SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1886

Razvojna okolina za ispitivanje postupaka raspoređivanja

Ines Bonačić

Zagreb, travanj 2011.

Razvojna okolina za ispitivanje postupaka raspoređivanja

Sažetak

Opisana je klasična teorija raspoređivanja, svojstva poslova i strojeva te kriteriji izrade rasporeda. U praktičnom dijelu rada ostvaren je programski sustav za simulaciju različitih okolina raspoređivanja koji uključuje okoline s jednim strojem, paralelnim jednolikim i nesrodnim strojevima te proizvoljnu obradu. Sustav ima nekoliko ocjenjivača za postojeće kriterije raspoređivanja, a postoji i mogućnost uključivanja proizvoljne komponente ocjenjivača rasporeda. Kao primjeri proizvoljnih raspoređivača ostvareni su raspoređivač temeljen na genetskom algoritmu i raspoređivač temeljen na funkciji prioriteta u okolini nesrodnih strojeva.

Ključne riječi: teorija raspoređivanja, okolina strojeva, okolina poslova, ocjenjivač rasporeda, raspoređivač, kriterij raspoređivanja, genetski algoritam, okolina nesrodnih strojeva, okolina jednolikih strojeva, raspoređivač temeljen na funkciji prioriteta

Development enviroment for testing scheduling algorithms

Abstract

A classic scheduling theory, job and machine properties and scheduling criteria is described. In a practical part of work, software system for simulating different scheduling enviroments which includes single machine enviroment, uniform and unrelated machines enviroments as well as open shop enviroment is accomplished. The system has several evaluators for existing scheuling criteria, and there is also a possibility of including arbitrary scheduler evaluator components. As an example of an arbitrary schedulers, a scheduler based on genetic algoritam and a scheduler based on priority function in unrelated neviroment are accomplished.

Keywords: scheduling theory, job enviroment, machine enviroment, schedule evaluator, scheduler, scheduling criteria, genetic algorithm, unrelated machines enviroment, uniform machines enviroment, priority scheduler

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Raspoređivanje je proces u kojemu se zauzimaju sredstva za aktivnosti tijekom određenog vremenskog razdoblja. Rješenje problema raspoređivanja je raspored kojim je definirano kada se koja aktivnost obavlja i na kojem sredstvu. Proces raspoređivanja redovito se koristi u mnogim proizvodnim i uslužnim djelatnostima. Neki od problema raspoređivanja koje susrećemo su: problem raspoređivanja poslova u višeprocesorskom sustavu, problem rasporeda sati, raspored poslova u tvornici, raspodjela radne snage.

Problem raspoređivanja često, u ovisnosti o veličini problema i zadanim ograničenjima, spada u klasu *NP* potpunih problema. Za takve probleme metoda iscrpne pretrage nije prikladna jer je vrijeme potrebno za takvu vrstu pretrage neprihvatljivo dugo. Zbog toga se problem raspoređivanja često rješava heurističkim metodama pretrage. Iako se za takve vrste pretrage ne može garantirati da će pronađeno rješenje biti optimalno, moguće je dobiti dovoljno dobra rješenja u prihvatljivom vremenu. Neki od primjera heurističkih algoritama korištenih za probleme raspoređivanja su: genetski algoritam, algoritam mravlje kolonije, pohlepni algoritmi, lokalno pretraživanje i tabu pretraživanje.

U ovom radu opisana je primjena genetskog algoritma na problem raspoređivanja. Genetski algoritam (GA) je stohastička, heuristička metoda optimiranja koja oponaša proces evolucije u prirodi postupcima selekcije, križanja i mutacije. Najčešće se koristi kada je prostor pretrage osobito velik, kao primjerice u problemu raspoređivanja.

Programski sustav za simulaciju okolina raspoređivanja sastoji se od komponente koja učitava ulazne podatke, komponente koja predstavlja model simulatora, te ocjenjivača kvalitete rasporeda. Različite komponente ocjenjivača povezane su preko sučelja. Ulazni podaci definiraju okolinu poslova i okolinu strojeva. Model simulatora sastoji se od skupa razreda u kojima je opisan problem raspoređivanja, razreda koje opisuju raspored, metoda za učitavanje ulaznih podataka, te sučelja koja omogućavaju lako dodavanje promjenjivih komponenti u sustav kao na primjer ocjenjivača rasporeda.

U drugom poglavlju opisan je rad genetskog algoritma. Na početku rada stvara se početna populacija jedinki koja se može generirati na više različitih načina. Nakon generiranja početne populacije obavlja se selekcija jedinki kojom se izabiru jedinke iz trenutne generacije koje će se prenijeti u sljedeću generaciju. Uloga selekcije je da se sačuva dobar genetski materijal. Postoji više načina selekcije, a opisani su najčešći. Nakon selekcije nad izabranim jedinkama obavljaju se genetski operatori križanja i mutacije. Algoritam je iterativan, izvršava se dok nije zadovoljen neki od uvjeta zaustavljanja. Uvjeti zaustavljanja mogu biti maksimalan broj generacija, maksimalno dozvoljeno vrijeme, pronalazak rješenja koji zadovoljava zadane kriterije, ne postižu se bolji rezultati kroz nekoliko iteracija. Velik utjecaj na rezultat genetskog algoritma ima odabir parametara: veličina populacije, broj iteracija i vjerojatnost mutacije.

U trećem poglavlju formalno je opisan problem raspoređivanja. Opisana su svojstva poslova, definirane najčešće okoline strojeva, te svojstva zadataka. Nakon izvođenja programa i prikupljanja izlaznih veličina u sustavu slijedi vrednovanje rasporeda. Postoji više kriterija po kojima se može vrednovati raspored na temelju izlaznih veličina. Izlazne veličine i kriteriji vrednovanja, te uvjeti i postupci raspoređivanja također su objašnjeni u ovom poglavlju.

U četvrtom poglavlju su opisani ulazni podaci, objašnjen je model simulatora, te implementirani načini ocjenjivanja kvalitete rasporeda.

U petom poglavlju opisano je korištenje simulatora, opisan je genetski algoritam koji rješava problem raspoređivanja, te je ispitan utjecaj parametara na rad genetskog algoritma. Također, u petom poglavlju, dan je opis raspoređivača temeljenog na pravilima, te su rezultata tog raspoređivača uspoređeni s rezultatima dobivenim genetskim algoritmom.

U šestom poglavlju dan je zaključak.

2. Genetski algoritam

Genetski algoritam (GA) je heuristička metoda optimiranja koja oponaša proces evolucije u prirodi, moguće ga je primijeniti na širok skup problema. Obično se primjenjuje za probleme za koje nije moguće naći optimalno rješenje u prihvatljivom vremenu. Za takve probleme genetski algoritam može naći dovoljno dobra rješenja u prihvatljivom vremenu.

Genetski algoritam je iterativan algoritam pretraživanja. Na početku rada stvara se početna populacija jedinki. U svakoj iteraciji se nad jedinkama iz populacije izvode genetski operatori. Svrha operatora je usmjeriti pretragu prema boljim rješenjima i izbjeći ulazak u lokalni optimum.

Prvi korak algoritma je stvaranje početne populacije jedinki. Nakon stvaranja početne populacije jedinki, selekcijom se biraju jedinke koje sudjeluju u rekombinaciji (križanje) i mutaciji. Uloga operatora križanja je lokalno pretraživanje. Mutacija u pretraživanje unosi raznolikost u populaciju jedinki i obavlja ulogu potpunog pretraživanja. Nakon generiranja potomaka obavlja se odabir preživjelih jedinki na temelju vrijednosti evaluacijske funkcije. Veličina populacije ostaje konstantna iz generacije u generaciju. Moguće je unaprijed odrediti broj generacija ili se može zadati uvjet zaustavljanja. Najbolja jedinka u posljednjoj generaciji predstavlja optimalno rješenje. Slijedi pseudokod jednostavnog genetskog algoritma:

```
genetski_algoritam
{
        kreiraj pocetnu populaciju;
        evaluiraj svaku jedinku;
        sve dok nije zadovoljen uvjet zaustavljanja
        {
                selektiraj roditelje;
                krizaj roditelje;
                mutiraj dobivene potomke;
                evaluiraj novodobivene kandidate;
                izaberi rjesenja za slijedecu generaciju;
        }
}
```

Slika 2.1: Pseudokod genetskog algoritma

2.1. Parametri genetskog algoritma

Parametri genetskog algoritma su:

- Veličina populacije.
- Broj generacija (iteracija).
- Vjerojatnost križanja. Vjerojatnost križanja zadaje se kod generacijskih algoritama.
- Vjerojatnost mutacije.
- Broj jedinki za eliminaciju. Kod eliminacijskog genetskog algoritma zadaje se broj jedinki za eliminaciju

Optimiranje parametara genetskog algoritma je složen postupak i zahtjeva velik broj eksperimenata. Parametri genetskog algoritma imaju veliki utjecaj na učinkovitost algoritma.

2.2. Generiranje početne populacije

Početna populacija se sastoji od jedinki koje su najčešće generirane slučajnim odabirom. Početnu populaciju se može generirati i uniformno, tako da su sve jedinke identične. Može se na početku usaditi optimalno rješenje dobiveno nekom drugom metodom. Veličina populacije je zadana parametrom, a optimalna vrijednost ovisi o problemu, a obično sadrži nekoliko stotina ili tisuća jedinki.

2.3. Uvjet zaustavljanja

Uvjet zaustavljanja je kriterij prema kojem se odlučuje da li će genetski algoritam nastavi s pretraživanjem ili će se zaustaviti. Svaki od kriterija prema kojima se donosi odluka o zaustavljanju se provjerava nakon svake iteracije. Navedeni su neki od kriterija zaustavljanja:

Ako je postignut maksimalan broj generacija kojeg je korisnik zadao.

Ako je isteklo maksimalno dozvoljeno vrijeme.

Ako je pronađeno rješenje koje zadovoljava zadane kriterije.

Ako više nije moguće postići bolje rezultate.

2.4. Funkcija cilja

Uloga funkcije cilja je vrednovanje jedinke tako da svaka jedinka može biti rangirana. Bolja jedinka ima veću vjerojatnost preživljavanja i sudjelovanja u reprodukciji. Ukupna dobrota populacije D definirana je na sljedeći način:

$$D = \sum_{i=1}^{vel.pop} dobrota(v_i)$$

Prosječna dobrota populacije $\overline{D}$ definirana je ovako:

$$\overline{D} = \frac{D}{vel.pop}$$

2.5. Selekcija

Selekcijom se izabiru jedinke iz trenutne generacije koji će se prenijeti u sljedeću generaciju. Uloga selekcije je da se sačuva dobar genetski materijal, a loši geni odumiru. Prije nego što se jedinke prenesu u sljedeću generaciju primjenjuju se genetski operatori križanje i mutacija nad jedinkama. Jedinke dobivene primjenom genetskih operatora su one koje se prenose u sljedeću generaciju. Lošije jedinke također trebaju imati određenu vjerojatnost preživljavanja da bi izbjegli preuranjenu konvergenciju. Da bi sačuvali najbolju jedinku iz generacije, uvodimo postupak elitizma pa se najbolja jedinka prenosi nepromijenjena u sljedeću generaciju. Prema načinu na koji se genetski materijal prenosi u sljedeću generaciju postupci selekcije dijele se na *generacijske* i *eliminacijske*. Generacijski algoritam u svakoj iteraciji raspolaže s dvije populacije. Izabiru se najbolje jedinke koje će sudjelovati u reprodukciji i kopiraju se u drugu populaciju (međupopulacija). Zatim se međupopulacija treba nadopuniti jer je manja od populacije. Obično se popunjava duplikatima jedinki. Nad dobivenom populacijom se obavljaju operatori križanja i mutacije. Karakteristične vrste selekcija koje koristi generacijski GA su *jednostavna selekcija* i *turnirska selekcija*. Eliminacijskom selekcijom briše se određeni broj lošijih jedinki iz populacije, a populacija se popunjava jedinkama dobivenim reprodukcijom. Broj jedinki koje se eliminiraju naziva se mortalitet. Prema načinu odabira jedinki selekcije se dijele na *proporcionalne* i *rangirajuće*. Kod proporcionalnih selekcija vjerojatnost selekcije je proporcionalna s dobrotom jedinke. Proporcionalne selekcije dijele se na *jednostavnu selekciju* i *stohastičku selekciju*. Kod rangirajućih selekcija vjerojatnost odabira ovisi o poziciji jedinke u listi jedinki sortiranih prema dobroti. Rangirajuće selekcije dijele se na *turnirsku selekciju* i *sortirajuću selekciju*.

2.5.1. Jednostavna selekcija

Jednostavna selekcija je selekcija gdje je vjerojatnost da neka jedinka bude izabrana proporcionalna funkciji dobrote, tj. genetski materijal najboljih jedinki prenosi se u sljedeću generaciju.

2.5.2. Linearno sortirajuća selekcija

Kod linearno sortirajuće selekcije vjerojatnost odabira proporcionalna je poziciji jedinke u poretku jedinki sortiranih prema dobroti. Vjerojatnost selekcije računa se prema formuli:

$$p(i) = \frac{i}{\sum_{i=1}^{N} i} = \frac{2i}{N(N+1)}$$

Najbolja jedinka ima indeks N, a najgora ima indeks 1.

2.5.3. K-turnirska selekcija

K-turnirska selekcija u svakom koraku odabire K jedinki iz populacije. Koristi se jednovstavna selekcija K puta da bi se dobio podskup jedinki. Najbolja jedinka tog podskupa dalje sudjeluje u reprodukciji.

2.5.4. Selekcija najboljih

Selekcijom se odabire najbolja jedinka. Ako postoji više jedniki koje imaju jednaku dobrotu, odabire se jedna od njih slučajnim odabirom.

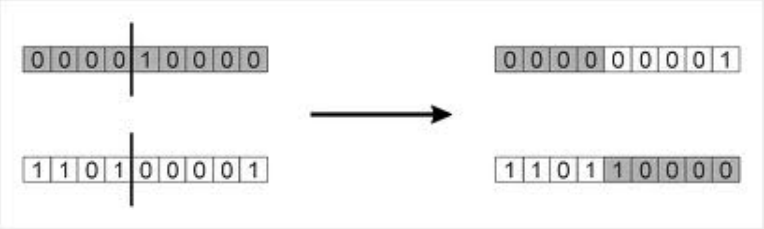
2.6. Genetski operatori

Nakon postupka selekcije, slijedi primjena genetskih operatora nad jedinkama neke populacije. Genetski operatori mijenjaju genetski materijal jedinki. Primjenom genetskih operatora na jedinke neke populacije stvaraju se nove jedinke ili se mijenjaju postojeće jedinke koje se prenose u sljedeću generaciju. Genetske operatore možemo podijeliti na binarne i unarne. Binarni genetski operator je križanje (engl. *crossover*). Križanjem dviju jedinki nastaje jedna ili dvije nove jedinke. Unarni genetski operator je mutacija. Mutacija se obavlja nad jednom jedinkom, a rezultat je izmijenjena jedinka.

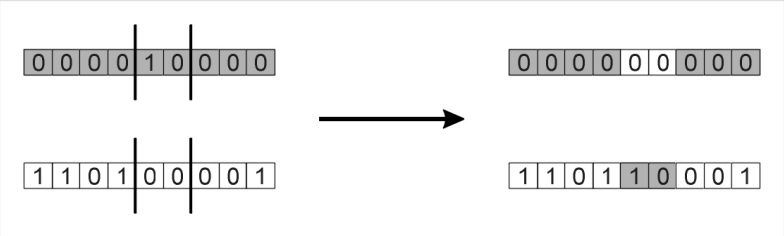
2.6.1. Križanje

Križanje je binarni genetski operator. Križanjem dviju jedinki (roditelji) nastaju nove jedinke (djeca). Dijete nasljeđuje karakteristike svojih roditelja. Postoji više tehnika križanja, a najčešće su:

Križanje s jednom točkom prekida (engl. *one-point crossover*). Odabire se jedna točka prekida koja se nalazi na istim pozicijama na oba roditelja. Svi podaci koji se nalaze iza te točke se međusobno zamjenjuju između roditelja i kao rezultat dobivamo djecu.



Križanje s više točaka prekida (engl. *multi-point crossover*). Odabiru se točke prekida na oba roditelja u kojima se nizovi dijele na podnizove . Križanje se obavlja alterniranjem podnizova i kao rezultat imamo dijete.



"Izreži i spoji" (engl. *"cut and splice"*). Ova varijanta križanja rezultira promjenom dužine novih jedinki (djece). Na svakom od roditelja se odabire točka prekida na proizvoljnim pozicijama.

Jednoliko križanje (engl. *uniform crossover*). Kod ovakvog križanja svakom genu roditelja pridružuje se fiksna vjerojatnost nasljeđivanja na djecu.

Aritmetičko križanje (engl. *arithmetical crossover*). Aritmetičko križanje je definirano za vektore realnih brojeva. Križanjem roditelja $\vec{x}$ i $\vec{y}$ nastaju djeca:

$$Dijete1 : \alpha\vec{x} + (1-\alpha)\vec{y}$$

$$Dijete2 : \alpha\vec{y} + (1-\alpha)\vec{x}$$

Parametar α je slučajni broj između nula i jedan.

Heurističko križanje (engl. *heuristic crossover*). Operator križanja koji koristi dobrotu oba roditelja da bi odredili smjer pretraživanja. Križanjem nastaje jedno dijete prema sljedećoj jednadžbi:

$$Dijete = r * (y_2 - y_1) + y_2$$

Kromosom y_2 ima veću dobrotu, a vrijednost r se izabire slučajno između nula i jedan.

Križanje za uređene kromosome (engl. *crossover for ordered chromosomes*). Koristi se kad je kromosom uređena lista, kao npr. uređena lista gradova kod problema trgovačkog putnika. Postoji više metoda križanja za uređene kromosome, a navedene su neke od njih:

Djelomično mapirano križanje (engl. *partially-mapped crossover*). Uzimaju se dva roditelja i nasumično se odabiru dvije točke križanja, a elementi između njih iz prvog roditelja preslikavaju se na ista mjesta kod djeteta. Iz drugog roditelja izdvajaju se svi elementi između točaka križanja koji nisu još preslikani. Za svaki ne preslikani element treba naći kod djeteta pripadni element koji je preslikan iz prvog roditelja na njegovo mjesto. Ako je mjesto koje element zauzima u drugom roditelju kod djeteta slobodno, element se preslikava na to mjesto, a ako je navedeno mjesto u djetetu zauzeto elementom , element se preslikava na mjesto koje element zauzima kod drugog roditelja, ako je ono slobodno. Inače postupak se ponavlja dok se ne pronađe slobodno mjesto. Elemente koji nisu

iz segmenta između točaka križanja zatim se preslikavaju iz drugog roditelja u dijete. Drugo dijete se dobiva na analogan način, uz zamijenjene uloga roditelja.

Kružno križanje (engl. *cycle crossover*). Kružno križanje kod djece nastoji sačuvati što više informacija o apsolutnom poretku elemenata kod roditelja. Elementi kromosoma svrstavaju se u cikluse . Ciklus je podskup elemenata kromosoma koji ima svojstvo da se svaki njegov element nalazi u podskupu elemenata dobivenom izdvajanjem elemenata na istim mjestima kod drugog roditelja.

Poredano križanje (engl. *order crossover*). Koristi se kod problema kod kojih je informacija o jedinki sadržana u poretku elemenata kromosoma. Ovaj je operator osmišljen tako da očuva dio poretka iz jednog roditelja i relativni poredak iz drugog roditelja.

2.6.2. Mutacija

Mutacija je genetski operator koji djeluje nad jednom jedinkom populacije (unarni operator). Kao rezultat mutacije dobivamo promijenjenu jedinku. Primjenom operatora mutacije održava se genetska raznolikost kroz generacije. Genetski algoritam bi bez korištenja operatora mutacije lako mogao zapeti u lokalnom optimumu. Ako nakon mutacije dobijemo bolju jedinku, ona će vjerojatno preživjeti i u sljedećim generacijama. Ako dobijemo lošiju jedinku nakon mutacije, ona će odumrijeti selekcijom. Uloga mutacije je i obnavljanje izgubljenog genetskog materijala. Korisnički zadan parametar p_m određuje vjerojatnost mutacije jednog bita. Ako je vjerojatnost mutacije bliska jedinici, algoritam postaje algoritam slučajne pretrage prostora rješenja. Ako je vjerojatnost mutacije bliska nuli, algoritam može već na početku zapeti u lokalnom optimumu. U nastavku su navedene mutacije koje se često koriste:

Obrtanje bitova (engl. *flip bit*). Ovaj operator mutacije može se koristiti samo za binarni prikaz. Potrebno je zadati vrijednost parametra p_m (učestalost mutacije) iz intervala $[0,1]$. Za svaki bit izvlači se vrijednost iz intervala $[0,1]$, te ako je vrijednost manja od p_m vrijednost bita se mijenja.

Granična mutacija (engl. *boundary mutation*). Zamjenjuje vrijednost izabranog gena s donjom ili gornjom graničnom vrijednošću

tog gena. Koristi kod cijelih ili realnih brojeva.

Jednolika mutacija (engl. *uniform mutation*). Zamjenjuje vrijednost izabranog gena sa slučajnom varijablom čija se vrijednost nalazi između korisnički zadanih granica za taj gen. Jednolika mutacija može se koristiti samo za realne brojeve.

Nejednolika mutacija (engl. *nonuniform mutation*). Koristi se kod realnih brojeva. Nejednolika mutacija svakom genu dodaje neku vrijednost iz distribucije oko nule. Ovakav operator mutacije sprječava stagnaciju na početku evolucije, u kasnijim stupnjevima genetskog algoritma, kad je već približno određen optimum, promjene su sve manje.

Gaussova mutacija (engl. *Gaussian mutation*). Ovaj operator mutacije izabranom genu Gaussovu razdiobu oko slučajno izabrane vrijednosti. Može se koristiti samo kod cijelih i realnih brojeva.

Mutacija zamjenom (engl. *swap mutation*). Izabiru se dva elementa na kromosomu i međusobno se zamijene.

Mutacija ubacivanjem (engl. *insert mutation*). Izabiru se dva elementa kromosoma i zatim se jedan od elemenata pomiče do drugog.

Mutacija premetanjem (engl. *scramble mutation*). Nasumično se odabire podniz elemenata i zatim se elementi tog podniza nasumično izmiješaju.

Mutacija obrtanjem (engl. *inversion mutation*). Izabiru se nasumično dva elementa i zatim se promijeni redoslijed elementa koji se nalaze između.

3. Problem raspoređivanja

Raspoređivanje je proces izrade rasporeda, tj. alokacija resursa za aktivnosti u određenom vremenskom periodu s ciljem optimiranja jedne ili više funkcija cilja. Problem raspoređivanja može uključivati jedan resurs ili više resursa. Ako problem uključuje samo jedan resurs, raspoređivanje se sastoji od određivanja redoslijeda aktivnosti na tom sredstvu. Ako se radi o problemu s više resursa, prvo treba odrediti koje će se aktivnosti odvijati na pojedinom stroju, a zatim redoslijed izvođenja. Dodavanjem ograničenja koja povezuju više aktivnosti ili pojedine aktivnosti i resurse problem raspoređivanja postaje složeniji. Resursi mogu biti npr. strojevi u radionici, piste na aerodromu itd. Aktivnosti mogu biti operacije u procesu proizvodnje, polijetanja i slijetanja na aerodromima, itd. Svaka aktivnost ima određeni prioritet, najraniji trenutak kada može započeti i željeno vrijeme završetka obavljanja. Funkcije cilja također mogu biti različite. Primjerice, minimizacija trenutka završetka posljednje aktivnosti, minimizacija aktivnosti koje se obavljaju nakon željenog vremena završetka, itd. Za formalan opis problema, potrebno je definirati svojstva poslova koji se raspoređuju, svojstva resursa koji se koriste i mjerila vrednovanja rasporeda. Objekt čije izvođenje želimo omogućiti obično se naziva posao ili zadatak. Ovisno o okruženju raspoređivanja, posao se može sastojati od više aktivnosti ili od jedne nedjeljive aktivnosti. Uobičajeno je skup svih poslova označavati sa J, a pojedini posao sa J_j. Skup zadataka se označava sa T, a pojedini zadatak sa T_j.

3.1. Svojstva poslova

U nastavku su navedena svojstva poslova:

Trajanje izvođenja (p_{ij}). p_{ij} označava trajanje izvođenja posla j na stroju i. Ako trajanje izvođenja posla j ne ovisi o stroju ili ako se radi o okruženju sa samo jednim strojem, oznaka stroja i se može izostaviti.

Vrijeme pripravnosti (r_j). Vrijeme pripravnosti r_j posla j je trenutak

kada posao dolazi u sustav, tj. najraniji trenutak kada posao j može započeti s izvođenjem.

Vrijeme željenog završetka (d_j). Vrijeme željenog završetka posla d_j je trenutak do kojeg se očekuje da posao bude završen. Dozvoljeno je da posao završi kasnije, ali se kažnjava pri ocjenjivanju rasporeda.

Vrijeme nužnog završetka ($\overline{d_j}$). Vrijeme nužnog završetka $\overline{d_j}$ je trenutak do kojeg posao mora završiti.

Težina (w_j). Težina w_j posla j je faktor prioriteta koji kaže koliki je prioritet posla j u odnosu na druge poslove u sustavu.

3.2. Okolina strojeva

Problem raspoređivanja opisan je trojkom $\alpha|\beta|\gamma$. Polje α opisuje okolinu strojeva. Polje β određuje svojstva zadataka u problemu raspoređivanja. Polje γ opisuje kriterij vrednovanja rasporeda. Moguće okoline strojeva specificirane u polju α su:

Jedan stroj (1). Okolina strojeva gdje postoji samoj jedan stroj u sustavu je najjednostavnija okolina strojeva i specijalni slučaj svih ostalih složenijih okolina strojeva.

Paralelni identični strojevi (P_m). Postoji m identičnih paralelnih strojeva. Ako m nije naveden, broj strojeva je proizvoljan. Posao j se sastoji od samo jedne operacije i može se izvoditi na bilo kojem od m strojeva. Brzina obrade pojedinog zadatka jednaka je na svim strojevima.

Jednoliki strojevi (Q_m). Postoji m paralelnih strojeva različitih brzina, ali neovisnih o vrsti zadatka koji se na njemu izvršavaju. Brzina stroja i se označava sa s_i. Ako svaki posao ima definirano trajanje izvođenja p_j, tada će se posao j na stroju i obaviti u trajanju od $p_{ij} = p_j s_i$.

Nesrodni strojevi (R_m). Ovakvo okruženje predstavlja generalizaciju prethodnog. Postoji m paralelnih strojeva i svaki ima različitu brzinu u ovisnosti o zadatku koji se na njemu obavlja.

Proizvoljna obrada (J_m). Okruženje proizvoljne obrade obuhvaća skup

poslova J od kojih se svaki posao J_j sastoji više nedjeljivih operacija. Broj operacija pojedinog posla označava se sa n_j, a pojedine operacije sa T_{1j}, T_{2j}, ..., $T_{n_j j}$. Redoslijed operacija i strojevi na kojima se izvršavaju su određeni. Moguće je da se operacije nekog posla obavljaju više puta na jednom stroju ili da se ne obavljaju nijednom na nekom drugom stroju. Trajanje obrade svake operacije ovisi o operaciji i o stroju na kojem se obavlja.

Obrada tijeka (F_m). Svaki posao ima jednak broj operacija, a broj operacija je jednak broju strojeva. Svaka operacija mora se obaviti jedanput na svakom stroju redoslijedom koji je za sve poslove jednak.

Otvorena obrada (O_m). Za razliku od okruženja obrade tijeka, redoslijed obrade pojedinih operacija u okruženju otvorene obrade je proizvoljan, ali se svaka operacija mora obaviti na predviđenom stroju.

3.3. Okolina zadataka

Svojstva zadataka u problemu raspoređivanja koja se odnose na dodatne uvjete zapisana su u polju β.

Prekidivost zadataka. Prekidivost zadataka implicira da nije nužno da posao zauzima stroj sve dok se ne obavi do kraja. Dozvoljeno je prekinuti izvršavanje posla bilo kad i umjesto njega izvršavati neki drugi posao na tom stroju. Posao koji je obavljen prije prekida nije izgubljen. Kada prekinuti posao nastavi s izvršavanjem zauzimat će stroj onoliko dugo koliko je posla preostalo. Kada je dozvoljeno prekidanje u polje se zapisuje oznaka *pmtn*.

Ograničenja u redoslijedu. Ograničenja u redoslijedu mogu postojati u okruženjima s jednim strojem i u okruženjima s više strojeva. Zahtijevaju da jedan ili više poslova budu obavljeni prije nego drugi posao započne. Postoji više formi ograničenja u rasporedu:

Ako svaki posao ima najviše jednog prethodnika i najviše jednog sljedbenika zadaci su organizirani u obliku lanaca, oznaka u polju β je *chains*.

Ako svaki posao ima najviše jednog sljedbenika, oznaka je *intree*.

Ako svaki posao ima najviše jednog prethodnika, oznaka je *outtree.*

Vremena pripravnosti. Ako su svi zadaci raspoloživi od početnog trenutka problem raspoređivanja je statički. U polje β se ne upisuje nikakva oznaka. Ako nisu svi zadaci raspoloživi od početnog trenutka, tj. ako je za svaki posao definirano vrijeme pripravnosti (engl. *ready time*), problem je dinamički. U polje β se upisuje oznaka r_j.

Trajanja postavljanja. Ako se na jednom stroju izvodi više vrsta zadataka, nakon završetka jednog zadatka, ponekad je potrebno stroj prilagoditi za obradu zadatka druge vrste. Trajanje postavljanja je vrijeme potrebno da se stroj prilagodi za obradu druge vrste zadatka. Za svaki mogući par prethodnika i sljedbenika treba definirati trajanje postavljanja.

Trajanje izvođenja. Općenito, trajanje izvođenja svakog zadatka je proizvoljno zadano, a u polje β se ne upisuje oznaka. Moguće je zadati da svi zadaci imaju jednako trajanje i tada se u polje β upisuje

trajanje svih zadataka. Moguće je i trajanja izvođenja svih zadataka ograditi donjom i gornjom graničnom vrijednošću.

Dozvoljeno čekanje. U sustavima u kojima poslovi dolaze u različitim trenucima, moguće je da neki resurs bude slobodan i da su na raspolaganju poslovi čija obrada može započeti odmah, ali u budućnosti u sustav dolazi posao većeg prioriteta. Ako su poslovi neprekidivi, treba donijeti odluku da li započeti odmah s poslom manjeg prioriteta ili čekati posao većeg prioriteta. Ovisno o mogućnosti čekanja, sustav može imati dozvoljeno čekanje ili nedozvoljeno čekanje. Sustav ima nedozvoljeno čekanje ako se sredstvo ne smije namjerno držati bez obrade. Ako nije posebno naglašeno, podrazumijeva se da postupak raspoređivanja dozvoljava čekanje.

Čekanje među procesorima. Kada se u sustavu nalazi posao koji se sastoji od više operacija, pretpostavlja se da posao može čekati neko vrijeme prije nego nastavi obradu na sljedećem stroju. Dozvoljeno čekanje ovisi o kapacitetu spremnika među strojevima. Ako nije dozvoljeno čekanje, u polje se upisuje oznaka *no-wait*, a ako je spremnik ograničenog kapaciteta upisuje se oznaka *wait*. Ako kapacitet nije ograničen, u polje se ne upisuje ništa.

3.4. Vrednovanje rasporeda

Nakon izvođenja programa i prikupljanja izlaznih veličina u sustavu slijedi vrednovanje rasporeda. Postoji više kriterija po kojima se može vrednovati raspored na temelju izlaznih veličina.

3.4.1. Izlazne veličine sustava

U nastavku su navedene izlazne veličine sustava:

- **Vrijeme završetka** C_j (engl. *completion time*)- trenutak u kojemu aktivnost j završava izvođenje
- **Protjecanje** F_j (engl. *flowtime*) – količina vremena koju je neka aktivnost provela u sustavu:
$$F_j = C_j - r_j$$
- **Kašnjenje** L_j (engl. *lateness*) – razlika (pozitivna ili negativna) između vremena završetka i vremena željenog završetka:
$$L_j = C_j - d_j$$
- **Zaostajanje** T_j (engl. *tardiness*) – pozitivni iznos kašnjenja neke aktivnosti, ako je kašnjenje negativno, zaostajanje je jednako nuli:
$$T_j = max\{0, L_j\}$$
- **Preuranjenost** E_j (engl. *earliness*) – negativni iznos kašnjenja neke aktivnosti, ako je kašnjenje pozitivno, preuranjenost je jednaka nuli:
$$E_j = max\{0, -L_j\}$$
- **Zakašnjelost** U_j – označava da li je neka aktivnost prekoračila željeno vrijeme završetka:
$$U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}$$

3.4.2. Mjerila vrednovanja rasporeda

Na osnovu navedenih izlaznih veličina definiraju se kriteriji vrednovanja rasporeda. Neki od kriterija navedeni su u nastavku:

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

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

- **Najveće kašnjenje** L_{max} (engl. *maximum lateness*) – najveće kašnjenje, a definirano je kao :

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

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

$$F_w = \sum_j w_j F_j$$

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

$$T_w = \sum_j w_j F_j$$

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

$$U_w = \sum_j w_j U_j$$

- **Težinska preuranjenost i težinsko zaostajanje** ET_w (engl. *weighted earliness and weighted tardiness*) – definira se kao zbroj težinske preuranjenosti i težinskog zaostajanja, uz posebne težinske faktore za obje vrijednosti:

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

3.5. Uvjeti raspoređivanja

Mogućnost izrade rasporeda ovisi o različitim uvjetima raspoređivanja koji mogu isključiti mogućnost upotrebe nekih postupaka.

3.5.1. Podjela uvjeta raspoređivanja

Uvjeti raspoređivanja se mogu podijeliti po nekoliko osnova:

Raspoloživost parametara

Prema raspoloživosti parametara, sustavi raspoređivanja podijeljeni su u dvije skupine:

1. **Predodređeno raspoređivanje (engl. *offline scheduling*)**. Kod predodređenog raspoređivanja, sve potrebne vrijednosti su poznate prije početka raspoređivanja.
2. **Raspoređivanje na zahtjev (engl. *online scheduling*)**. Kod raspoređivanja na zahtjev, odlučuje se na temelju trenutno dostupnih podataka, a podaci o budućnosti sustava nisu dostupni. Vrijednosti sustava mogu se mijenjati tijekom rada sustava.

Pouzdanost parametara

Okoline raspoređivanja mogu se razlikovati prema pouzdanosti parametara. Vrijednosti parametara mogu biti procijenjene s različitim preciznostima. Prema takvom kriteriju, raspoređivanje možemo podijeliti u sljedeće grupe:

1. **Determinističko raspoređivanje.** Pretpostavlja se da su vrijednosti parametara određene s dovoljnom preciznošću neovisno o trenutku u kojem postaju poznate.
2. **Stohastičko raspoređivanje**. Procijenjene vrijednosti parametara sustava kod stohastičkih okruženja nisu precizne, pa se vrijednosti parametara mogu zapravo saznati nek nakon što određeni dio sustava završi s radom. Vrijednosti parametara su definirane funkcijama koje odgovaraju određenim vjerojatnosnim raspodjelama, pa na temelju toga možemo planirati izradu rasporeda.

3.5.2. Načini izrade rasporeda

Proces raspoređivanja može se odvijati neprekinuto ili u više koraka. S obzirom na vrijeme stvaranja rasporeda, pristup izradi rasporeda može biti statički i dinamički, ili kao kombinacija oba načina.

Statički

Kod statičkog pristupa izrade rasporeda, prije početka rada sustava izradi se cijeli raspored. Ovakav način izrade rasporeda je moguć jedino ako su dostupni svi potrebni parametri. Statičko izrađivanje rasporeda nije moguće u uvjetima rada u stvarnom vremenu.

Dinamički

Kod dinamičkog pristupa izrade rasporeda, raspored se gradi iterativno, paralelno s radom sustava. Postupci za dinamičku izradu rasporeda najčešće određuju samo sljedeće stanje sustava i pozivaju se svaki puta kada nastane neka promjena u sustavu. Dinamički pristup se može koristiti i kod raspoređivanja na zahtjev i kod predodređenog raspoređivanja. Dinamičko raspoređivanje se obično koristi u uvjetima rada u stvarnom vremenu ili kad imamo promjenjive parametre.

3.6. Postupci raspoređivanja

3.6.1. Složenost algoritama raspoređivanja

Izbor algoritma ovisi o trajanju izvođenja, mogućnostima reagiranja na promjene parametara i kvaliteti dobivenog rješenja. Da bi mogli koristiti određeni algoritam u zadanoj okolini moramo odrediti ocjenu vremenske složenosti. Često se koristi O notacija za ocjenu složenosti. Veliki broj problema raspoređivanja je $NP-težak$, pa je nemoguće naći algoritam koji daje rješenje u polinomnom vremenu. Takvi problemi se najčešće rješavaju aproksimacijskim algoritmima ili heurističkim algoritmima. Za heurističke algoritme ne možemo garantirati da će pronaći optimalno rješenje.

3.6.2. Heuristički postupci raspoređivanja

Heuristički postupci raspoređivanja uključuju stohastičnost, pa za jednake početne uvjete ne dobivamo isto rješenje. Mogu se podijeliti u dvije grupe: *metode pretraživanja prostora stanja* i *algoritme koji rješenje problema grade izravno*. Metode pretraživanja prostora stanja nalaze moguće rješenje, a zatim ga iterativno ocjenjuju i mijenjaju sve dok se ne zadovolji zadani uvjet završetka. Koriste se u predodređenom statičkom raspoređivanju. Neki od postupaka su: dinamičko programiranje, neuronske mreže, evolucijski algoritmi, itd. Algoritmi koji rješenje problema grade izravno reagiraju na određene događaje u sustavu i daju sljedeće stanje sustava. Uvodi se metrika pomoću koje se ocjenjuju elementi sustava i biraju se na temelju vrijednosti metrike. Kada se definira pravilo, primjenjuju se paralelno s radom sustava dok se raspored ne izradi do kraja.

4. Simulator

Namjena simulatora je simuliranje različitih okolina raspoređivanja.

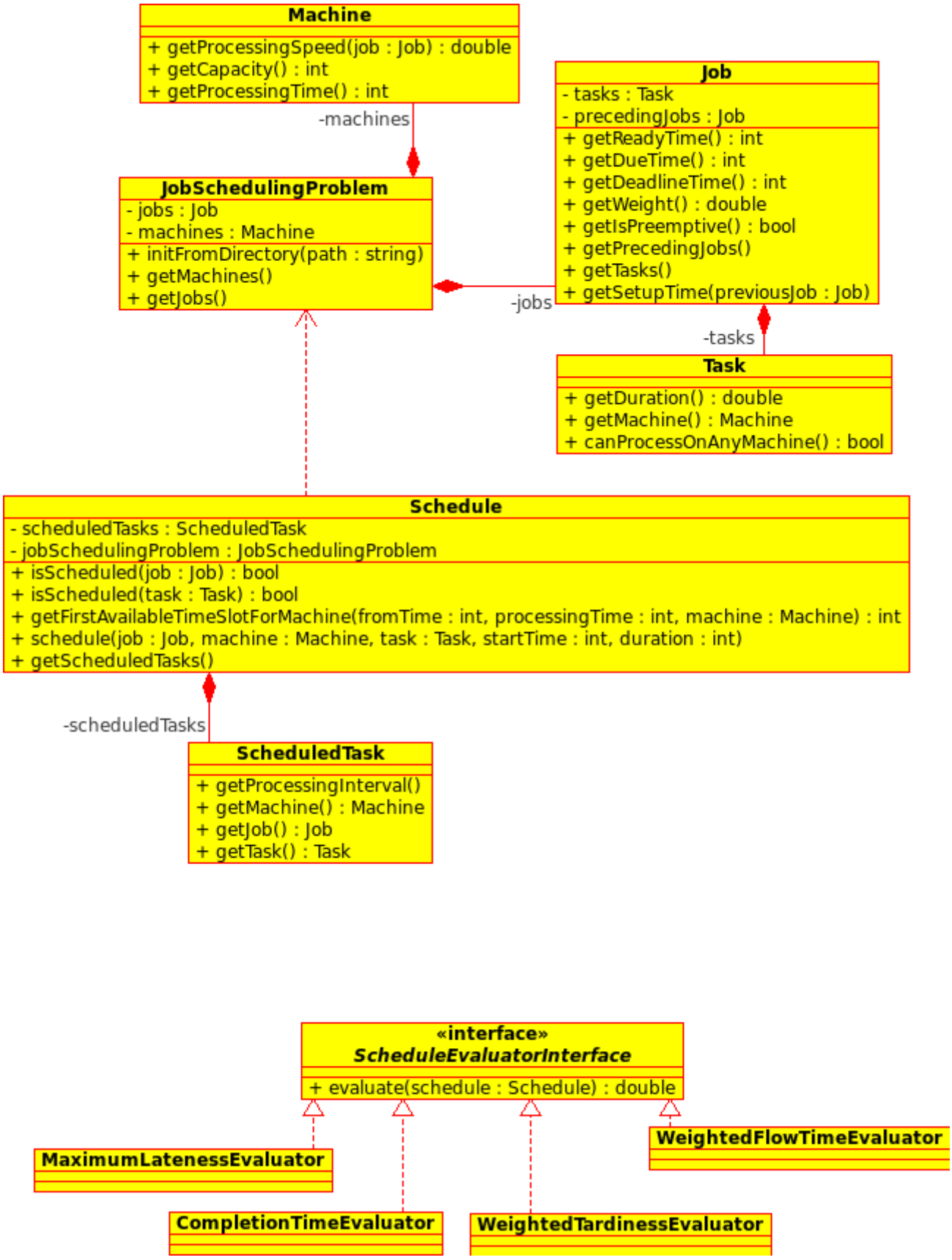
4.1. Model simulatora

Model simulatora sastoji se od skupa razreda koji čuvaju ulazne podatke, razreda u kojem je definiran problem raspoređivanja te ostvareno učitavanje ulaznih podataka, razreda koji čuva raspored za pojedini zadatak, razreda koji čuva ukupan raspored, te sučelja za ocjenjivače rasporeda. Ulazni podaci u sustav određuju okolinu strojeva i okolinu poslova. Nalaze se u razredima *Machine*, *Job* i *Task*. U ovisnosti o ulaznim podacima možemo razlikovati više okolina strojeva. U sklopu simulatora podržane su sljedeće okoline:

- **Jedan stroj** - u sustavu postoji samo jedan stroj kapaciteta nula
- **Paralelni identični strojevi** - u sustavu postoji više identičnih strojeva. Kapacitet stroja kaže koliko strojeva ima u sustavu. U okolini paralelnih identičnih strojeva svaki posao se sastoji od jedne operacije i može se izvoditi na bilo kojem stroju.
- **Jednoliki strojevi** - svaki stroj ima različitu brzinu, ali neovisnu o poslu koji je izvršava
- **Nesrodni strojevi** - za svaku moguću kombinaciju zadatka i stroja definirano je posebno trajanje izvođenja
- **Proizvoljna obrada** - obuhvaća skup poslova od kojih se svaki posao sastoji od više nedjeljivih operacija, operacije se moraju izvoditi određenim redoslijedom i na određenim strojevima, trajanje obrade svake od operacija ovisno je o operaciji i o stroju na kojemu se izvodi
- **Obrada tijeka** - svaki posao ima jednak broj operacija koji odgovara broju strojeva, a svaka operacija mora se obaviti točno jedanput na svakom stroju i to po redoslijedu koji je jednak za sve poslove

– **Otvorena obrada** - redoslijed obrade pojedinih operacija je proizvoljan, a svaka operacija se mora obaviti na predviđenom stroju

Na slici 4.1 prikazan je dijagram razreda.



Slika 4.1: Dijagram razreda

U nastavku je opisana struktura simulatora.

4.1.1. Razred Machine

U razredu *Machine* nalaze se podaci o pojedinom stroju. Podaci su sljedeći: id, vrijeme izvođenja, kapacitet stroja, brzina stroja neovisna o poslu koji se na njemu obavlja i brzina stroja definirana u ovisnosti o poslu koji se na njemu izvršava. Kapacitet stroja kaže koliko se takvih identičnih strojeva nalazi u sustavu.

4.1.2. Razred Job

U razredu *Job* nalaze se podaci koji opisuju pojedini posao. Pojedini posao može se dohvatiti preko imena ili preko id-a. Za svaki posao, zadaje se vrijeme kada dolazi u sustav, vrijeme kada očekujemo da posao bude gotov i vrijeme nužnog završetka. Postoje dvije verzije problema raspoređivanja, kada je vrijeme pripravnosti nula za sve zadatke (statički problem) i kada je vrijeme pripravnosti definirano za svaki zadatak i u općenitom slučaju je različito od nula (dinamički problem). Svaki posao ima težinski faktor koji određuje koliki je prioritet posla. Posao može biti prekidiv ili neprekidiv. Ako je posao prekidiv tada je moguće obaviti dio posla na nekom stroju, zatim zaustaviti izvođenje i započeti izvođenje nekog drugog posla, te nastaviti izvođenje prethodno započetog posla tamo gdje je stao. Mogu postojati ograničenja u redoslijedu kojim se poslovi moraju izvoditi. U ulaznim datotekama je zadana lista poslova prethodnika uz svaki posao. Posao se ne može obavljati dok se ne obave svi poslove iz te liste. Ako se na istom stroju izvode različiti poslovi, stroj se mora prilagoditi za novu vrstu posla. Vrijeme koje je potrebno da se stroj prilagodi za novi posao koji će se na njemu izvoditi zove se vrijeme postavljanja. Vrijeme postavljanja je različito za svaki par poslova. Mapa u kojoj se nalaze vremena postavljanja za svaki par poslova također se nalazi u razredu *Job*. Ako ne postoji podatak u mapi, vrijeme postavljanja ovisi samo o trenutnom poslu.

U dinamičkim problemima poslovi dolaze u različitim vremenskim trenucima i ponekad se isplati neki stroj namjerno držati bez obrade. Ovisno o mogućnosti čekanja, govorimo o dozvoljenom čekanju ili o nedozvoljenom čekanju. Ako se posao sastoji od više operacija, pretpostavlja se da djelomično završeni posao može čekati neko vrijeme na obradu na sljedećem stroju, a vrijeme čekanja je ograničeno kapacitetom spremnika koji može biti nula (nije dozvoljeno čekanje) ili može imati neku vrijednost različitu od nula (dozvoljeno je čekanje).

4.1.3. Razred Task

U nekim okolinama, strojevi se sastoje od više različitih zadataka koji se moraju izvršavati određenim redoslijedom. Podaci koji opisuju pojedini zadatak nalaze se u razredu *Task*. U razredu *Job* postoji vektor koji sadrži pokazivače na sve zadatke od kojih se sastoji taj posao. Također, svaki zadatak sadrži pokazivač na posao kojem pripada i na stroj na kojem se mora izvoditi. Ako se zadatak može obavljati na bilo kojem stroju, u ulaznu datoteku job.txt treba upisati oznaku ANY. Za svaki zadatak je definirano trajanje izvođenja. Ukupno trajanje nekog posla jednako je sumi trajanja pojedinih zadataka tog posla. Da bi se pojednostavnio problem, može se zadati da svi zadaci imaju jednako trajanje izvođenja.

4.1.4. Razred JobSchedulingProblem

U razredu *JobSchedulingProblem* opisan je problem raspoređivanja. Problem raspoređivanja čine lista poslova koje treba rasporediti po strojevima i lista strojeva. U razredu *JobSchedulingProblem* se obavlja i učitavanje ulaznih podataka direktorija u kojem se nalaze datoteke s ulaznim podacima. Ulazni podaci su podaci kojima su opisane okoline poslova i strojeva.

4.1.5. Razred Interval

Svaki zadatak je raspoređen na neki od strojeva i zauzima taj stroj u određenom vremenskom intervalu. Svaki interval je definiran početnim trenutkom i trajanjem. Te varijable nalaze se u razredu *Interval*. Pomoću metode *isInside* razreda *Interval* moguće je ustanoviti da li se neki trenutak nalazi unutar nekog intervala, a pomoću metode *isOverlaping* moguće je vidjeti da li se neka dva intervala preklapaju.

4.1.6. Razred ScheduledTask

Raspored za svaki zadatak nalazi se u razredu *ScheduledTask*. Razred sadrži pokazivač na stroj na kojem se zadatak izvodi, pokazivač na posao kojem pripada, pokazivač na zadatak, te interval u kojem se zadatak obavlja. Najvažnije metode razreda su metoda kojom se dohvaćaju posao, zatim metode kojima se dohvaća zadatak i stroj, te metoda koja dohvaća interval u kojem raspoređeni zadatak zauzima stroj.

4.1.7. Razred Schedule

Razred *Schedule* sadrži ukupan raspored svih zadataka. Osim vektora koji sadrži pokazivače na raspoređene zadatke, u razredu *Schedule* nalazi se i pokazivač na *JobSchedulingProblem*, tj. na problem raspoređivanja (vektor strojeva i vektor poslova). Metoda *schedule*koja se nalazi u razredu *Schedule* u jedan vektor zapisuje raspored za svaki posao, odnosno za svaki zadatak. Metoda kao parametre prima pokazivač na posao koji se raspoređuje, pokazivač na stroj na kojem je raspoređen zadatak, zadatak koji se izvršava na tom stroju, vrijeme kada počinje izvršavanje zadatka i trajanje izvršavanja zadatka kao ulazne parametre. Pomoću metode *getFirstAvailableTimeSlotForMachine* možemo pronaći prvi slobodan termin na nekom stroju. Zadaje se i koliko dugo će stroj biti zauzet. Kao parametre je potrebno zadati stroj koji se zauzima i trajanje izvođenja zadatka kojeg želimo rasporediti na tom stroju.

4.1.8. Sučelje ScheduleEvaluatorInterface

Simulator ima i sučelje za evaluatore rasporeda *ScheduleEvaluatorInterface*. Metoda evaluate vraća vrijednost koja opisuje raspored. Postoje razni evaluatori rasporeda kao npr. ukupna duljina rasporeda (posljednje vrijeme završetka svih poslova u sustavu), najveće kašnjenje, težinsko protjecanje (suma težinskog protjecanja svih poslova), težinsko zaostajanje (težinska suma zaostajanja svih poslova), težinska zakašnjelost i težinska preuranjenost i težinsko zaostajanje (zbroj težinske preuranjenosti i težinskog zaostajanja, uz posebne težinske faktore za obje vrijednosti). Od navedenih evaluatora implementirani su: ukupna duljina rasporeda, najveće kašnjenje, težinsko protjecanje i težinsko zaostajanje. Sustavu se može dodati bilo koji proizvoljan evaluator rasporeda.

4.2. Opis ulaznih podataka

Ulazni podaci u sustav su podaci koji opisuju okolinu strojeva i okolinu poslova. Podaci se zadaju direktorijem koji sadrži ulazne datoteke machine.txt, job.txt, constraints.txt, previous.txt, speeds.txt, setup_time.txt. Učitavanje podataka obavlja se pomoću metoda iz razreda JobSchedulingProblem.

U datoteci *machine.txt* nalaze se podaci o strojevima. U datoteci se nalaze sljedeći podaci za svaki stroj: ime stroja, brzina izvođenja na stroju neovisna o poslu koji se na njemu obavlja i kapacitet stroja. Brzina stroja i kapacitet definiraju okolinu strojeva. Kapacitet nekog stroja označava koliko identičnih strojeva te vrste ima u sustavu. Ako u sustavu postoji samo jedan stroj čiji je kapacitet jedan, govorimo o okolini jednog stroja. Ako postoji samo jedna vrsta stroja u sustavu, kapaciteta većeg od jedan, govorimo o okolini paralelnih identičnih strojeva. U sustavu može biti više različitih strojeva različitih kapaciteta. U okolini jednolikih strojeva svi strojevi imaju različite brzine, ali neovisne o poslu koji se na njima obavlja. U okolini nesrodnih strojeva svaki stroj ima različitu brzinu ovisnu o poslu koji se na njemu obavlja. Podaci o strojevima se učitavaju pomoću metode *initMachinesFromFile*. U svakoj liniji u datoteci nalaze se podaci za jedan stroj. Podaci su međusobno odvojeni prazninama.

U datoteci *job.txt* nalaze se podaci o poslovima i zadacima od kojih se posao sastoji. Poslovi se mogu sastojati od više različitih zadataka. U svakoj liniji se nalaze podaci koji opisuju jedan posao i zadaci od kojih se sastoji taj posao. Svaki posao ima težinski faktor koji određuje prioritet tog posla u sustavu. Uz ime posla i težinski faktor zadani su i podaci vezani za pojedine zadatke posla. Za svaki zadatak je zadano ime stroja na kojem se mora izvršavati ili oznaka ANY ako se zadatak može izvršavati na bilo kojem stroju i trajanje pojedinog zadatka, a međusobno su odijeljeni dvotočkom. Podaci za pojedini zadatak međusobno su odijeljeni zarezom. Učitavanje podataka se obavlja pomoću metode *initJobsFromFile*.

U datoteci *constraints.txt* nalaze se ograničenja vezana za poslove. Učitavanje podataka iz datoteke constraints.txt se obavlja pomoću metode *initConstraintsFromFile*. Ograničenja su sljedeća: vrijeme pripravnosti, vrijeme željenog završetka, vrijeme nužnog završetka i prekidivost (0 ako posao nije prekidiv, 1 ako je prekidiv). U svakoj liniji u datoteci nalazi se ime posla i ograničenja vezana za taj posao redoslijedom kojim su prethodno navedena.

U datoteci *previous.txt* nalaze se ograničenja u redoslijedu izvođenja. Za svaki posao koji se ne može obaviti prije nego što se obave neki drugi poslovi, postoji lista poslova prethodnika. U svakoj liniji zapisano je ime posla i lista poslova prethodnika koji su međusobno odvojeni prazninom, a elementi liste su odvojeni zarezom.

Podaci se učitavaju pomoću metode *initPrecedingJobsFromFile*.

U datoteci *speeds.txt* nalaze se podaci o brzini izvođenja posla na nekom stroju za svaki par poslova i strojeva. Učitavanje se obavlja pomoću metode *initSpeedsFromFile*. Ako ne postoji podatak u datoteci speeds.txt, računa se s brzinom stroja koja je neovisna o poslu koji se na njemu obavlja (okolina jednolikih strojeva).

U datoteci *setup_time.txt* nalaze se vremena postavljanja stroja za svaki par poslova. Ako se na nekom stroju obavljaju različite vrste poslova, potrebno je nakon sto jedan posao završi prilagoditi stroj sljedećem poslu koji se na njemu treba obaviti. Vrijeme potrebno da se stroj prilagodi za novi posao naziva se vrijeme postavljanja. U svakoj liniji nalaze se imena poslova i vrijeme postavljanja, a podaci su odvojeni prazninama. Podaci se učitavaju pomoću metode *initSetupTimesFromFile*. Dan je primjer definiranja problema za okolinu jednolikih strojeva. Podaci u ulaznim datotekama su sljedeći:

machine.txt

M0 1.96 1

M1 3.70 2

M2 2.32 1

job.txt

J0 0.8 ANY:33

J1 0.45 ANY:23

J2 0.56 ANY:57

J3 0.41 ANY:31

constraints.txt

J0 15 40 10000 0

J1 3 28 10000 0

J2 16 62 10000 0

J3 25 52 10000 0

previous.txt

J0 J1,J2

speeds.txt

J0 M0 1.0

J0 M1 1.0

J0 M2 1.0

J1 M0 1.0

J1 M1 1.0

J1 M2 1.0

J2 M0 1.0

J2 M1 1.0

J2 M2 1.0

J3 M0 1.0

J3 M1 1.0

J3 M2 1.0

setup_time.txt

J0 J1 1

J0 J2 1

4.3. Vrednovanje rasporeda

Simulator ima sučelje za evaluatore rasporeda *ScheduleEvaluatorInterface* prikazanog ispod:

```
class ScheduleEvaluatorInterface{

        public:

        virtual double evaluate( Schedule* schedule)=0;

};
```

Metoda *evaluate* vraća vrijednost koja opisuje raspored, a veći broj znači bolji raspored. Implementirani su sljedeći evaluatori:

- **CompletionTimeEvaluator** - vrednovanje rasporeda na temelju ukupne duljine rasporeda
- **MaximumLatenessEvaluator** - vrednovanje rasporeda na temelju najvećeg kašnjenja

- **WeightedFlowTimeEvaluator** - vrednovanje rasporeda na temelju sume težinskog protjecanja svih poslova
- **WeightedTardinessEvaluator** - vrednovanje rasporeda na temelju težinske sume zaostajanja svih poslova

Sustavu je moguće dodavati nove evaluatore rasporeda.

5. Korištenje simulatora

Korištenjem programskog sustava za simulaciju različitih okolina raspoređivanja omogućena je brža i lakša izrada raspoređivača. Simulator omogućava da se usmjerimo samo na algoritam izrade rasporeda. Potrebno je zadati okolinu strojeva i okolinu poslova, te komponentu raspoređivača. Okolina strojeva i okolina poslova definirani su podacima u ulaznim datotekama. Osim simulatora, implementirano je i nekoliko različitih ocjenjivača rasporeda koji su preko sučelja povezani sa simulatorom. Vrijednost koju vraća ocjenjivač rasporeda predstavlja ocjenu rasporeda. Uz postojeće ocjenjivače, sustavu se mogu dodati i nove komponente ocjenjivača.

5.1. Raspoređivanje pomoću genetskog algoritma

Budući da problem raspoređivanja pripada razredu *NP* problema, za rješavanje tog problema prikladno je koristiti genetski algoritam ili neku drugu heurističku metodu, jer je metoda iscrpnog pretraživanja neprikladna za ovakve probleme. Ovakav način raspoređivanja ne garantira da pronalaženje optimalnog rješenja, ali daje dovoljno dobro rješenje. Za usporedbu s rezultatima raspoređivanja pomoću genetskog algoritma implementirana je heuristička metoda raspoređivanja temeljena na pravilima. Genetski algoritma koji je implementiran rješava problem raspoređivanja za okoline nesrodnih strojeva. Algoritam je opisan u razredu *JobSchedulingSolverGA*. Potrebno je zadati sljedeće parametre genetskog algoritma: veličinu populacije, broj iteracija i vjerojatnost mutacije. Zadani parametri čuvaju se u razredu *JobSchedulingSolverParamsGA*. Razred sadrži i pokazivač na problem raspoređivanja te na sučelje evaluatora.

```
genetski_algoritam
   {

            kreiraj pocetnu populaciju;
            sve dok nije zadovoljen uvjet zaustavljanja
            {

                    selektiraj roditelje;
                    krizaj roditelje;
                    mutiraj dobivene potomke;

            }
   }
```

Slika 5.1: Pseudokod genetskog algoritma

Pseudokod algoritma prikazan je na slici 5.1. Algoritam traži najbolji raspored na način opisan u nastavku.

Prvo treba stvoriti početnu populaciju. Veličina populacije se zadaje preko parametara i konstantna je kroz generacije. Stvara se onoliko rasporeda kolika je veličina populacije. Jedinke početne populacije stvaraju se jednostavnim algoritmom, tako da se svaki posao dodijeli slučajno izabranom stroju, izračuna se vrijeme izvršavanja posla na tom stroju, pronalazi se prvi slobodan interval za taj stroj i na kraju se rasporedi posao.

Nakon što je stvorena početna populacija nad populacijom se primjenjuju genetski operatori križanja i mutacije u svakoj iteraciji. Broj iteracija zadaje se preko parametra. U svakoj iteraciji križanje se obavlja više puta.

Operator *križanja* ostvaren je na sljedeći način. Izabiru se tri jedinke slučajnom odabirom i križaju se najbolje dvije, a treća se zamijeni jedinkom nastalom križanjem. Postupak se ponavlja onoliko puta kolika je veličina populacije. Samo križanje se obavlja na slijedeći način. Dohvaćaju se svi poslovi i spremaju se u vektor. Za svaki posao računaju se trajanja izvođenja na strojevima za obje jedinke koje se križaju. Ako izvođenje nekog posla traje kraće za neku jedinku, postoji veća šansa da će taj posao u novom rasporedu biti raspoređen na isti stroj kao u boljoj jedinki. Na taj način se postiže da se križanjem prenesu bolja svojstva na novu jedinku. Neke jedinke mogu biti izabrane za križanje više puta, a neke nijednom. Kad su svi poslovi dodijeljeni nekom stroju, treba još izračunati trajanje

izvođenja za svaki posao, te naći vremenske intervale u kojima se mogu izvršavati na izabranim strojevima. Na kraju treba još napraviti ukupan raspored.

Nakon operatora križanja slijedi mutacija. Vjerojatnost mutacije zadaje se preko parametara. S obzirom na vjerojatnost mutacije i veličinu populacije računa se broj rasporeda nad kojima će se obaviti operator mutacije. Jedinke koje imaju lošija svojstva će imati veće šanse da budu mutirane. Sve jedinke u populaciji se ocjenjuju, a zatim se izabiru jedinke koje će biti mutirane.

Operator *mutacije* opisan je u nastavku. Mutiranje se obavlja tako da se za n poslova promijeni stroj na kojem se izvršavaju. Novi stroj se izabire slučajnim odabirom. Broj poslova n koji će biti dodijeljeni nekim drugim strojevima je proizvoljan. Ovakva vrsta mutacije je jednolika mutacija. Za svaki posao se mora ponovo izračunati interval u kojem zauzima pojedini stroj. Zatim se mora naći prvi slobodan termin za stroj kojemu je dodijeljen i na kraju se mora rasporediti.

Nakon što smo primijenili operatore križanja i mutacije u svakoj iteraciji i došli do zadnje generacije, treba naći najbolju jedinku u populaciji koja predstavlja najbolji raspored. Ne postoji garancija da je pronađen optimalan raspored, ali predstavlja dovoljno dobro rješenje. Korištenjem simulatora znatno je olakšana izrada rasporeda.

5.2. Raspoređivanje zasnovano na pravilima

Budući da problem raspoređivanja pripada razredu *NP* problema, za rješavanje tog problema prikladno je koristiti heurističke metode jer je nepraktično tražiti rješenje iscrpnim pretraživanjem. Na taj način ne možemo biti sigurni da ćemo naći optimalno rješenje, ali možemo naći dovoljno dobro rješenje. Osim raspoređivača temeljenog na genetskom algoritmu, implementiran je i raspoređivač temeljen na pravilima. Metoda raspoređivanja temeljena na pravilima je heuristička metoda i daje dobre rezultate pa može služiti za usporedbu s rezultatima raspoređivanja pomoću genetskog algoritma.

U ovom radu, implementirano je pravilo gdje je prioritet zadatka obrnuto proporcionalan trajanju zadatka. Pravilo je opisano u razredu *SchedulingPolicySPT*. Povezano je sa raspoređivačem preko sučelja *SchedulingPolicy* prikazanog dolje:

```
class SchedulingPolicy {
        public:
        virtual ~SchedulingPolicy(){}
        virtual int priority(
                        Job* job,
                        Task* task,
                        Machine* machine,
                        Schedule* schedule) = 0;
};
```

Sustavu se mogu dodavati nova pravila raspoređivanja. Algoritam raspoređivanja se nalazi u razredu *PolicyScheduler*. Da se stvori raspored potrebni su lista strojeva i lista poslova te pravilo raspoređivanja. Na temelju tih podataka algoritam gradi ukupan raspored svih poslova na strojevima u određenim vremenskim intervalima. Poslovi se raspoređuju koristeći metodu *schedule* koja je dio simulatora. Algoritam gradi raspored na sljedeći način. Dohvaćaju se svi strojevi, a zatim se sortiraju prema dostupnosti. Zatim se u petlji za svaki stroj traže svi zadaci koji se na njemu mogu izvoditi. Prolazi se kroz listu strojeva sortiranih prema dostupnosti tako da se prvo traže zadaci koji se mogu izvoditi na stroju koji je prvi dostupan. Ako nema zadataka koji se mogu izvoditi na stroju, provjerava se da li ima zadataka koji se mogu izvoditi na sljedećem stroju.

Ako ima zadataka koji se mogu izvoditi na trenutnom stroju, računa se prioritet svih poslova i dohvaća se onaj koji ima najveći prioritet. Zatim se računa trajanje izvođenja posla na tom stroju i traži se prvi slobodni trenutak u kojem se može dodijeliti posao. Slobodni interval se nalazi pomoću metode *getFirstAvailableTimeSlotForMachine* koja je dio simulatora. Nakon toga se zadatak raspoređuje. Metode *shedule* i *getFirstAvailableTimeSlotForMachine* koje su dio simulatora znatno olakšavaju proces raspoređivanja. Podskup zadataka kojima se ocjenjuje prioritet pronalazi se u tri koraka:

1. Nalazi se skup zadataka za koje se provjeravaju sva ograničenja osim raspoloživosti i sprema se u vektor.

2. Nalazi se trenutak u kojem najraniji spremni zadatak završava, uzimajući u obzir trajanje zadatka i vrijeme pripravnosti.

3. Gradi se podskup zadataka kojima se ocjenjuje prioritet tako da se iz liste dobivene u 1. koraku uzimaju zadaci kojima je vrijeme pripravnosti ranije od trenutka u kojemu najraniji spremni zadatak završava. S tom provjerom

je ispunjen uvjet da podskup zadataka kojima ocjenjujemo prioritet sadrži samo one zadatke koji mogu započeti prije nego što može završiti najkraći raspoloživi (spremni) zadatak na tom stroju.

Poslovi se raspoređuju sve dok ima neraspoređenih poslova.

Pravila raspoređivanja koja se često koriste su:

- SPT (engl. *shortest processing time*), prioritet obrnuto proporcionalan trajanju zadatka:
$$\pi_j = \frac{1}{p_j}$$
- WSPT (engl. *weighted shortest processing time*), funkcija prioriteta je definirana kao:
$$\pi_j = \frac{w_j}{p_j}$$
- EDD (engl. *earliest due date*), definirano sa:
$$\pi_j = \frac{1}{d_j}$$
- Montagne pravilo, definirano kao:
$$\pi_j = \frac{w_j}{p_j} * \left(1 - \frac{d_j}{\sum_{i=1}^{n} p_i}\right)$$

gdje je n broj poslova koje treba rasporediti.

- Rachamadugu & Morton pravilo, definirano sa:
$$\pi_j = \left(\frac{w_j}{p_j}\right) * \left[exp\left(\frac{-(d_j - p_j - time)^+}{k * p_{AV}}\right)\right]$$

gdje je p_{AV} prosječno trajanje svih neraspoređenih poslova (tj. svih onih za koje se računaju prioriteti), time je trenutno vrijeme, a operator '$+$' definiran je kao:

$$x^+ = max\{x, 0\}$$

6. Rezultati

Kvaliteta rješenja dobivenih genetskim algoritmom kao i utjecaj parametara testirani su na generiranom skupu problema različitih veličina. Veličina problema ovisi o broju poslova i strojeva na kojima se ti poslovi izvode. Zadani broj poslova u skupu primjera je 12, 25, 50 i 100, a strojeva 3, 6 i 10. Svi primjeri predstavljaju okolinu nesrodnih strojeva. Za svaku kombinaciju broja poslova i strojeva u skupu za testiranje nalazi se nekoliko različitih primjera problema. Ispitan je utjecaj parametara genetskog algoritma na kvalitetu rješenja.

Kvaliteta rasporeda je negativna vrijednost vremena završetka posljednjeg posla u sustavu. U svim se eksperimentima raspored vrednuje po kriteriju završetka posljednjeg posla u sustavu. Ispitivanje je vršeno na skupu rasporeda. Normirana kvaliteta rasporeda i dobivena je pomoću sljedeće formule:

$$n_i = \frac{d_i - d_{min}}{d_{max} - d_{min}}, \tag{6.1}$$

gdje je n_i normirana kvaliteta rasporeda i, d_i je kvaliteta rasporeda i, a d_{max} i d_{min} su najveća i najmanja prosječna vrijednost svih ponavljanja u skupu rasporeda dobivenih ispitivanjem.

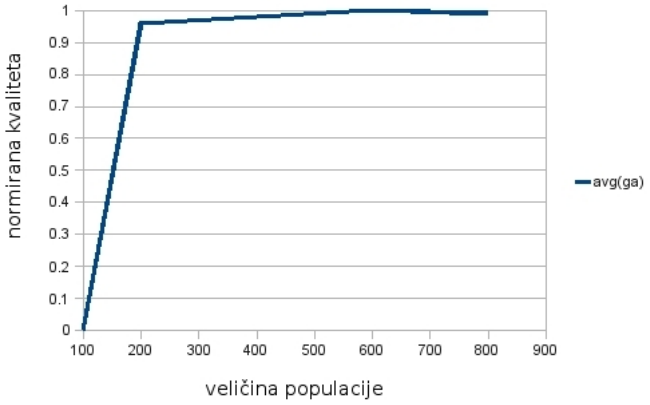
Kriterij zaustavljanja genetskog algoritma je najveći broj iteracija ili 400 iteracija bez poboljšanja rezultata. Navedeni kriterij korišten je u svim eksperimentima.

Na slici 6.1 prikazan je utjecaj veličine populacije na rezultate dobivene genetskim algoritmom. Eksperiment je napravljen na skupu primjera sa 50 poslova i 6 strojeva. Na x osi nalaze se vrijednosti veličine populacije, na y osi nalazi se normirana kvaliteta dobivena kao što je opisano u formuli 6.1. Parametri korišteni u ovom eksperimentu navedeni su u tablici 6.1. Na slici 6.1 možemo vidjeti kako kvaliteta rješenja prvo izrazito raste s povećanjem populacije, a nakon toga se utjecaj povećanja populacije bitno smanjuje. Možemo pretpostaviti da je utjecaj povećanja

Broj iteracija	300
Vjerojatnost mutacije	0.3
Veličina populacije	100-800
Uvjet zaustavljanja	300 iteracija
Broj ponavljanja	5

Tablica 6.1: Parametri genetskog algoritma

populacije nakon neke točke zanemariv.

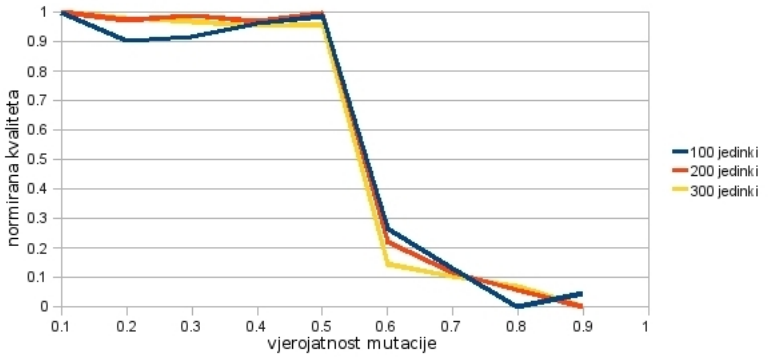


Slika 6.1: Utjecaj veličine populacije

Na slici 6.2 prikazan je utjecaj mutacije na rezultate dobivene genetskim algoritmom. Eksperimet je napravljen na primjeru sa 100 poslova i 10 strojeva. Na x osi se nalaze vjerojatnosti mutacije, a na y osi je normalizirana kvaliteta rasporeda kao što je opisano formulom 6.1. Rezultati su dobiveni kao prosječna vrijednost 10 ponavljanja. Na slici 6.2 se može vidjeti da poslije nekog praga povećavanje mutacije smanjuje kvalitetu dobivenih rasporeda. Test je napravljen za okolinu nesrodnih strojeva. Parametri genetskog algoritma prikazani su u tablici 6.2.

Broj iteracija	1000
Vjerojatnost mutacije	promjenjiva
Veličina populacije	100-300
Uvjet zaustavljanja	1000 iteracija ili 400 iteracija bez poboljšanja
Broj ponavljanja	10

Tablica 6.2: Parametri genetskog algoritma



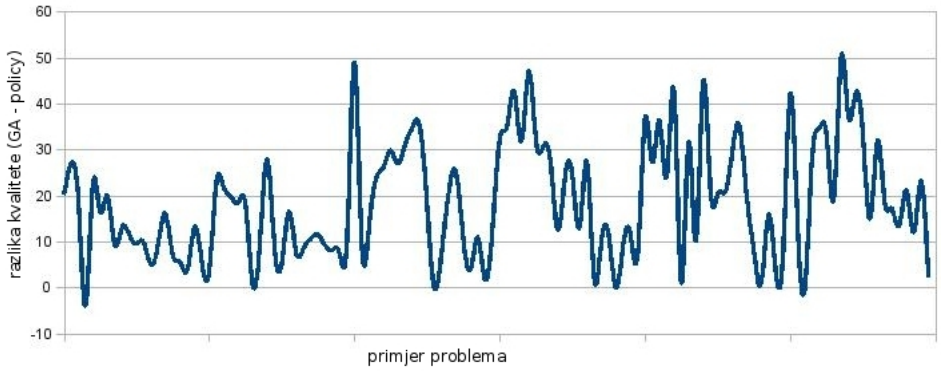
Slika 6.2: Utjecaj mutacije

Kako bi odredili kvalitetu rješenja dobivenih genetskim algoritmom, rezultati su uspoređeni sa metodom raspoređivanja temeljenom na SPT pravilu. Ta metoda generira rješenje sljedećim postupkom. U svakoj iteraciji raspoređuje se jedan posao u sustavu, a postupak završava kada su raspoređeni svi poslovi. Raspored za jedan posao generira se tako da se nađe prvi slobodan stroj, a zatim se na taj stroj rasporedi posao najvećeg prioriteta. Prioritet posla ovisi o vremenu završetka tog posla. Izlazna veličina koja se optimira u oba sustava je vrijeme završetka posljednjeg posla.

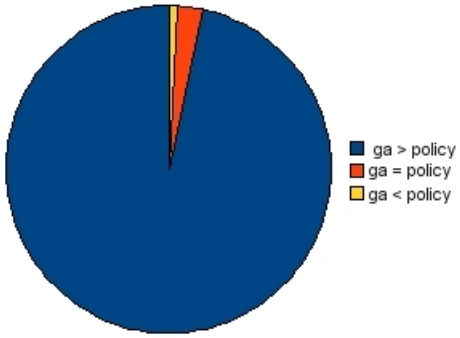
Na slici 6.3 prikazana je razlika između rezultata dobivenih genetskim algoritmom i rezultata dobivenih raspoređivanjem temeljenim na pravilima. Na x osi se nalaze ispitni primjeri, a na y osi razlika između dva algoritma. Vrijednost na y osi prikazuje koliko je rješenje dobiveno genetskim algoritmom bolje od rješenja dobivenog raspoređivanjem temeljenom na pravilima. Vrijednost 0 označava da su rješenja jednake kvalitete, vrijednost veća od 0 označava da je genetski algoritam bolji, dok vrijednost manja od 0 označava da je bolji rezultat dobiven raspoređivanjem temeljenim na pravilima. Rezultati dobiveni genetskim algoritmom bolji su u skoro svim primjerima problema. Test je napravljen za okolinu nesrodnih strojeva s veličinom populacije od 1000 jedinki i 300 iteracija. Vrijednosti parametara mogu se vidjeti u tablici 6.3. Na slici 6.4 može se vidjeti na koliko je primjera genetski algoritam bio bolji, lošiji ili jednak od raspoređivanja temeljenog na pravilima.

Broj iteracija	300
Vjerojatnost mutacije	0.3
Veličina populacije	1000
Uvjet zaustavljanja	300 iteracija
Broj ponavljanja	5

Tablica 6.3: Parametri genetskog algoritma



Slika 6.3: Usporedba rezultata dobivenih genetskim algoritmom s rezultatima dobivenim raspoređivanjem temeljenim na SPT pravilu



Slika 6.4: Kvantitativna usporedba rezultata dobivenih genetskim algoritmom s rezultatima dobivenim raspoređivanjem temeljenim na SPT pravilu

7. Zaključak

U praktičnom dijelu ovog rada ostvaren je sustav za simulaciju različitih okolina raspoređivanja. Također, ostvareni su sustavi raspoređivanja temeljeni na genetskom algoritmu i na pravilima. Ispitan je utjecaj parametara na rezultate genetskog algoritma, te su rezultati dobiveni genetskim algoritmom uspoređeni s rezultatima dobivenim raspoređivanjem temeljenim na pravilima.

Korištenjem programskog sustava za simulaciju različitih okolina raspoređivanja olakšana je implementacija oba raspoređivača. Korištenje simulatora je omogućilo veću usmjerenost na rad samog algoritma. Simulator u sebi sadrži komponente zajedničke svim raspoređivačima, te biblioteku komponenata za vrednovanje rasporeda, omogućuje jednostavno proširivanje te biblioteke novim kriterijima vrednovanja. Također, upotrebom simulatora olakšano je učitavanje ulaznih podataka.

Ispitan je utjecaj parametara mutacije i veličine populacije na rad genetskog algoritma. Povećavanjem parametra mutacije kvaliteta rasporeda dobivenih genetskim algoritmom prvo se povećava, a zatim nakon neke vrijednosti počinje naglo padati. Povećanjem veličine populacije poboljšava se kvaliteta dobivenih rasporeda. Za manje vrijednosti veličine populacije je povećanje kvalitete izrazito, no prirast kvalitete opada kako vrijednosti veličine populacije postaju sve veće dok u nekoj vrijednosti daljnje povećavanje ne uzrokuje kvalitetniji raspored.

Kvaliteta rezultata dobivenih genetskim algoritmom uspoređena je s kvalitetom rezultata dobivenih raspoređivanjem temeljenim na pravilima. Uz pravilan izbor parametara genetski algoritam pokazao se boljim na primjerima različitih veličina. Genetski algoritam davao je slabije rezultate uz loše podešene parametre. Izbor parametara je osobito važan za dobre rezultate genetskog algoritma.

LITERATURA

[1] Michael L. Pinedo, *Scheduling Theory, Algorithms, and Systems*, 2008 Springer

[2] Domagoj Jakobović, *Raspoređivanje zasnovano na prilagodljivim pravilima*, 2005.

[3] Melanie Mitchell, *An Introducition to Genetic Algorithms*, 1996.

[4] Marin Golub, *Genetski Algoritam*, Prvi dio, 1997.