima u redoslijedu [11] (engl. *precedence constrains)* – ako posao s_4 ima prednost nad poslom s_8 to znači da se posao s_8 ne smije početi izvršavati prije nego što se izvrši posao s_4, prema tome, uvjet prethođenja poslova određuje redoslijed njihovog izvršavanja. Ako poslovi nisu povezani (nema strjelice između njih u grafu), tada za njih ne vrijedi uvjet prethođenja i oni se mogu izvršavati proizvoljnim redoslijedom. U ovom slučaju, kao što se vidi na grafu sa slike 3.1, posao prethodnik ima manji indeks od svog posla sljedbenika.



Slika 3.1 Direktan aciklički graf rasporeda poslova

Na slici 3.1 prikazan je direktan aciklički graf rasporeda poslova. Graf ima deset čvorova koji predstavljaju poslove i oni su međusobno povezani strjelicama (po tome se može vidjeti koji posao prethodi kojem poslu). Brojevi s lijeve strane svakog čvora predstavljaju vrijeme izvođenja pojedinog posla.

Ovi se grafovi mogu zapisati u različitim formatima, npr. u formatu *Prototype Standard Task Set* koji se sastoji od dijela koji prikazuje povezanost poslova (engl. *task graph part)* i dijela s informacijama (engl. *information part)* koji počinje sa znakom "#". Prvi dio tog formata prikazuje u svakom retku indeks posla, vrijeme izvršavanja posla, broj prethodnika i indekse prethodnika, a drugi, informacijski dio, sadrži informacije o imenu grafa, o makimalnom i minimalnom broju prethodnika, o

duljini kritičnog puta [7]... Problem je pojednostavljen, jer je komunikacijski trošak jednak nuli; kad komunikacijski trošak ne bi bio jednak nuli, format bi se promijenio (imao bi informacije o komunikacijskom trošku između pojedinih poslova).

Drugi format koji se danas češće koristi i koji je jednostavniji je *Prototype Standard Task Graph Set form [7]*. Taj format prikazan je za problem rasporeda 10 poslova na 5 procesora (graf na slici 3.1) na slici 3.2.

```
10   5
 1   3   0
 2   5   0
 3   6   1   1
 4  10   1   3
 5  11   2   1   2
 6  14   1   3
 7  15   1   2
 8  16   2   3   5
 9  18   1   4
10  20   2   6   9
11   0   2   9  10
```

Slika 3.2 Prototype Standard Task Graph Set oblik

U prvom retku broj deset označava broj poslova, a broj pet broj procesora na koje se ti poslovi razdjeljuju. U drugom retku broj jedan označava posao s indeksom 1, idući broj prikazuje vrijeme izvršavanja tog posla, broj 0 označava da posao s_1 nema prethodnika. U četvrtom retku prikazan je posao s indeksom 3 (s_3) kojem je vrijeme izvršavanja 6, taj posao ima jednog prethodnika i to je posao s indeksom 1 (s_1). U šestom retku prikazan je posao s_5, njegovo vrijeme izvršavanja je 11, ima dva prethodnika, a to su poslovi s_1 i s_2 (što se vidi iz zadnjih dvaju brojeva u istom retku). Tako je sve i za ostale retke. Zadnji redak predstavlja pomoćni čvor (engl. *exit dummy node)* kojem je vrijeme izvršavanja 0 i ima dva prethodnika – poslove s_9 i s_{10} .

4. Primjena genetskog algoritma na problem rasporeda poslova

Postoji mnogo različitih prilagođenih genetskih algoritama koji se koriste za rješavanje problema raspoređivanja poslova u višeprocesorskom sustavu. U nekim primjerima poslovi i procesori mogu predstavljati pacijente i bolničku opremu, zrakoplove i pilote, razrede i profesore, vlakove i strojovođe, gradove i trgovačkog putnika, poslove i strojeve, pozive i usluge... Sve su to problemi rasporeda koji se moraju riješiti. U nastavku je opisana implementacija genetskog algoritma koja je prikazana u znanstvenom članku [6], uz čiju se pomoć vrlo učinkovito može riješiti problem rasporeda poslova na više procesora.

4.1 Prikaz rješenja

Rješenje će biti prikazano u obliku dvodimenzionalnog polja. Genetski algoritam prilagođen konkretnom problemu rasporeda poslova prikazan je na slici 4.1 [6].

```
Genetski algoritam {
inicijalizacija (P(0)); //inicijalizacija početne populacije
for (i=0;i<l;i++) { //i - broj iteracija (generacija)
slučajno odaberi tri jedinke iz P(i);
odredi najgore jedinke za eliminaciju;
križaj preostale jedinke;
mutiraj dijete;
evaluiraj dijete;
zamijeni evaluiranu jedinku novim djetetom; } }
```

Slika 4.1 Modificirani genetski algoritam

Već je spomenuto kako je S broj poslova, a M broj procesora. U ovom je slučaju broj procesora fiksan (jedan od problema može biti i pronalaženje minimalnog broja procesora tako da vrijeme izvođenja ne prijeđe zadanu granicu i ne bude dulje o kritičnog puta u direktnom acikličkom grafu).

4.2 Inicijalizacija populacije

Na početku se slučajno generira početna populacija. Slučajno se odabire jedan od M procesora i dodaje mu se jedan posao iz liste poslova sortiranih uzlazno prema indeksima. Pseudokod inicijalizacije prikazan je na slici 4.2.

```
inicijalizacija(P(0)) {
for(i=0;i<n;i++) {  //n - veličina populacije
for(j=1;j<=M;j++) {  //M - broj populacija
skup poslova koje radi m - ti procesor proširi poslom s(j); }
// r - slučajno izabran cijeli broj iz intervala [1, M]
evaluacija(i - ta jedinka); }
```

Slika 4.2 Inicijalizacija populacije

Kromosom (moguće rješenje problema) predstavljen je listom koja za svaki procesor (u ovom primjeru, za njih 5) sadrži indeks posla koji se na njemu obavlja (brojevi su indeksi poslova). Primjer je dan na slici 4.3. S obzirom da su poslovi za određeni procesor sortirani uzlazno po indeksima odmah je zadovoljen uvjet prethođenja izvršavanja poslova na istom procesoru, ali nije uzet u obzir uvjet prethođenja poslova koji se izvršavaju na različitim procesorima niti je uzeto u obzir vrijeme izvršavanja pojedinog posla. Prema tome, ovaj kromosom ne predstavlja optimalan raspored poslova, jer nije u potpunosti zadovoljen uvjet njihovog prethođenja (prema grafu na slici 3.1). Za prikaz kromosoma koristi se primjer definiran u poglavlju 3.2 (graf poslova).

A[1]	2	6	8
A[2]	1	3	
A[3]	4	5	
A[4]	7		
A[5]	9	10	

Slika 4.3 Prikaz kromosoma

4.3 Selekcija

Za ovaj problem koristit će se eliminacijska turnirska selekcija. Ona bira 3 jedinke (kromosoma) i eliminira najlošiju među njima, a od preostalih dviju generira se kromosom dijete koje će imati naslijeđena dobra svojstva. Pseudokod selekcije prikazan je na slici 4.4.

```
Genetski algoritam {
inicijalizacija(P(0));
for (i=0;i<l;i++) {
slučajno odaberi tri kromosoma iz P(i);
označi najgori kromosom za eliminaciju;
križaj preživjele kromosome; mutiraj dijete;
evaluiraj dijete;
zamijeni eliminirani kromosom novin djetetom; } }
```

Slika 4.4 Pseudokod selekcije

4.4 Križanje

Pseudokod križanja prikazan je na slici 4.5. Ovaj algoritam križa dva slučajno odabrana kromosoma i stvara treći (radi se s indeksima poslova koji su poredani uzlazno). Uvjet prethođenja mora biti zadovoljen.

```
Križanje (A;B) {
if (A==B) {  //eliminacija duplikata
if (A je bolje rješenje)  mutiraj(B);  //elitizam
else mutiraj(A);
slučajno odabiranje djeteta;
return }
for(i=1;i<=M;i++) {
if (posao s(i) isti za oba roditelja) { kopiraj s(i) na isto mjesto u djetetu }
else { slučajno kopiraj posao jednog od roditelja; } } }
```

Slika 4.5 Pseudokod križanja

Križanje se odvija između dvaju roditelja (A i B, slika 4.6). Ako su indeksi poslova kod oba roditelja jednaki, tada se u dijete (C) kopira taj isti gen na isto mjesto, tj. na istom će se mjestu nalaziti taj jedini indeks posla. Ukoliko su indeksi poslova roditelja različiti, u dijete se kopira indeks odabran proizvoljno od jednog roditelja. Ako su indeksi isti, a A je bolje rješenje, mutirat će se gen drugog roditelja (B). Ako nisu isti, mutiraju se geni prvog roditelja (A), a dijete se generira slučajno [6].

A[1]	2	6	8
A[2]	1	3	
A[3]	4	5	
A[4]	7		
A[5]	9	10	

B[1]	1	3	4
B[2]	2		
B[3]	5	8	
B[4]	6	9	
B[5]	7	10	

C[1]	1	3	4
C[2]	2		
C[3]	5	8	
C[4]	6		
C[5]	7	10	

Slika 4.6 Primjer križanja

Najjednostavnije je za križanje uzeti uniformno križanje. Slučajno se generira maska koja određuje iz kojeg će se roditelja gen kopirati u dijete. Maska je niz nula i jedinica i duljine je jednake duljini roditelja. Npr. ako je maska za neki posao jednaka jedinici, to znači da će se u dijete kopirati gen iz prvog roditelja, a ako je nula, to znači da će se u dijete kopirati gen iz drugog roditelja. Ako se radi s maskom, tada je prikaz kromosoma jednostavniji – složena je lista koja ima poslove poredane po indeksima uzlazno, a roditelje (kao i dijete nakon križanja) čini niz indeksa procesora (u ovom je primjeru to slučajno izabrani cijeli broj iz intervala [1,5]). Iz tog se oblika dijete ponovo može prebaciti u prijašnji oblik (kao npr. na slici 4.8).

4.5 Mutacija

Nakon križanja nad kromosomima se može provesti i mutacija. Ona se provodi nad jednim roditeljem ako su oba roditelja jednaka. Tako se eliminiraju duplikati. Mutacija mijenja vrijednost gena s nekom unaprijed određenom vjerojatnošću (koja se može mijenjati ili biti fiksna).

```
Mutacija {
generiraj dva slučajna broja iz intervala [1, M], mora biti zadovoljeno
    a) r#b i,
    b) red koji stoji uz procesor čiji je indeks r nije prazan
iz reda r slučajno odaberi posao i ukloni ga iz reda q; }
```

Slika 4.7 Pseudokod mutacije

Npr. r=3 i q=2 i slučajno je odaberen posao s indeksom 4.

A[1]	2	6	8
A[2]	1	3	
A[3]	4	5	
A[4]	7		
A[5]	9	10	

Slika 4.8 Prije mutacije

A[1]	2	6	8
A[2]	1	3	4
A[3]	5		
A[4]	7		
A[5]	9	10	

Slika 4.9 Poslije mutacije

Slučajno se odaberu dva broja: p i q iz intervala [1,M] (M je broj procesora). Kako bi se mutacija uspješno obavila, ti brojevi moraju zadovoljavati određene uvjete kako je prikazano u algoritmu na slici 4.7. Prvi izabrani broj označava nad kojim se genom primjenjuje mutacija. Posao će se s procesora kojem odgovara taj gen prebaciti na drugi procesor (slučajno odabran, drugi broj - vrijednost q). Mutirani kromosom prikazan je na slici 4.9 [6].

4.6 Evaluacija

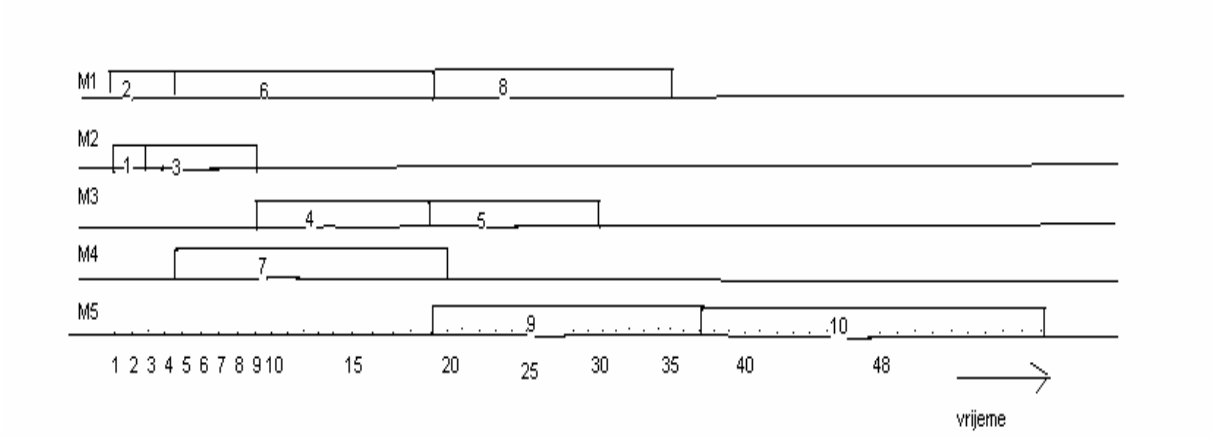
```
Evaluacija {
for(i=1;i<=M;i++) {  //resetiraj FTP svim procesorima
//FTP[i] je zavrsno vrijeme posljednjeg posla na procesoru i
for(i=1;i<=S;i++) {
FTP[m]+=trajanje(s(i));
for(j=1;j<=M;j++) {
if (j==m) continue;
  // u ovoj je liniji ocuvana relacija prethođenja
if(((s(x) je element od j)<s(j)) && (FTP[j]>pom))
  // x je najveci indeks od poslova na procesoru m
  //pri cemu je x<j
FTP[m]=FTP[j]+s(i); } }
FT=max(i je elment od [1,M]) {FTP[i]}; }
  //FT je zavrsno vrijeme svakog posla
```

Slika 4.10 Pseudokod evaluacije

Na slici 4.10 prikazana je evaluacija [6]. Radi se o računanju završnog vremena rasporeda.

Funkcija cilja određena je negativnom vrijednošću završnog vremena rasporeda koje predstavlja kromosom, jer se koristi turnirska eliminacijska selekcija. Završno vrijeme rasporeda je vrijeme kad se izvrši posljednji posao (engl. *finishing time, FT) [10],* a to je isto što i ukupna duljina rasporeda (engl. *makespan).* Za sva moguća rješenja (kromosome) može se, ovisno o vremenu trajanja pojedinog posla i pazeći na uvjet prethođenja, odrediti vremenski raspored poslova unutar pojedinog procesora. Završno vrijeme rasporeda nastoji se minimizirati – najbolji će biti onaj kromosom čiji raspored ima najmanje vrijeme izvršavanja poslova [9]. Raspored poslova može se prikazati Ganttovim dijagramom [10].

Raspored poslova za kromosom sa slike 4.3 prikazan je Ganttovim dijagramom na slici 4.11.



Slika 4.11 Gantogram rasporeda poslova

Iz slike 4.11 vidi se da završno vrijeme rasporeda za ovaj slučaj iznosi čak 57 vremenskih jedinica. Poslovi s_7, s_4 i s_9 mogu se početi izvršavati tek kad se do kraja izvrše poslovi s_2, s_3, odnosno s_7. Poslovi s_1 i s_2 mogu se, budući da nemaju prethodnike s kojima su povezani, izvršavati proizvoljnim redoslijedom pa se oba izvršavaju odmah na početku.

4.7 Utjecaj genetskih parametara na učinkovitost algoritma

Eksperimentalno je pokazano kako na učinkovitost genetskih algoritama (o rješavanju kojeg god se problema radilo) najviše utječu njegovi parametri, a to su veličina populacije, vjerojatnost mutacije i broj iteracija (generacija). Dokazano je da se za veći broj iteracija postižu bolja rješenja, ali samo ako je i populacija velika.

Za primjer su uzeta dva problema [6]. Jedan je problem raspodjele 452 posla na 20 procesora, a drugi 473 poslova na 4 procesora. Za prvi se problem optimalni raspored postiže u 537 vremenskih jedinica. Za vjerojatnost mutacije uzeta je vrijednost 1%, veličina populacije je 20 jedinki, a broj iteracija 10000. Za nekoliko desetaka pokusa, u 15% slučajeva dostignut je lokalni optimum.

Optimalni raspored za drugi problem uz pomoć jednostavnog genetskog algoritma dobije se, u najgorem slučaju, za 1187 vremenskih jedinica, a u najboljem za 1182

vremenske jedinice. Prosječno rješenje dobije se za 1184 vremenskih jedinica i to za populaciju od 50 jedinki, za vjerojatnost mutacije 0.01 ili 1% i za 30000 iteracija [6].

5. Zaključak

Problem rasporeda poslova rješava se heurističkim metodama, jer on pripada klasi NP problema. Glavno svojstvo problema te klase je da još uvijek nije poznato postoji li upotrebljiv algoritam polinomijalne složenosti koji pronalazi optimalno rješenje. Za njegovo rješavanje postoje mnogi algoritmi, a jedan od njih je i genetski algoritam.

Pronalaženje optimalnog rješenja (ili bar rješenja blizu optimalnog) za raspored poslova u višeprocesorskom sustavu znatno je pojednostavljeno ako je komunikacijski trošak između dvaju poslova koji se izvršavaju na istom procesoru jednak nuli, kad nema komunikacijskog kašnjenja i kad su brzine procesiranja svih procesora jednake. Najbolje rješenje predstavlja kromosom s najvećom vrijednošću funkcije dobrote (u ovom je slučaju njena vrijednost negativna vrijednost završnog vremena rasporeda), tj. kromosom čiji raspored ima najmanje vrijeme izvršavanja poslova.

Kvaliteta dobivenog rješenja ovisi o genetskim parametrima; za veliku populaciju, uz veliki broj iteracija postiže se najbolje rješenje. Također, za isti se broj iteracija i veću populaciju, najbolje rješenje postiže uz manju vjerojatnost mutacije.

6. Literatura

[1] Marin Golub, *Skripta 1. dio: Genetski algoritmi*,
http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, 10. 4. 2008.

[2] Vlatko Čerić, *Genetski algoritmi,*
http://web.efzg.hr/dok/INF/Ceric/spo/(2d)_genetski_algoritmi.pdf, 10. 4. 2008.

[3] Genetski, 22.4.2008., *Genetski algoritmi – predavanje,*
http://www.fer.hr/_download/repository/GApredavanje.pdf, 10. 4. 2008.

[4] Goran Radanović, *Pregled heurističkih algoritama,* 10. 4. 2008.,
http://www.zemris.fer.hr/~golub/ga/studenti/seminari/2007_radanovic/ostali.html

[5] Michael Rinehart, Vida Kianzad, Shuvra S. Bhattacharyya, *A Modular Genetic Algorithm for Scheduling Task Graphs,*
http://www.ece.umd.edu/DSPCAD/papers/rine2003x1.pdf, 10. 4. 2008.

[6] Marin Golub, Suad Kasapović, *Scheduling multiprocessor tasks with genetic algorithms,* 30. 10. 2001.,
http://www.zemris.fer.hr/~golub/clanci/ai2002.pdf, 10. 4. 2008.

[7] Kasahara Laboratory, Waseda University, *Standard Task Graph Set*
http://www.kasahara.elec.waseda.ac.jp/schedule/, 10. 4. 2008.

[8] Lee Wang, Howard J. Siegel, Vwani P. Roychowdhury, Anthony A. Maciejewski, *Task Matching and Scheduling in Heterogeneous Computing Environments using a Genetic-Algorithm-Based Approach, 10. 4. 2008.*
http://cisr.nps.edu/downloads/MSHN/jpdc_11_97_ga_leew.pdf

[9] Ognjen Fabris, Diplomski rad br. 1523., *Primjena genetskih algoritama za raspoređivanje poslova u višeprocesorskom sustavu,* 10. 4. 2008.,
http://www.zemris.fer.hr/~golub/ga/studenti/fabris/Diplomski%20rad%20broj%201523.html

[10] Sanjay R Sutar, Jyoti P. Sawant, Jyoti R. Jadhav,
Task Scheduling For Multiprocessor Systems Using Memetic Algorithms,
http://www.comp.brad.ac.uk/het-net/tutorials/P27.pdf, 10. 4. 2008.

[11] Kasahara Laboratory, Waseda University *Optimal Schedules for Prototype Standard Task Graph Set*, 10. 4. 2008.,
http://www.kasahara.elec.waseda.ac.jp/schedule/optim_pe.html

[12] Yi – Hsuan Lee, Cheng Chen, *A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems,* 10. 4. 2008.,
http://parallel.iis.sinica.edu.tw/cthpc2003/papers/CTHPC2003-18.pdf

7. Sažetak

Genetski se algoritmi danas koriste za rješavanje mnogih problema. Prilikom rješavanja nekog problema uz pomoć genetskog algoritma potrebno je modificirati algoritam i prilagoditi ga danom problemu (npr. podešavanjem njegovih parametara kako bi se brže konvergiralo prema optimalnom rješenju i sl.). Također je potrebno odrediti kako će se problem prikazati (odrediti način kodiranja kromosoma). Na početku se odabire početna populacija te se nad njom obavlja selekcija (u ovom slučaju turnirska eliminacijska selekcija). Nakon selekcije ostat će jedinke s vrlo dobrim svojstvima te će se nad njima obavljati križanje (u ovom slučaju uniformno križanje) i, po potrebi, mutacija. Proces će se ponavljati sve dok se ne zadovolji zadani uvjet zaustavljanja. Kad je on zadovoljen, rješenje koje se dobije je optimum ili vrlo blizu optimuma.