SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1801

Raspoređivanje u proizvoljnoj obradi uz pomoć genetskog algoritma

Mirko Vladović

Zagreb, srpanj, 2009.

Sadržaj

1. Uvod

Ovaj diplomski rad se bazira na pokušaju rješavanja problema proizvoljne obrade heurističkom metodom, konkretno genetskim algoritmom. Problem proizvoljne obrade (eng. *Job Shop Scheduling Problem*) je jako težak praktični problem. Može ga se pronaći u industriji i upravljanjima resursima. Cilj optimizacije je izraditi raspored koji bi minimizirao neki zadani kriterij. Problem je NP-težak, što znači da ne postoji odgovarajući algoritam polinomske složenosti koji pronalazi optimalno rješenje. Egzaktne metode su primjenjive samo za male primjere problema, pa je pozornost istraživanja prebačena na približne metode. Jedna od tih metoda su i genetski algoritmi. U ovom radu je dana teoretska i praktična analiza tog algoritma. Na kraju je obavljeno eksperimentiranje s parametrima te je dana usporedba kvalitete rješenja s već postojećim pokušajima rješavanja istog problema koji se mogu pronaći u literaturi.

Radnja se sastoji od sedam poglavlja. Prvo je poglavlje uvodno i daje kratki sažeti uvid u problematiku.

Drugo poglavlje se bavi detaljnijim upoznavanjem s problemom proizvoljne obrade. Dana je matematička definicija problema, pregledi različitih varijanti problema te sličnih problema kao i različitih kriterija optimizacije te vrsta rješenja.

Treće poglavlje sadrži osvrt na postojeće metode za rješavanje problema koje postoje u literaturi. Dani su kratki opisi kako pojedini načini rade te ocjena njihove konkurentnosti u usporedbi s drugim algoritmima.

Četvrti dio ima za temu genetski algoritam. Opisan je kratko prirodni proces evolucije koji je bio ideja za nastanak ove metode. Dan je opis rada, te su detaljnije obrađeni prikazi rješenja i genetski operatori koji se koriste u programskoj implementaciji koja prati ovaj diplomski rad.

Peti dio daje analizu rezultata te osjetljivosti kvalitete rješenja i računalne složenosti o izboru parametara. Rezultati su uspoređeni s rezultatima drugih radova koji su se mogli pronaći u literaturi.

Šesti dio daje zaključak cijelog rada.

Sedmo poglavlje daje popis korištene literature.

2. Problem proizvoljne obrade

2.1 Podrijetlo problema

Problem proizvoljne obrade (eng. *Job Shop Problem*) se može najčešće naći u malim specijaliziranim proizvodnim zadatcima koji se bave specijaliziranim proizvodnim procesima kao što su male korisničke narudžbe [6]. Obrada dotad aktivnog posla se prebacuje na drugi posao, nakon što je prethodni završio. Po prirodi ovog tipa proizvodnje, takve radnje su specijalizirane u umijeću i postupku obrade raznih zadataka. Primjer bi bio strojarska radnja koja proizvodi dijelove za avionsku industriju. Naime, većina komponenata na zrakoplovima se izrađuje u malim količinama, u odnosu na iPodove. Drugi primjeri gdje se isti problem pronalazi su brušenje, izoštravanje alata, brušenja matrica te proizvodnja alata. Suprotnost bi bila proizvodnja sa stalnim protokom kao što je tekstil, čelik ili hrana.

Pokretne trake (eng. *transfer lines*) su već dugo učinkovit način za proizvodnju tipa velike količine i male raznolikosti (eng. *high-volume/low-variety*). Kod proizvodnja drugačijeg kategorije, one male količine i visoke raznovrsnosti (eng. *low-volume/high-variety*) te one srednje količine i srednje različitosti (eng.*mid-volume/mid-variety*) nalazimo mnoštvo poteškoća. Ove dvije zadnje navedene vrste proizvodnje zahvaćaju čak 50%-75% ukupne proizvodnje, sa trendom rasta. Problemi su da su vremena izrade preduga, a iskorištenje strojeva malo. Mnoge od ovih poteškoća u produkciji mogu se pripisati pogreškama u planiranju i raspoređivanju resursa. Problem proizvoljne obrade se bavi upravo proizvodnjom male količine i velike raznovrsnosti kod kojih nalazimo ove probleme [10]. Uspješno rješavanje navedenog problema ima velik utjecaj na profitabilnost i kvalitetu proizvoda.

2.2 Definicija problema

Problem spada u grupu diskretnih ili kombinatorijalnih optimizacija.

- Matematička definicija uključuje skup od *m* strojeva (eng. *machines*) na kojima se obavlja skup od *n* poslova (eng.*jobs*).

- Svaki se posao sastoji od zadataka (eng. *tasks*) koji se moraju obaviti na različitim strojevima. Svaki zadatak unutar jednog posla ima svoje vrijeme trajanja i obavlja se na drugom stroju.
- Pri tome se mora poštovati **pravilo prednosti** (eng.*precedence constraint*) po kojem je definirano kojim redom posao obilazi strojeve, odnosno raspored između zadataka istog posla.
- Dok jedan posao zauzima neki stroj, drugi posao ne može dobiti taj stroj. Kada započne obrada jednog posla, ne može se prekinuti s trenutnim poslom i privremeno dodijeliti stroj nekom drugome poslu. Zadatci unutar svakog posla su nedjeljive operacije.
- Različiti zadatci istog posla se ne mogu istovremeno obavljati, makar se obavljaju na različitim strojevima. Tek nakon što završe svi prethodni zadatci istog posla na red dolazi promatrani zadatak.

Cilj je pronaći takav raspored poslova koji optimira unaprijed zadano mjerilo izvedbe.

Problem proizvoljne obrade je generalizacija poznatog problema trgovačkog putnika. Taj se problem zadaje kao skup gradova koji se moraju obići točno jednom. Ti su gradovi zamišljaju kao čvorovi u grafu. Svaki brid u gradu (ili udaljenost između dvaju gradova) ima pridruženu težinu. Zadatak je optimizacije pronaći stazu u grafu takvu da ukupni zbroj težina bridova koje se nalaze u stazi (ukupna duljina puta) bude najmanji. Problem proizvoljne obrade se svodi upravo na to ukoliko se radi samo o jednom stroju na kojem se obavljaju poslovi.

Problem proizvoljne obrade nije samo NP-težak, već je jedan od najgorih pripadnika tog razreda. To znači da se vrijeme potrebno da bi se pronašlo optimalno rješenje povećava eksponencijalno sa veličinom problema. O težini problema svjedoči činjenica da je jedan 10x10 (deset poslova na deset strojeva) problem ostao neriješen 25 godina [2].

Gotovo svi praktični problemi raspoređivanja mogu se izraziti kao problem proizvoljne obrade, obično kao ograničene i ublažene verzije ovog klasičnog problema [7].

2.3 Varijacije problema i slični problemi

Problem raspoređivanja proizvoljne obrade ima više inačica. Ako su na početku poznati svi poslovi, zadatci i njihova trajanja te se zna na kojim strojevima se obavljaju pojedini zadatci, tada se radi o statičkom problemu (eng. *static*). Ako svi poslovi i vremena trajanja njihovih zadataka nisu poznati odmah na početku rada sustava nego poslovi dolaze kasnije u sustav tada je riječ o dinamičkom problemu (eng. *dynamic*).

Probleme raspoređivanja (eng. *scheduling problems*) možemo još podijeliti na determinističke i stohastičke. Deterministički problem ima poznata vremena trajanja i ostale parametre, dok stohastički ima nepoznate neke ili sve parametre.

Također postoji slučaj gdje ne postoji pravilo prednosti (eng. *precedence constraint*) što znači da nije unaprijed određen redoslijed strojeva kojim određeni posao mora ići nego se to može uraditi bilo kojim redoslijedom. Drugim riječima, zadatci unutar posla nemaju zadan redoslijed izvođenja nego se to može učiniti proizvoljno. Tada je riječ o otvorenom problemu (eng. *Open Shop Scheduling Problem (OSSP)*). Kod OOSP-a prostor rješenja je veći nego kod JSSP (eng. *Job Shop Scheduling Problem*). Slabije je zastupljen u literaturi iako je važan problem i može se primjeniti na situacije gdje je raspored zadataka unutar poslova proizvoljan, kao što su popravci automobila i nadogradnja računala [3].

Ako postoji identično pravilo prednosti za sve poslove tada je riječ o protočnom problemu *(*eng. *Flow Shop)*. Primjer u industriji je pokretna traka (eng. *assembly line*) koja se koristi u npr. automobilskoj industriji. Svaki posao (automobil) ima isti redoslijed strojeva koje mora proći.

Moguća je i kombinacija otvorenog problema (*Open Shop*), problema proizvoljne obrade (*Job Shop*) te protočnog problema (*Flow Shop*). Kod navedenog problema neki poslovi imaju zadano identično pravilo prednosti, neki pak imaju pravilo prednost ali im se razlikuje od drugih, a treći nemaju uopće pravilo prednosti. Ovi se problemi zovu miješani (eng. *Mixed Shop*) [8].

Još jedna inačica se može naći u literaturi[9]. Zove se grupni problem (eng. *Group Shop*). Kod ove definicije pravilo prednosti ne mora postojati na cijelom skupu

operacija unutar jednog posla, nego može varirati od toga da su u potpunosti neovisne (*Open Shop*) do slučaja da postoji redoslijed koji uključuje sve zadatke jednog posla (*Job Shop*).

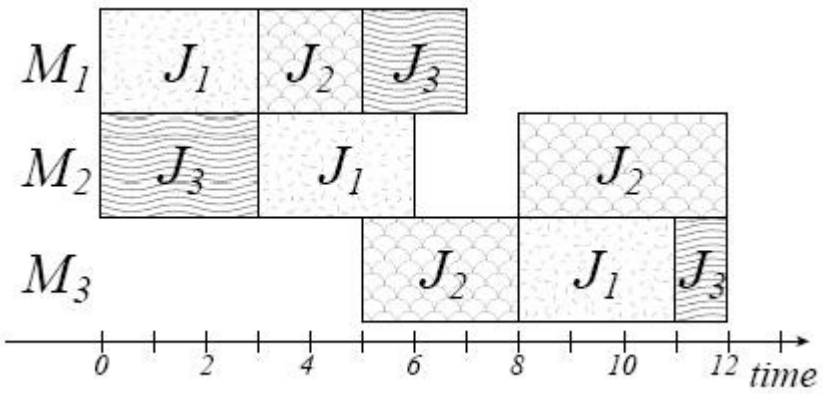
Problem kod kojeg jedan zadatak može biti procesiran na više strojeva zove se fleksibilni(klizni) JSSP (eng. *Flexible*).

Dodatni zahtjevi mogu biti rokovi završetka (eng.*deadlines*) gdje svaki posao ima definirano vrijeme kada bi trebao biti gotov. Ako se radi o dinamičkom problemu tada se zadaje i vrijeme kada posao može početi, odnosno vrijeme dolaska posla u sustav.

U ovome radu bavi se statičkim i dinamičkim problemom proizvoljne obrade (*Job Shop*). Zadatak je deterministički što znači da su vremena unaprijed zadana i jasno definirana.

2.4 Cilj optimizacije problema

Najjednostavniji i najraširenij cilj optimizacije je minimizirati ukupno trajanje (eng. *makespan*) koje počinje početkom prvog zadatka i završava krajem zadnjeg zadatka.



Slika 1-Ganttov dijagram za 3x3 problem

Na slici 1 je prikazan Ganttov dijagram za problem sa tri stroja i tri posla. Ukupno trajanje (*makespan*) je 12 vremenskih jedinica. U redcima su prikazani strojevi tako da prvi redak pripada prvom stroju itd. Svaki zadatak pojedinog posla je prikazan pravokutnikom čija je duljina jednaka trajanju te operacije.

Ukoliko problem ima definirane rokove do kada bi poslovi trebali biti završeni tada se mogu izračunati kašnjenja (eng. *lateness*) za svaki posao oduzimanjem stvarnog trenutka završetka posla i predviđenog završetka posla. Tada se mogu sva kašnjenja zbrojiti i tada se najbolji raspored smatra onaj sa najmanjim ukupnim kašnjenjem svih poslova. Može se promatrati pojedinačna kašnjenja tako da se uzme u obzir samo posao sa najvećim kašnjenjem.

Nekad se poslovima pridružuju težinski faktori pa tako važniji poslovi imaju veći težinski faktor. Tada se njihovo kašnjenje množi sa težinskim faktorom, sve skupa se zbraja i opet najbolji raspored se smatra onaj sa najmanjom ukupnom zakašnjelosti (eng. *tardiness*).

U literaturi još postoji i mjerilo kazne (eng.*unit penalty*). Ukoliko je posao zakasnio onda mu se pridružuje faktor 1 i to se množi sa težinskim faktorom. Ukoliko nije zakasnio pridružen mu je faktor 0. Opet se sve pojedinačne kazne zbrajaju i traži se ukupna suma kazni, koju se želi minimizirati. Slično kao mjera zakašnjelosti, samo se tamo gleda i koliko je posao zakasnio, dok se kod kazni samo gleda da li postoji kašnjenje.

Također postoji pristup koji rabi nešto statistike, a to je računanje standardne devijacije od podataka o kašnjenjima poslova. Može se računati i kvadratna devijacija.

Od ostalih ciljeva minimizacije spominje se i srednje vrijeme obrade (eng. *mean flow time*) gdje se promatra vrijeme od trenutka kada je posao spreman za obradu na strojevima do trenutka kada je obrada posla završena. Zbrajaju se vremena obrade za svaki pojedini posao i traži se prosjek.

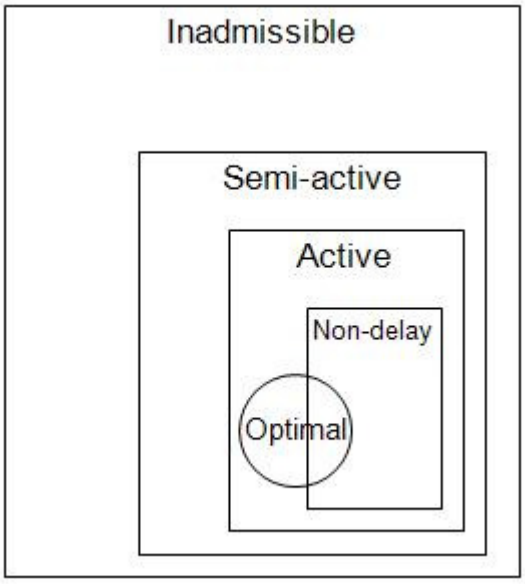
2.5 Vrste rješenja

Postoje 4 razreda izvedivih rješenja u JSSP-u. [4]

1) Rješenje s nedopustivim trajanjem izvođenja prelazi ukupno vrijeme koje je predviđeno za trajanje izvođenja svih poslova (eng. *inadmissible*). Takvo rješenje ima višak vremena u kojem je stroj neaktivan (eng. *excess idle time*).

To je vrijeme u kojem stroj ne radi ništa i može se poboljšati pomicanjem zadataka naprijed. Zbog toga vremena rješenje se nalazi u nedopustivom prostoru rješenja.

2) Kad se eliminira višak neaktivnog vremena, dobiva se **poluaktivni** raspored (eng. *semi-active schedule*). Takav raspored nema viška neaktivnog vremena, ali se još može poboljšati pomicanjem operacija (zadataka) prema naprijed bez kašnjenja drugih operacija. Rješenje se ne može poboljšati bez mijenjanja cijelog rasporeda [5] .
3) **Aktivno** rješenje (eng. *active*) je rješenje gdje se ukupno trajanje ne može poboljšati a da se ne prekrši pravilo prednosti (eng. *precedence constraint*). Skup aktivnih rješenja je podskup skupa poluaktivnih rješenja. Optimalno rješenje mora biti aktivno rješenje.
4) Postoji još jedan skup rješenja, a to je rješenje **bez odgode** (eng. *Non-delay*). U tome skupu rasporeda nijedan stroj se ne drži neaktivnim dok postoje operacije koje bi on mogao obrađivati. Bitno je naglasiti da optimalno rješenje nije uvijek rješenje bez odgode. Odnosi između tipova rješenja prikazani su u Slici 2.



Slika 2- Vrste rješenja

3. Pregled algoritama za rješavanje problema proizvoljne obrade

3.1 Matematičke metode

Ovaj skup algoritama ima matematičku verifikaciju te pronalazi optimalno rješenje a mana mu je što za veće inačice problema (sa većim brojem poslova i strojeva) ne može dati rješenje u realnom vremenu. Naime ukupan broj različitih rješenja u ovome problemu je $(n!)^m$. m je broj strojeva dok je n broj poslova. Tako se primjerice za problem 20x10 dolazi do spektakularne brojke od $7.2651 * 10^{183}$. Potrebno je napomenuti da je to najgori slučaj, jer zbog pravila prednosti nisu svi ti rasporedi ispravni [11].

3.1.1 Mješovito cjelobrojno linearno programiranje

Prepoznato je od strane mnogih istražitelja da se problem proizvoljne obrade može formulirati i naći optimum kao mješoviti cjelobrojni linearni program (eng. *mixed integer linear programming*). Prvo se mora definirati problem.

Zadana su dva skupa. Konačan skup poslova $\{\mathcal{J}_i\}_{i=1}^{n}$ koji se sastoji od n poslova te konačan skup od m strojeva $\{\mathcal{M}_k\}_{k=1}^{m}$. o_{ik} je operacija posla $\mathcal{J}_i$ koja se obavlja na stroju $\mathcal{M}_k$ u neprekinutom vremenskom razdoblju τ_{ik} .Zadatak je optimizacije naći vremena početka operacija $t_{ik} \geq 0$ takva da je ukupno trajanje najmanje. To je formalno prikazano u jednadžbi 1.

$$C^*_{max} = min\left(C_{max}\right) = \min_{feasible\ schedules} \left(max\left(t_{ik} + \tau_{ik}\right) : \forall\ \mathcal{J}_i \in \mathcal{J},\ \mathcal{M}_k \in \mathcal{M}\right). \quad (1)$$

Još je potrebno zadati pravilo prednosti (*precedence constraint*) te pravilo kapaciteta (eng. *capacity constraint, disjunctive constraints*) po kojem samo jedna operacija može biti na jednom stroju u određeno vrijeme.

Minimise C_{max} *subject to :*

starting times	$t_{ik} \geq 0$	$\{i, p\} \in \mathcal{J}$ $\{k, h\} \in \mathcal{M}$
precedence constraint	$t_{ik} - t_{ih} \geq \tau_{ih}$	if O_{ih} precedes O_{ik}
disjunctive constraint	$t_{pk} - t_{ik} + \mathcal{K}(1\text{-}y_{ipk}) \geq \tau_{ik}$	$y_{ipk} = 1$, if O_{ik} precedes O_{pk},
	$t_{ik} - t_{pk} + \mathcal{K}(y_{ipk}) \geq \tau_{pk}$	$y_{ipk} = 0$, otherwise

where $\mathcal{K} > \left(\sum_{i=1}^{n} \sum_{k=1}^{m} \tau_{ik} - min\,(\tau_{ik}) \right)$

Slika 3- Ulazni parametri za mješovito cjelobrojno programiranje

Kao što vidimo na slici 3 pravila kapaciteta (*disjunctive constraint*) su implementirana preko cjelobrojnih binarnih varijabli. $\mathcal{K}$ je veliki broj i da bi se ispravno definiralo područje izvedivih rješenja $\mathcal{K}$ mora biti veći od zbroja svih vremena obrade (trajanje jedne operacije ili zadatka) umanjenog za iznos najmanjeg od spomenutih vremena.

Unatoč konceptualnoj eleganciji ovog načina rješavanja, broj cjelobrojnih varijabli raste eksponencijalno što ga čini nepraktičnim za veće inačice problema.

3.1.2 Lagrangeovo pojednostavnjenje

Ovaj pristup (eng. *Lagrangian Relaxation*) koristi prethodni pristup s time što se drugačije odnosi prema cjelobrojnim ograničenjima koja su zadana. On ih izostavlja iz nejednakosti i dodaje u funkciju cilja sa pripadnim težinskim faktorom. Tako ako rješenje zadovoljava ograničenje bit će nagrađeno oduzimanjem određenog težinskom faktora, a ako u suprotnom slučaju uvjeti nisu zadovoljeni bit će kažnjeno dodavanjem težine što će smanjiti kvalitetu rješenja. Opisani način također zahtjeva veliko vrijeme izvođenja i također kao i prethodni nije primjenjiv za veće varijante problema proizvoljne obrade.

3.1.3 Metoda grananja i ograničenja

Ova metoda (eng. *Branch and Bound*) pripada skupini sustavnog pobrojavanja (eng. *enumeration*) kandidata gdje se veliki podskupovi beskorisnih rješenja odbacuju u grupama. Osnovna ideja ovog pristupa je zamisliti prostor rješenja kao stablo odluke. Pretraživanje počinje u korijenu stabla odakle se granaju moguća rješenja. Svaki čvor na razini *p* predstavlja djelomični redoslijed od *p* operacija. On ispušta onoliko

grana koliko je mogućih odluka. Taj se korak zove grananje (*branch*) i jedna od najpoznatijih strategija je SEC (eng. *Settling Essential Conflicts*) [11]. Različite grane imaju različitu odluku treba li operacija O_i biti postavljena prije operacije O_j ili obrnuto. Slijedi postupak koji se zove ograničavanje (*bound*). Prije početka pretrage definiraju se dvije vrijednosti. Gornja granica (eng. *upper bound*) ili UB te donja granica (eng. *lower bound*) ili LB. Gornju granicu se inicijalno postavlja na početku programa. Može je se postaviti na najveću vrijednost kojom smo zadovoljni. Naravno ako se radi o ukupnom trajanju cijelog rasporeda (*makespan*) kojom smo zadovoljni. Ako prilikom pretrage stabla naiđemo na rješenje koje ima bolju (manju) vrijednost od dotad aktualne gornje granice tada to rješenje postaje gornja granica i vrijednost se ažurira. Donja granica se računa preko posebne funkcije. Kada dođe do grananja čvora, za svaku granu se računa vrijednost spomenute funkcije. Funkcija predstavlja procjenu kvalitete koje može dati skup rješenja, dakle koliko će biti najbolje rješenje koje možemo pronaći u skupu koji nastaje grananjem čvora. Ukoliko je ta procijenjena donja granica veća (lošija) od trenutno važeće gornje granice (koja predstavlja dotad postignuti globalni optimum) smatra se da skup rješenja nije dovoljno dobar (lošiji od onoga koji je već postignut) i odbacuje ga se iz daljnje pretrage. Ovaj se korak zove podrezivanje(eng.*pruning*). Kada se dohvate listovi (čvorovi koji se ne mogu više granati i koji predstavljaju pojedinačna rješenja), algoritam se vraća do najvišeg čvora u stablu čiji listovi nisu još dohvaćeni.

U najboljem slučaju, kada vrijeme izvođenja nije imperativ procedura završava kada su svi čvorovi ili riješeni ili podrezani. Ako se nema dovoljno vremena na raspolaganju onda se postavlja neki kriterij kako što je maksimalan broj iteracija ili ako se najbolja i najlošija vrijednost rješenja dovoljno približe, npr. ako je $\frac{max-min}{max+min}$ dovoljno mali broj.

U načelu ovaj pristup je neprimjenjiv za velike inačice problema zato što zahtjeva veliko vrijeme izvođenja. To je razlog zašto su mnogi istraživači svoju pažnju poklonili približnim metodama.

3.2 Približne metode

Iako ne jamče optimalno rješenje približne metode (eng. *Approximate metods*) su pogodnije za veće varijante problema jer osiguravaju dovoljno dobro rješenje a završavaju u realnom vremenu.

Dijele se na:

- Raspoređivanje po prioritetu
- Heuristika uskog grla
- Umjetna inteligencija
- Metaheurističke metode

3.2.1 Pravila raspoređivanja po prioritetu

Približne metode za rješavanje problema proizvoljne obrade su prvi put predstavljene u obliku metoda raspoređivanja po prioritetu (eng. *Priority Dispatch Rules*). Zbog lakoće implementiranja i malih računalnih zahtjeva vrlo su popularna tehnika. Ideja je da se na svakom koraku algoritma iz skupa operacija koje se mogu u tom trenutku dodijeliti nekom stroju pripisuje prioritet [11].

Pravila raspoređivanja po prioritetu se mogu podijeliti u tri razreda [12]:

1) Jednostavna pravila, zasnivaju se na značajkama o poslovima. Navedeni su neki primjeri:

- Nasumično (eng. *random*). Slučajnim odabirom iz konfliktnog skupa se odabire jedna operacija koja će biti prva obrađena.
- FIFO (eng.*First in, first out*) raspoređivanje na osnovu toga koja je operacija prva dostupna za obradu.
- SPT (eng. *shortest processing time*) odabire se posao koji je preostalo najmanje vremena za izvođenje od svih poslova.
- DD (eng. *deadline*) izabire onaj posao koji ima najraniji rok do kada bi se trebao završiti.
- Giffler-Thompsonov algoritam. Ovo se danas smatra temeljem svih složenih pravila prioriteta a važnost mu potječe iz činjenice da izrađuje aktivna rješenja. Procedura započinje pronalaskom prve još nedodijeljene operacije

sa najmanjim vremenom izvođenja. Zatim se u sporni skup stavljaju sve operacije koje se izvode na istom stroju kao i prethodno spomenuta operacija, a imaju svojstvo da mogu započeti ranije nego što je trenutak završetka operacije sa najmanjim trajanjem. Nakon toga se iz toga skupa odabire jedan zadatak koji će stvarno biti obrađen. Postupak se ponavlja dok sve operacije nisu obrađene [11].

- NINQ (eng. *number in next queue*) izabire posao tako da sljedeća operacija koristi stroj koji ima najkraći red čekanja.

2) Drugi razred se sastoji od kombinacija pravila prvog razreda. Tipični primjer je korištenje SPT-a dok red čekanja ne naraste na pet operacija, a zatim prelaženje na FIFO. Ova kombinacija pravila sprječava poslove s dugim vremenom izvršavanja od predugog stajanja u redovima čekanja.

3) Treći razred sadržava pravila koja se obično zovu pravila težinskih pokazatelja (eng. *Weight Priority Indexes*). Ideja je upotrijebiti više od jednog podatka o poslovima da bi se izradio raspored. Podatcima se dodjeljuju težinski faktori koji odražavaju njihovu relativnu važnost. Definira se ciljna funkcija f(x). Primjerice: f(x) = TežinskiFaktor1 x UkupnoTrajanjePosla(x) + TežinskiFaktor2 x (TrenutnoVrijeme – PredviđenoVrijemeZavršetkaPosla(x)). Svaki put kada se nova operacija raspoređuje funkcija se izračunava za svaki pojedini posao. Poslovi se sortiraju na osnovu rezultata ove funkcije, onaj posao sa najvećom vrijednosti f(x) biva sljedeći raspoređen.

Različiti kriteriji optimizacije imaju svoja posebna pravila prioriteta. SPT se smatra dobrim izborom za minimizaciju srednjeg vremena prolaska poslova kroz sustav (eng. *mean flow time*). Inače za druge ciljeve nijedno pravilo jednostavno pravilo ne dominira već se za dobre rezultate trebaju koristiti kombinacije više pravila.

Vidljivo je da pravila prioriteta odabiru samo jednu operaciju i dodaju je u postojeći raspored dok metoda grananja i ograničavanja razmatra sve operacije, implicitno ili eksplicitno. U literaturi [11] postoji i pristup koji balansira ova dva načina. Koristi se

varijanta pretraživanja prostora stanja i to optimizacija pretraživanja najboljeg prvog (eng. *best first strategy*). Zove se pretraživanje pomoću snopa (eng.*beam search*). Razlika u odnosu na metodu grananja i ograničavanja je da se samo određeni broj najboljih čvorova razmatra u pretraživanju. To smanjuje vrijeme izvođenja, ali postoji rizik da se i optimalno rješenje odbaci(podreže). Pravila prioriteta se koriste da bi se izračunale donje granice (*lower bounds, LB*) potencijalnih rješenja. Ovaj pristup je polučio dobar uspjeh i njegovi najbolji rezultati se razlikuju od optimalnih za 8.3%. Ali odstupanje od optimuma je još uvijek veliko, s time da se vrijeme izvođenja u odnosu na pojedinačna pravila prioriteta povećalo za tri reda veličine. Ovo je dovelo do zaključka da bi se pravila prioriteta trebala koristiti više za stvaranje inicijalnog rješenja, a ne kao potpuni sustav za rješavanje problema proizvoljne obrade [11].

3.2.2 Heuristika uskog grla

Dugo godina pravila prioriteta su bila jedina približna metoda za probleme raspoređivanja. Pojava sve jačih računala kao i naglasak na pažljivo dizajnirane, dobro proučene i implementirane tehnike doveo je do profinjenijih pristupa koji su mogli premostiti jaz između pravila prioriteta i zahtjevnih egzaktnih metoda. Jedan takav pristup je i heuristika uskog grla (eng. *Shifting bottleneck heuristic*) [11].

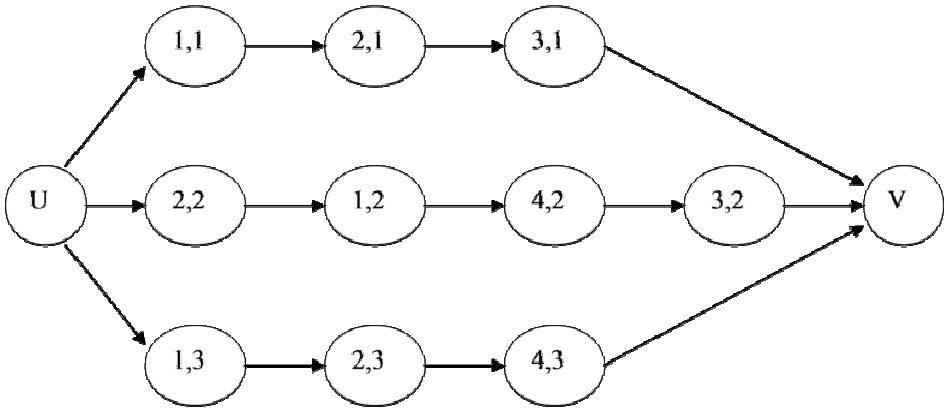
Metoda će biti objašnjenja na primjeru koji je preuzet iz literature [6].

Inačica problema ima 3 stroja i 4 posla.

Job	Machine Sequence	Processing Times
1	1,2,3	$p_{11} = 10, p_{21} = 8, p_{31} = 4$
2	2,1,4,3	$p_{22} = 8, p_{12} = 3, p_{42} = 5, p_{32} = 6$
3	1,2,4	$p_{13} = 4, p_{23} = 7, p_{43} = 3$

Slika 4- 3x4 problem sa zadanim vremenima obrade na strojevima u obliku p_{ij} gdje je i broj koji predstavlja stroj, a j broj koji predstavlja posao

Na početku treba definirati način rješavanja koji se koristi ne samo u ovoj metodi već i u drugima, a to je grafički način. Svaki posao ima izvor (označen na slici 5 s U) te ponor (označen na istoj slici s V). Pojedini poslovi su predstavljeni redovima u grafu. Svaki čvor je jedna operacija (zadatak). Bridovi grafa su usmjereni tako da se poštuje pravilo prednosti.



Slika 5 – Grafički prikaz problema proizvoljne obrade. U čvorovima grafa se nalaze dva broja gdje prvi broj označava stroj, a drugi broj posao

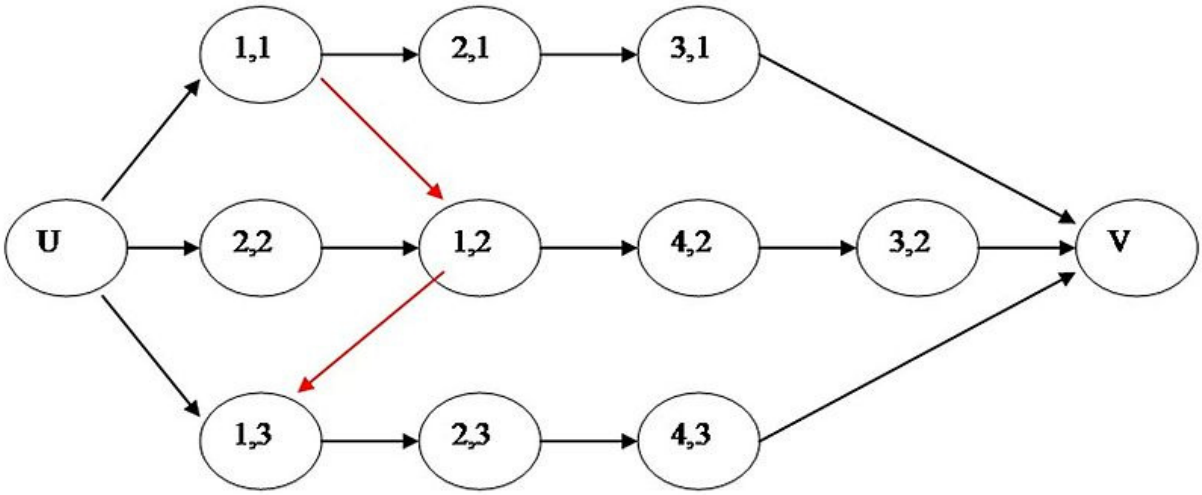
Cilj optimizacije je minimizirati ukupno vrijeme izvođenja (*makespan*). Nakon što je problem ovako definiran preko grafa izračunava se inicijalna vrijednost ukupnog trajanja. Sva vremena obrade pojedinih operacija unutar pojedinog posla se zbrajaju i traži se posao sa najdužim trajanjem. To je početna vrijednost ukupnog trajanja. To bi bilo rješenje problema da ne postoji sukob oko resursa jer se dvije operacije ne mogu obavljati u isto vrijeme na istom stroju. U ovom slučaju ta vrijednost je 22.

Sljedeći korak je prva iteracija algoritma a sastoji se u odabiru koji je od strojeva usko grlo (eng. *bottleneck*) . Prvo se određuju parametri za svaki stroj posebno. Na tablici 1 vidimo te parametre za prvi stroj. U prvom retku su navedenim poslovi. U drugom retku su vremena trajanja obrade pojedine operacije na datom stroju. U trećem retku su vremena prispijeće operacija. Ona ovise o pravilu prednosti i jednaka su zbroju trajanja obrade svih operacija koje su prije promatrane operacije u spomenutom pravilu. Kako se vidi slici 4 operacija drugog posla je druga po redu i mora čekati 8

vremenskih jedinica da se počela obavljati. Četvrti redak je ključan parametar i to je kašnjenje (eng. *Lateness*). Računa se za svaki posao na svakom stroju. Traži se usmjerena staza u grafu koja predstavlja poredak operacija na jednom stroju koja će minimizirati najveće od svih kašnjenja na tom stroju. U ovom slučaju je prikazana na slici 6. Odabrana je staza tako da se prvo obavlja posao 1, zatim posao 2, a treći je na redu posao 3. Tako se dobivaju brojevi koji predstavljaju kašnjenja. Prvi posao ne kasni ništa jer ne čeka nikoga i prvi se obavlja na stroju 1. Drugi posao kasni 2 vremenske jedinice jer čeka prvi posao koji je zauzeo prvi stroj. Treći posao kasni najviše (5) jer čeka i prvi i drugi posao da mu ustupe prvi stroj.

poslovi	1	2	3
p_{1j}	10	3	4
r_{1j}	0	8	0
L_{1j}	0	2	5

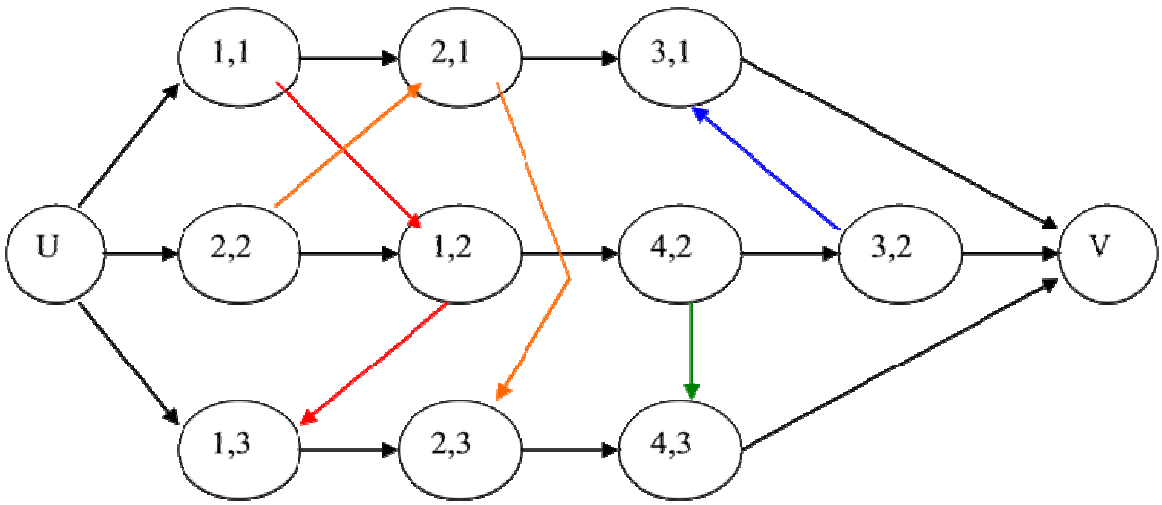
Tablica 1- Parametri za prvi stroj koji se koriste za traženje uskog grla



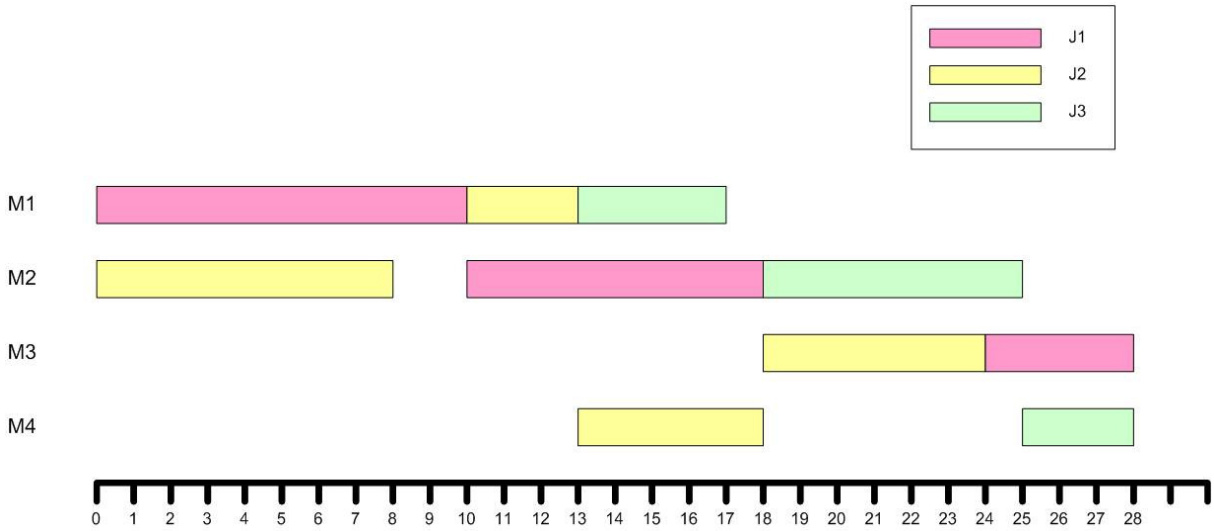
Slika 6 – Prva iteracija algoritma.

Ovaj se postupak obavlja za svaki stroj i traži se stroj koji ima najveće kašnjenje. U ovom slučaju to je prvi stroj i on se proglašava uskim grlom. Ako postoje dva stroja sa jednakim kašnjenjem bilo koji od njih se može odabrati kao usko grlo. Nakon što je usko grlo određeno, staza sa dva nova brida mora biti razmatrana u daljnjim proračunima. Ukupno trajanje se ponovno izračunava s tim novim disjunktivnim ograničenjima koja potječu iz fizikalnog zahtjeva da se više poslova ne može obavljati na istom stroju u isto vrijeme. Najduži put da se dođe do operacije (čvora u grafu) je novo vrijeme prispijeća.

U drugom koraku od preostalih strojeva se ponovno odabire usko grlo. Postupak je isti kao u prvoj iteraciji. Novo ukupno trajanje je jednako zbroju ukupnog trajanja u prethodnom koraku i najvećeg kašnjenja novog uskog grla. Ako je najveće kašnjenje na svim strojevima nula, onda se sva disjunktivna ograničenja (staze) crtaju na grafu. Algoritam završava kada su svi strojevi obrađeni i njihove staze ucrtane na grafu kako je prikazano na slici 7.



Slika 7 – Posljednja iteracija algoritma



Slika 8-Ganttov dijagram optimalnog rješenja. U redcima su strojevi, a operacije su predstavljene pravokutnicima. Operacije istog posla su prikazane istom bojom. Na osi apscise je vrijeme trajanja. Ukupno trajanje dok se završe svi poslovi iznosi 28 vremenskih jedinica [13].

Dakle kao što je vidljivo iz primjera, premještanje uskog grla se sastoji od više postupaka koji se iterativno ponavljaju. To su: identifikacija podproblema, odabir uskog grla, rješenje podproblema i reoptimizacija cijelog rasporeda. Odabiranje uskog grla i njegova optimizacija u ranom dijelu postupka imaju motivaciju u pretpostavci da bi to isto u kasnijoj fazi dovelo do pogoršanja rasporeda u vidu većeg vremena ukupnog trajanja [11]. Općenita slabost ove metode je zahtjevnost programiranja te činjenica što cijela procedura mora biti završena da bi se dobilo rješenje. Također nema načina da se odredi veličina podproblema i raspored kojeg stroja treba biti popravljen da bi se poboljšalo rješenje. Metoda bi trebala modifikaciju za bolji rad na inačicama problema gdje je usmjeravanje poslova složenije. Unatoč ovim problemima, metoda se koristila u mnogim radovima koji su poboljšali gornje i donje granice teških problema [11].

3.2.3 Umjetna inteligencija

3.2.3.1 Ekspertni sustavi

Ekspertni sustavi (eng. *expert systems*) i sustavi na bazi znanja (eng. *knowledge-based systems*) su bili predvladavajući od metoda umjetne inteligencije na ovom području kada su se pojavili '80tih godina prošlog stoljeća. Oni imaju četiri glavne prednosti: [12]

1) Možda najvažnija pogodnost je korištenje kvalitativnih i kvantitativnih znanja u donošenju odluka.
2) Sposobni su stvoriti heuristike koje su mnogo složenije od jednostavnih pravila prioriteta koje su opisane ranije.
3) Najbolja heuristika može biti donesena u prema podatcima o cijelom problemu kao što su trenutni poslovi, očekivani novi poslovi, trenutno stanje resursa, robni fond te ljudstvo.
4) Skupljaju složene relacije u novim elegantnim strukturama podataka i imaju specijalne tehnike za manipulaciju istih podataka

Postoje i ozbiljni nedostatci:

1) Mogu biti vremenski zahtjevni za izradu i verifikaciju, kao i za održavanje i ažuriranje.
2) Proizvode samo izvediva rješenja, pa je teško moguće odrediti koliko je rješenje blisko optimalnome.
3) Konačno, budući da su povezani usko sa sustavom za koji su izrađeni da bi njime upravljali, ne mogu napadati općenite probleme

Sastoje se od baze znanja i sustava za zaključivanje. U bazi znanja se nalaze formalizacije znanja koje koriste ljudski stručnjaci u pravila, procedure, heuristike i ostale apstraktne vrste. Više struktura podataka se nalazi u bazi znanja kao što su semantičke mreže, okviri, skripte, predikatna logika i produkcijska pravila. Sustav za zaključivanje odabire strategiju koja se primjenjuje na bazu znanja za rješenja problema. Može biti ulančana prema naprijed(upravljana podatcima) ili ulančana prema natrag (upravljana ciljem).

Istraživanja su pokazala da kombinacija ekspertnih sustava sa drugim heuristikama donosi dobre rezultate [11]. Nažalost ove tehnike imaju veliko vrijeme izvođenja jer su slične metodi grananja i ograničavanja.

3.2.3.2 Raspodijeljena umjetna inteligencija: Agenti

Zbog ograničenog znanja i sposobnosti rješavanja problema pojedinačnog ekspertnog sustava takvi sustavi imaju problema u rješavanju velikih inačica problema. To je dovelo do razvoja raspodijeljenih sustava umjetne inteligencije (eng. *distributed artificial inteligence (AI)*). Pri tome se koristio princip *podijeli pa vladaj.* To zahtijeva dekompoziciju problema kao i razvoj drugačijih ekspertnih sustava koji će moći surađivati da bi riješili ukupni problem. Odgovor AI zajednice je paradigma agenta. Agent posjeduje potpuni sustav znanja u sebi. Skup agenata može biti heterogen u odnosu na dugoročno znanje, kriterij ocjenjivanja rješenja te ciljeve optimizacije kao i jezike, algoritme i sklopovske zahtjeve. Višeagentski sustav se integriranjem agenata koji se biraju iz biblioteke [12].

Primjera radi, višeagentski sustav može uključivati dva tipa agenata: jednu grupu orijentiranu na zadatke te drugu grupu orijentiranu na resurse. Svaki agent za zadatke bi bio odgovoran za rukovanje i nadziranje na resursima koji su sposobni za obavljanje tih zadataka. To se može obaviti bilo kojim mjerilom izvedbe koje se odnosi na zadatke, kao što je kašnjenje poslova. Svaki resursni agent bi bio odgovaran za jedan resurs ili za klasu resursa. Agenti za zadatke moraju poslati zahtjeve resursnim agentima zajedno sa popisom operacija koje će se obavljati na tim resursima. Primivši zahtjev, resursni agent mora izraditi novi raspored koristeći vlastito mjerilo izvedbe, kao što je maksimalno iskorištenje strojeva. Ovisno o rezultatu agent odlučuje o prihvaćanju zahtjeva. Da bi se izbjegla situacija u kojoj nijedan resursni agent ne prihvaća zahtjev, sustav koordinacije mora biti razvijen. Ne postoji opća smjernica za dizajn i implementaciju ove koordinacije. Vode se i debate o tome je li bolji centralizirani ili decentralizirani pristup ovom problemu.

3.2.3.3 Neuronske mreže

Neuronske mreže(eng. *neural networks, NN*) su okviri utemeljeni na moždanoj strukturi jednostavnih životnih entiteta. U ovim tehnikama podatak se prenosi kroz mrežu povezanih procesnih jedinica koji se zovu neuroni. Njihova jednostavnost, zajedno sa sposobnošću raspodijeljenog računanja kao i njihova sklonost učenju i generaliziranju učinila ih je popularnim alatom u mnogim aplikacijama.

Najrašireniji primjer su mreže sa propagacijom greške unatrag(eng. *Back-Error Propagation Networks*) [11]. One su trenirane na primjerima koji imaju svojstvo preslikavanja f: $S \subset \Re_n \rightarrow \Re_m$, iz proizvoljnog podskupa S n-dimenzionalnog Euklidskog prostora u m-dimenzionalni Euklidski prostor. Kada se uzorak pojavi na ulazu mreže pravilo ispravljanja pogreške podešava sinaptičke veze u skladu sa zadanim preslikavanjem. Stvarni odziv mreže se oduzima od željenoga da bi se dobio signal pogreške. Težinski faktori između neurona se ažuriraju da bi se stvarni odziv približio željenome odzivu.

Primjeri za učenje mogu biti iz stvarne primjene, a mogu biti i generirani preko pravila prioriteta.

U literaturi[11] se navodi kako modeli neuronskih mreža sa propagacijom greške unatrag pate od neuspješnog učenja i konvergencije prema lokalnom optimumu kada se suoče sa problemima složenog ulazno-izlaznog preslikavanja. Ne mogu riješiti generičke probleme zato što su uspješne samo ako se testni podatci ne razlikuju od podataka iz skupa za učenje za više od 20%.Neusklađenosti u vezi sa neuronskim mrežama uključuju pretjerane zahtjeve za neuronima te učenje iz ne-optimalnih primjera koji su pribavljeni od strane eksperta, postojećih podataka ili pravila prioriteta. NN se koriste u kombinaciji s drugim metodama, i u većini slučajeva ta druga metoda se rabi za optimizaciju dok se NN koriste za brzi dohvat podataka iz baze. Trenutno se NN ne smatraju konkurentnima sa najboljim heuristikama na nijednom razredu optimizacijskih problema [11].

Rezultati su pokazali da metode umjetne inteligencije ne daju dobra rješenja, a imaju velike računalne zahtjeve.

3.2.4 Metaheuristički algoritmi

Metaheuristički algoritmi se primjenjuju na probleme gdje ne postoji odgovarajući usko vezani problemski algoritam ili kada ga je nepraktično implementirati [6]. Pokazali su se uspješnima u rješavanju NP-teških problema gdje ne postoje načini rješavanja polinomske složenosti. Ovo područje sadrži mnoštvo algoritama, a najvažniji za problem proizvoljne obrade i oni koji će biti detaljnije obrađeni su:

- Simulirano kaljenje
- Tabu pretraživanje
- Genetski algoritam

3.2.4.1 Simulirano kaljenje

Pripada stohastičkim optimizacijskim algoritmima, a vrlina mu je što omogućava izlazak iz lokalnog optimuma.

Ideja za algoritam pronađena je u industriji [14]. Naime, da bi se metal očvrsnuo potrebno je postići da njegova kristalna rešetka ima minimalnu potencijalnu energiju. Zagrijavanje uzrokuje da se atomi pomaknu iz svoje inicijalne pozicije, dođu u stanje više energije, da bi im postupno hlađenje omogućilo da pronađu stanja sa manjom internom energijom nego što su imali na početku postupka.

U skladu sa industrijskom analogijom, definira se parametar temperature koji u početku ima visoku vrijednost da se postupno smanjivao. Početno rješenje se odabire slučajnim odabirom iz prostora stanja. Svako sljedeće rješenje se odabire iz susjedstva prethodnog. Prihvaća se ako je bolje, a može se prihvatiti i ako je lošije ali pri tom more biti zadovoljena vjerojatnost prihvaćanja. Ovaj dio algoritma prihvaćanja lošijeg rješenja omogućuje bijeg iz lokalnog optimuma. Veća vrijednost temperature znači i veću vjerojatnost dopuštanja lošijeg rješenja od prethodnoga. U početku s visokom vrijednošću temperature prostor pretraživanja je velik da bi se daljnjim iteracijama algoritma te padom temperature lokalizirao [14]. Metoda je osjetljiva na izbor parametara. Ako se temperature brzo hladi veća je vjerojatnost zarobljavanja u lokalnom minimumu. Za presporo hlađenje temperatura je visoka i algoritam češće zamjenjuje bolja rješenja lošijima i ima svojstvo slučajnog pretraživanja velikog prostora stanja. Temperatura mora biti u početku dovoljno visoka da bi se izbjegli

lokalni optimumi, a na koncu dovoljno niska da bi se algoritam zaustavio u globalnom minimumu.

Glavni nedostatci algoritama simuliranog kaljenja su pretjerana vremena izvršavanja da bi se postigla dobra rješenja te visoka ovisnost postupka o inačici problema gdje se više parametara mora pažljivo podesiti. Mogući razlozi zašto je potrebno toliko mnogo iteracija su da ove metode primjenjuju manje informacija o problemu nego druge metode koje se bave problemom proizvoljne obrade i problemima raspoređivanja. Iako sâmo ne polučuje dobre rezultate, sjedinjeno sa drugim metodama kao što su genetski algoritmi ili heuristika uskog grla može dati vrlo dobre rezultate.

3.2.4.2 Tabu pretraživanje

Algoritam koristi lokalno pretraživanje prostora stanja uz pomoć tabu liste na kojoj se nalaze rješenja koja su zabranjena i ne mogu se pojaviti u sljedećem koraku postupka. U listi tabua se nalaze ona rješenja koja su već bila odabrana kao najbolja u prethodnih nekoliko koraka. Time se izbjegavaju lokalni optimumi i cikličko ponavljanje rješenja. Iduće rješenje se bira pretraživanjem susjedstva postojećeg rješenja, uz spomenuti uvjet da se ne nalazi na popisu zabranjenih rješenja [14]. Svaki put kada se odabere novo rješenje, ono rješenje koje je najduže bilo u tabu listi se izbacuje. Ponašanje algoritma ovisi o duljini tabu liste. Ukoliko je premala može doći do cikličkog ponavljanja, a ukoliko je prevelika algoritam može postati prespor.

Metoda tabu pretraživanja se smatra najboljom u domeni problema proizvoljne obrade zbog ukupno izvrsnih rezultata koji uključuju kriterije kvalitete rješenja i vremena izvođenja. Kao i ostale heurističke pretrage, tabu pretraživanje zahtjeva fino podešavanje parametara, što nije uvijek lagan zadatak [11].

3.2.4.3 Genetski algoritam

Iako genetski algoritam pripada metaheurističkim strategijama, budući da je tema ovog diplomskog rada, bit će podrobnije proučen u sljedećem poglavlju. Ovdje će biti spomenuto da se kvaliteta rada genetskog algoritma kao samostalnog alata ne smatra konkurentnom, ali algoritmi koji koriste više metoda, a jedna od njih su i genetski algoritmi, prilično su visoko ocijenjeni. Tim postupcima pripada i genetsko lokalno pretraživanje. Tu se koristi genetski algoritam kao osnova, s time da se produkt operatora križanja ne koristi izravno u sljedećoj generaciji nego postaje ishodište lokalnog pretraživanja. Tako se pronalazi lokalni optimum u susjedstvu potomka križanja koji kao takav ulazi u narednu iteraciju genetskog algoritma. Ova se metoda smatra dosta dobrom u rješavanju problema proizvoljne obrade.

3.3 Zaključak

Studije objavljene u literaturi [11] zaključuju da najbolje rezultate daju hibridni mehanizmi, tj.algoritmi koji koriste više metoda i koji uključuju heuristiku uskog grla, tabu pretraživanje, genetsko lokalno pretraživanje i simulirano kaljenje.

4. Genetski algoritmi

4.1 Motivacija analogijom u prirodi

Jedno od najvećih otkrića u povijesti biologije i znanosti u cjelini je svakako teorija evolucije. Nju je prvi zapazio Charles Darwin u svome djelu *O podrijetlu vrsta*, 1859. godine. Tu je iznio svoje teze kako su se živi organizmi razvili postupno kroz milijarde godina, počevši od jednostaničnih ameba pa sve do čovjeka. Primijetio je da se broj jedinki pojedine vrste u dužem periodu u načelu ne povećava iako svaka vrsta stvara više potomaka nego je ukupna trenutna populacija vrste. Razlog je tome što dio populacije ne preživi borbu za opstanak i ugine prije nego što se stigne reproducirati [15]. Resursi kao što su hrana i voda su ograničeni i ne povećavaju se sa vremenom. Zaključio je da jedinke sa boljim svojstvima (otpornost na bolesti, motoričke sposobnosti, razvijenost osjetila, itd.) preživljavaju i prenose svoje osobine na potomstvo, dok one lošije umiru, a sa njima nestaju i njihove lošije osobine. Jedna takva izmjena generacija stvara novu populaciju, bolje prilagođenu za život u svome okolišu. Nakon više takvih generacija i nakupljanja promjena dolazi do formiranja potpuno novih vrsta. Također je činjenica da ne postoje dvije potpuno jednake jedinke u populaciji.

Dva su glavna pokretača evolucije: [16]

- Prirodna selekcija ili način na koji priroda izdvaja dobre jedinke za preživljavanje u borbi za opstanak
- Razmnožavanje čime se čuva raznolikost populacije koja generaciji omogućuje široki raspon značajki koje će im omogućiti prilagodljivost i preživljavanje (barem nekima) u sljedećoj prirodnoj selekciji

Kasnijim istraživanjima u dvadesetom stoljeću otkriveno je da su značajke jedinke određene nizom nukleotida u molekuli DNK (deoksiribonukleinska kiselina) koji se nazivaju genima. Svaki gen je odgovoran za jedno svojstvo organizma (npr. visina, boja očiju, broj zubi...). Potomak nasljeđuje dva gena za svako svojstvo: jedan od oca, jedan od majke [15]. U tome genetskom paru geni mogu biti ravnopravni tako da će rezultantno svojstvo biti negdje između svojstava koje imaju otac i majka, ili jedan

gen može biti dominantan, a drugi recesivan. U ovome drugom slučaju samo se osobina dominantnog gena prenosi na dijete.

Procesima križanja i mutacije se definira genetski materijal jedinke.

- Križanje se događa prilikom spolnog razmnožavanja. Pola kromosoma (strukture u kojoj se nalazi DNK) daje jedan roditelj a polovicu drugi. Prilikom križanja dolazi do izmjene gena. Rezultat je potpuno nova i jedinstvena stanica iz koje će se razviti isto tako novi i jedinstveni organizam.
- Drugi proces kojim se dobiva novi genotip je mutacija. Mutacija je slučajna promjena gena. Vjerojatnost mutacije i njezina zastupljenost u evoluciji je manja nego je to u slučaju križanja. Uloga joj može biti velika jer se može dobiti potpuno novo svojstvo koje može biti krucijalno u borbi za opstanak (npr. povećana otpornost na bolest), ali djelovanje može biti također negativno pri čemu jedinka neće preživjeti (npr. neki tjelesni defekt ili tumor). Vjerojatnost mutacije gena nije ista za sve gene i kreće se od 0.001% do 0.01% [15].

Procesima prirodne selekcije, križanja i mutacije odvija se spori proces evolucije koji se doveo do razvoja života na planetu Zemlji kakav je danas poznat.

4.2 Općenito o genetskom algoritmu

Ideja o nastanku algoritma se rodila kao pokušaj simulacije prirodnog procesa evolucije. Evolucija je sama po sebi postupak optimizacije koji očito daje dobre rezultate. Iako njezina primarna svrha nije pronaći najbolju jedinku (rješenje), nego prilagođavanje i pokušaj preživljavanja populacije na nove uvjete u okolišu, može se i tako promatrati. Najbolja (najsposobnija) jedinka ima najveću vjerojatnost preživljavanja.

Treba i napomenuti da je evolucija mnogo složeniji proces od genetskog algoritma. Ukupan genetski kod živih bića je puno veći nego onaj koji se koristi u računalnoj simulaciji. Da bi se zapisao stvarni kod samo jedne jedinke trebalo bi oko 1GB memorijskog prostora. To nije praktično za stvarne problema, a i bilo bi mnogo redundantnih podataka .

Genetski algoritam je prvi put predložen od strane dr. John H. Hollanda u 1970-im godinama. [15] On ga je zamislio kao računalni postupak koji koristi apstraktne jedinke. Svaka jedinka predstavlja rješenje problema koji se optimira, a sve su predstavljene jednakom podatkovnom strukturom. Ovisno o prikazu problema koji se obrađuje, jedinka može biti prikazana brojem, nizom, matricom, grafom, stablom... Jedinke se u skladu sa analogijom u prirodi nazivaju kromosomima. Svaka jedinka ima vlastitu vrijednost funkcije cilja ili dobrote. Bolje jedinke imaju bolji rezultat te funkcije što im daje veću vjerojatnost preživljavanja. Broj kromosoma u populaciji je stalan. Na populaciju se primjenjuju genetski operatori selekcije, križanja i mutacije. Selekcija oponaša borbu za opstanak gdje se uklanjaju loše jedinke. Slabije jedinke imaju veću vrijednost funkcije cilja (ukoliko se radi minimizacija) ili pak manju (ukoliko se radi maksimizacija). One se uklanjaju iz postupka. Zatim se primjenjuje operator križanja gdje mogu sudjelovati samo kromosomi koji su prošli operator odabira. Križanjem i razmjenom genetskog koda između boljih jedinki stvaraju se jedinke koje će zauzeti mjesto onih jedinki koje su otpale u selekciji. Još se primjenjuje i operator mutacije koji ima malu vjerojatnost upotrebe, a koji služi da se obnovio izgubljeni genetski materijal i izbjegao lokalni optimum.

Svaka iteracija ili generacija algoritma ima selekciju, križanje i mutaciju. Zaustavljanje algoritma se može zadati na više načina.

- Može se unaprijed definirati broj iteracija.
- Ako je pronađeno dovoljno dobro rješenje također se može zaustaviti algoritam.
- U nekim primjenama algoritam se zaustavlja i ako je veliki broj članova populacije dovoljno blizu u prostoru rješenja.

Početna (inicijalna) generacija populacije se može zadati na više načina [15].

1. Jedan je da se slučajnim odabirom (posredstvom generatora pseudoslučajnih brojeva) formiraju iz prostora rješenja. Ovaj je postupak dosta čest
2. Drugi je način da se jedinke ne generiraju slučajno nego se dobivaju kao izlaz iz neke druge optimizacijske procedure. Primjerice, pravilo prioriteta Giffler-Thompsonov algoritam koji proizvodi samo aktivna rješenja koristi se za stvaranje inicijalne generacije kod problema proizvoljne obrade

3. Treći način koji nije popularan je da generiraju sve međusobno iste jedinke. Tada je potrebno pričekati određeni broj generacija dok se genetska raznolikost dobije radom genetskih operatora.

Sljedeći odjeljak prikazuje pseudokod jednostavnog genetskog algoritma.

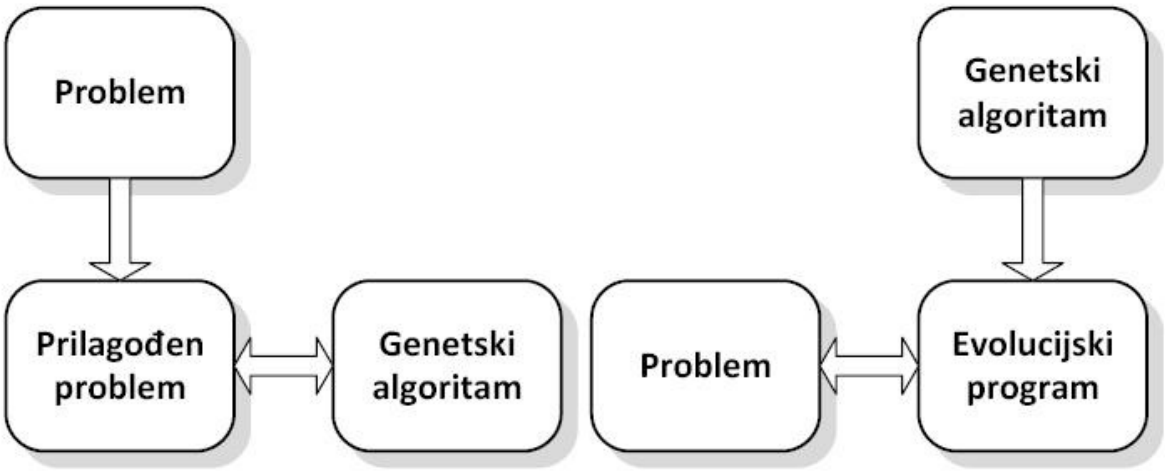
```
Procedura GenetskiAlgoritam
{
t=0;
generiraj inicijalnu populaciju potencijalnih rješenja P(0);
sve dok nije zadovoljen uvjet zaustavljanja radi
     {
     t=t+1;
     selektiraj P'(t) iz P(t-1);
     križaj jedinke iz P'(t) i djecu spremi u P(t);
     mutiraj jedinke iz P(t);
     }
Ispiši rješenje;
}
```

Postoje dva jednako popularna pristupa rješavanju problema pomoću genetskog algoritma [16].

1) prilagoditi genetski algoritam značajkama problema.
2) prilagoditi problem općenitijem genetskom algoritmu.

U prvom slučaju prilagodba algoritma problemu zahtjeva posebno definirane strukture podataka i genetske operatore koji su usko vezani za domenu problema. Ovime se dobiva usko specijalizirani algoritam, koji se tada često naziva *evolucijskim programom.* Takve implementacije imaju obično dobru učinkovitost i komercijalno su isplative. Loša strana je što su ovisni o problemu za koji su namijenjeni, te svaka promjena problema traži dodatno prilagođavanje. U programskoj implementaciji koja prati ovaj diplomski rad koristi se ovaj pristup.

U drugome slučaju problem se prilagođava genetskom algoritmu. Koriste se jednostavne strukture podataka kao što kromosom u obliku niza bitova. Kod ovoga pristupa negativno je što ne mora svaki niz bitova predstavljati izvedivo rješenje. Tada je potreban oporavak od pogreške. To dodatno usporava algoritam.



Slika 9- Dva načina pristupa problemu. U ovome radu je odabran prvi način

Parametri genetskog algoritma uključuju vjerojatnost mutacije, veličinu populacije, broj generacija(iteracija), te broj jedinka za eliminaciju u svakoj generaciji. Svaki pojedini problem zahtjeva vlastito eksperimentiranje i podešavanje parametara za optimalni rad. U literaturi [15] dva se skupa parametara smatraju dobrima za rješavanje. Prikazani su u tablici 2. Prvi se skupa parametara naziva mala populacija, a drugi skup velika populacija.

parametri	"mala" populacija	"velika" populacija
veličina populacije	30	100
vjerojatnost mutacije	1%	0.1%
broj jedinki za eliminaciju	veličina populacije/2	veličina populacije/4

Tablica 2 - Vrijednosti parametara koje se smatraju dobrima za rad genetskog algoritma

Parametri genetskog algoritma ne moraju biti konstantni tokom cijelog procesa izvođenja [15]. Dinamički mogu biti parametar broja iteracija (još se naziva funkcija vremena) te funkcija raspršenosti rješenja. Ako su rješenja previše raspršena treba povećati broj jedinki za eliminaciju (time se povećava broj križanja u svakoj generaciji jer se stvara onoliko novih jedinki koliko ih je uklonjeno u svakoj iteraciji). Također se smanjuje mutacija. Ako se dogodi bliskost rješenja potrebno je povećati mutaciju a smanjiti križanje. Križanjem dva potpuno jednaka kromosoma dobiva se treći potpuno jednak kromosom, a to nema smisla.

U radu je potrebno izbalansirati parametre križanja i mutacije. Ukoliko se koristi samo križanje algoritam ostaje zarobljen u lokalnom ekstremu. Ukoliko se koristi samo mutacija, algoritam će se nakon dovoljnog broja iteracija pronaći u blizini globalnog optimuma, ali uz nedovoljnu vrijednost parametra križanja neće moći konvergirati prema njemu.

4.3 Prikaz rješenja

Prikazi rješenja se dijele na izravne i neizravne. Izravni su oni prikazi (reprezentacije) gdje kromosom predstavlja raspored i genetski operatori djeluju na samom rasporedu. Neizravni su oni gdje jedinka ne predstavlja izravno raspored već predstavljaju slijed odluka kako izraditi raspored. U tom se slučaju operatori upotrebljavaju da bi se popravio redoslijed prioriteta unutar rasporeda [11].

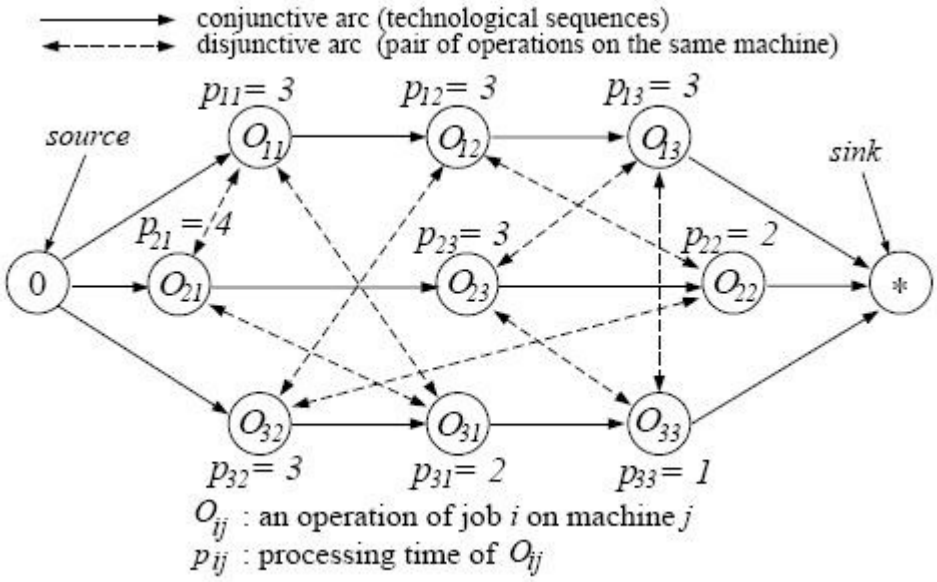
4.3.1 Usmjereni graf

Popularni je način prikaza u rješavanju problema proizvoljne obrade koji pripada u grupu neizravne reprezentacije rješenja. Usmjereni graf će biti prikazan na primjeru 3x3 koji je zadan na slici 10. Primjer je preuzet iz literature. [1]

job	Operations routing (processing time)		
1	1 (3)	2 (3)	3 (3)
2	1 (2)	3 (3)	2 (4)
3	2 (3)	1 (2)	3 (1)

Slika 10 – Jednostavni primjer s tri stroja i tri posla. Zadana su pravila prednosti i trajanja operacija. Primjerice prvi posao ide prvo na prvi stroj i to tri vremenske jedinice, zatim ide na drugi stroj tri vremenske jedinice itd.

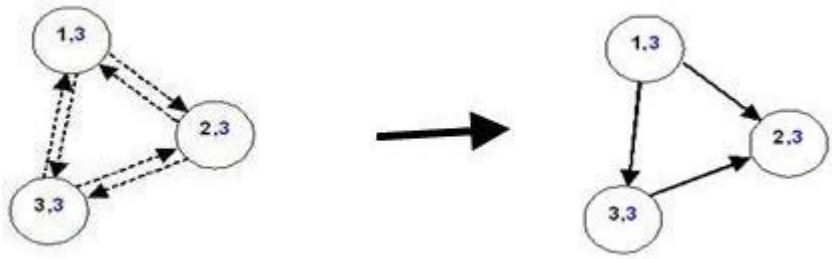
Problem proizvoljne obrade može biti prikazan kao disjunktivni graf. Čvorovi grafa predstavljaju pojedine zadatke (operacije). Svakom čvoru je pridružena težina i to u iznosu trajanja pojedine operacije. Definiraju se još dva čvora koji ne predstavljaju operacije, a zovu se izvor (označen na slici 11 s 0) i ponor (označen s * na istoj slici).



Slika 11 – Disjunktivni graf koji predstavlja problem proizvoljne obrade

Na grafu koji je prikazan na slici 11 zadana su dva skupa bridova. Prvi skup se naziva konjunktivni (eng. *conjunctive arcs*) i predstavlja tehnološki redoslijed operacija. To je zadano kao pravilo prednosti. Ti su bridovi imaju usmjerenje od operacije koja prethodi prema onoj koja je slijedi. Oni su postavljeni horizontalno na slici 11 tako da su strjelice bridova od lijeva prema desno. Drugi skup bridova se naziva disjunktivni (eng. *disjunctive arcs*). Tim su bridovima povezani parovi operacija koji se obavljaju na istome stroju. Na slici 11 su prikazani isprekidanim linijama i imaju dva usmjerenja.

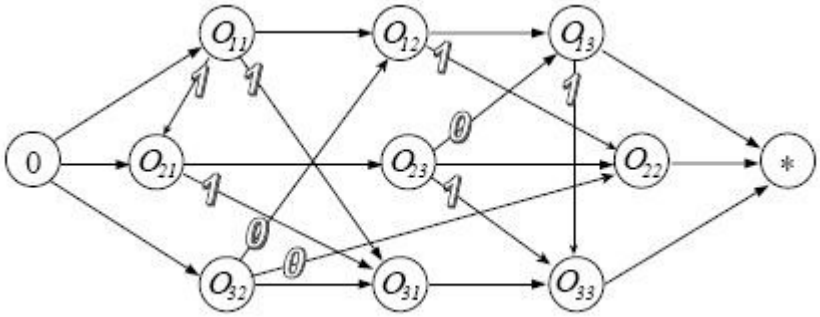
Zadatak je optimizacije pronaći usmjerenje disjunktivnim bridovima tako ne postoji ciklus u cijelom grafu. To je prikazano na slici 12.



Slika 12- Usmjeravanje bridova unutar podgrafa koji pripada jednom stroju

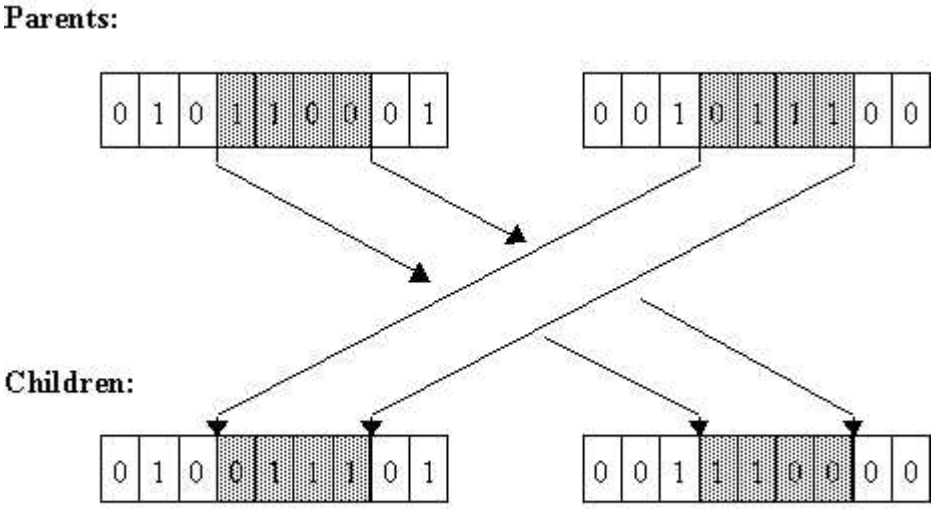
Ukoliko postoji ciklus takav raspored se ne može realizirati zbog neodređenosti redoslijeda operacija na tom stroju. Nakon što su svi bridovi usmjereni tako da nigdje nema ciklusa, dobiveno je poluaktivno rješenje. Najduži put od izvora do ponora naziva se kritični put. Zbroj trajanja operacija koje su na kritičnom putu je ukupno trajanje cijelog rasporeda (*makespan*).

Ovakva reprezentacija rješenja može se prikazati u računalu dosta jednostavno i to kao niz bitova. Ukupna duljina niza je $nm(n-1)$. Označavanje bridova se provodi po principu ako je operacija o_{ij} (gdje je i redni broj posla kojem pripada operacija, a j redni broj stroja na kojem se obavlja operacija) raspoređena prije operacije o_{kj} i vrijedi ($i < k$) tada je bridu pridružena jedinica, a u suprotnom slučaju pridružena mu je nula. Označavanje je prikazano na slici 13.



Slika 13- Označavanje bridova u grafu.

Pozitivno je kod ove reprezentacije rješenja što se mogu koristiti jednostavni operatori križanja kao što su križanje s jednom točkom prekida, križanje sa dvije točke prekida (prikazano na slici 14) i uniformno križanje. Kod tih križanja se jednostavno dio roditeljskog kromosoma kopira u svako dijete. Kod jedne točke prekida dio kromosoma (često polovina) se nasljeđuje od jednog roditelja, a polovina od drugog. Kod dvije točke prekida jedan dio kromosoma (zasjenjeni dio na slici 14) se kopira od jednog roditelja, a preostala dva dijela od drugog roditelja. Uniformno križanje je križanje sa $b-1$ točaka prekida. b je broj bitova u kromosomu. Tada se svaki pojedini gen kopira od drugog roditelja.



Slika 14- Križanje sa dvije točke prekida kakvo se može koristiti kod prikaza rješenja binarnim nizom

Negativno je kod ovih operatora što rezultat križanja može predstavljati nepostojeće rješenje. To se događa ukoliko dobiveni niz bitova predstavlja graf koji sadrži ciklus. Takav se raspored ne može realizirati i dobiveni potomak se naziva ilegalnim. Potreban je oporavak od pogreške [1]. Procedura oporavka se naziva harmonizacijski algoritam (eng. *harmonization algorithm*). Postupkom se stvara niz koji je što je moguće više sličniji ilegalcu, a da bude izvediv. Algoritam ima dvije faze: lokalnu harmonizaciju i globalnu harmonizaciju. Lokalna uklanja apsurdnosti unutar jednog stroja, dok globalna odstranjuje besmislenosti između strojeva. Originalni i ilegalni niz može se tretirati kao genotip, a popravljeni i izvedivi redoslijed kao fenotip i biti upotrebljen samo za evaluaciju funkcije cilja. To je jedan način, dok je drugi zamjena ilegalnog niza sa popravljenim. Ovaj se postupak zove prisiljavanje (eng. *forcing*) i može se se promatrati kao stečeni karakter, iako nije široko prihvaćeno da se takvo nasljeđivanje događa u prirodi. Kako učestalo prisiljavanje može uništiti raznolikost populacije ono je ograničeno na male elitne skupine. Takvo ograničeno prisiljavanje ima barem dvije vrline: značajan napredak u brzini konvergencije te kvaliteti rješenja. [1]

Ove poteškoće oko mogućnosti neizvedivih potomaka križanja imaju dodatne zahtjeve na implementaciju te je zbog te dodatne složenosti postupka odlučeno je da se reprezentacija rješenja kao binarni niz i graf neće koristiti u programskom ostvarenju koje prati ovaj diplomski rad (iako je prilično popularan u literaturi).

4.3.2 Permutacija s ponavljanjem

Problem proizvoljne obrade se može promatrati kao problem redoslijeda, nešto kao problem trgovačkog putnika [1]. Raspored se može predočiti kao skup permutacija poslova na svakom stroju. Takav prikaz se zove matrica redoslijeda poslova (eng. *job sequence matrix*). Na slici 15 je prikazan jedan mogući raspored za problem 3x3.

M_1	M_2	M_3
1 2 3	3 1 2	2 1 3

Slika 15 - Raspored operacija prikazan kao skup permutacija

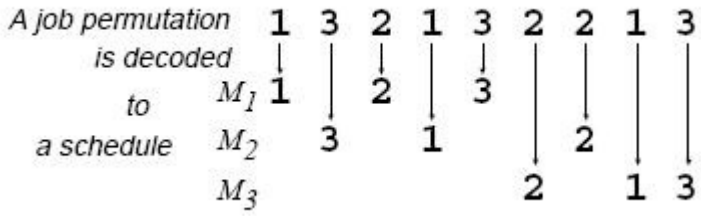
Osim ovog prikaza preko permutacija poslova još je jedan prikaz, također preko permutacije, moguć. Zove se permutacija s ponavljanjem. Svaki posao je predstavljen rednim brojem koji počinju od 1 pa do ukupnog broja poslova. Svaki posao se pojavljuje (ponavlja) onoliko puta koliko ima strojeva jer na svakom stroju ima po jednu operaciju. Tako dolazimo do ukupne duljine niza od *broj_poslova* x(puta) *broj_strojeva*. Da bi se prikazalo kako se permutacija pretvara u raspored promatramo jednostavni 3x3 problem (tri stroja i tri posla) na slici 16.

job	Operations routing (processing time)		
1	1 (3)	2 (3)	3 (3)
2	1 (2)	3 (3)	2 (4)
3	2 (3)	1 (2)	3 (1)

Slika 16- Jednostavni 3x3 problem za ilustraciju prikaza permutacije s ponavljanjem

Dakle svaki posao ima jedan redak u matrici. U svakom retku se nalaze strojevi koje posao obilazi. U zagradama je vrijeme trajanja obrade zadatka na pojedinom stroju. Ako promotrimo prvi redak vidimo da prvi posao prvo ide na stroj broj 1 tri vremenske jedinice, zatim na stroj 2 također tri vremenske jedinice, da bi završio na stroju 3 na kojem će obrada trajati 3 vremenske jedinice. Isti postupak je za posao broj 2 u drugom retku te za posao broj 3 u trećem retku.

Promotrimo primjer u kojem je slučajnim odabirom generiran raspored *132132231*. Ukupna duljina niza je 9 jer je broj strojeva 3 i broj poslova također 3. Prvi član je *1* što znači da je prva na redu prva operacija posla broj 1. Drugo pojavljivanje broja 1 u nizu znači da je na redu druga operacija prvog posla. To se lako pretvara u raspored poslova po strojevima kako je prikazano na slici 17.



Slika 17- Pretvaranje reprezentacije u raspored

Kako vidimo na slici 17 prvi posao ide prvo na prvi stroj. Treći posao ide prvo na drugi stroj pa je na mu je na Slici 3 pridružen stroj broj 2. Drugo pojavljivanje prvog posla na četvrtom mjestu u nizu stavlja taj posao na drugi stroj. Dok pak treća pojava broja 1 u reprezentaciji na osmom mjestu postavlja taj posao na treći stroj. Konačni je raspored poslova po strojevima STROJ BR.1 {1,2,3}, STROJ BR.2 {3,1,2} te STROJ BR.3 {2,1,3}.

Prednost je ove reprezentacije što se svaki raspored koji se generira može prevesti u ispravan (legalan) raspored. Nije potrebno popravljanje jer nije prekršeno pravilo prednosti. Mana je pak permutacije s ponavljanjem u tome što više različitih reprezentacija može biti prevedeno u isti raspored. [1]

Ukupni broj različitih permutacija s ponavljanjem je dan izrazom u jednadžbi 2. n je broj poslova a m je broj strojeva.

$$Number = \frac{(nm)!}{n * m!} \tag{4}$$

Vrline ovog prikaza rješenja koje ne trebaju dodatne opravke od pogrešaka pri križanju su bile motivacija da se baš ova reprezentacija koristi u programskoj implementaciji ovog diplomskog rada.

4.3.3 Kromosomi odluke

Ovakav prikaz rješenja pripada neizravnim reprezentacijama. Radi se o upravljanju odlukama o dodjeljivanju strojeva poslovima. Svaki kromosom ima duljinu od $n * m$ brojeva, gdje je n ukupan broj poslova, a m ukupan broj strojeva. Pojedini broj je u rasponu od nula do jedan i predstavljen je u implementaciji `double` tipom podataka u programskom jeziku C#. Brojevi se generiraju slučajnim odabirom. Tom se vrijednošću odlučuje koji će od raspoloživih poslova dobiti pravo na korištenje stroja. Primjerice ako se dva posla mogu izvoditi u datom trenutku na istom stroju i ako je vrijednost broja manja od 0.5 izvodi se posao sa manjim rednim brojem. Ukoliko je pak broj veći od 0.5, korištenje stroja dobiva posao sa većim rednim brojem. Slika 18 prikazuje primjere odlučivanja preuzete iz literature. [17]

Job	Tasks		
0	1	0	2
1	**0**	1	2
2	2	1	0
3	1	2	0

Job	Tasks		
0	**1**	0	2
1	**1**	2	
2	2	1	0
3	**1**	2	0

Job	Tasks		
0	1	0	2
1	1	2	
2	**2**	1	0
3	**2**	0	

Slika 18-Primjeri odlučivanja prava o korištenju stroja.

U prvom stupcu svake od tri tablica na slici 18 se nalazi popis poslova počevši od onoga sa rednim brojem 0 i zaključno s onim koji ima redni broj 3. U svakom retku koji pripada pojedinom poslu nalazi se raspored njegovih operacija koje se prikazane rednim brojem stroja na kojem se izvode. Drugi stupac tablice predstavlja operacije koje se mogu početi obrađivati u datom trenutku. Tako ako se promatra drugi stupac prve tablice vidi se da stroj 0 ima samo jedan raspoloživi posao i to je posao 1. Tako da bez obzira na vrijednost gena u kromosomu posao 1 dobiva stroj 0. Zatim se ta operacija briše iz tablice, a operacije prvog posla koje su iza nje pomiču se svaka za jedno mjesto ulijevo. Time se dobiva situacija koja je prikazana drugoj tablici na istoj slici. Sada ako se promatra stroj broj 1, on ima na raspolaganju tri posla i to posao 0, posao 1 i posao 3. Linearnom transformacijom se sustav odlučivanja preslikava na interval [0,1]. Tada da vrijednost gena u intervalu $[0,0.33]$ znači da posao 0 dobiva stroj 1, raspon gena od $\langle 0.33, 0.66]$ odgovara poslu 1, te konačno broj u intervalu od $\langle 0.66, 1]$ daje stroj 1 na korištenje poslu 3.

Potrebno je još i napomenuti kako se određuje za koji stroj se odnosi odluka. U ovoj implementaciji se to određuje slučajni odabirom posredstvom generatora pseudoslučajnih brojeva. Ukoliko za generirani stroj nije dostupan niti jedan posao, onda se ispitivanje prebacuje na stroj čiji je redni broj za jedan veći od ovoga.

Ukoliko se radi o dinamičkom problemu, onda nisu svi poslovi na raspolaganju u trenutku *t=0.* Za ovakvu vrstu problema ne koristi se permutacija s ponavljanjem već samo kromosomi odluke kao prikaz rješenja. Pretpostavka je bila da bi se slučajnim generiranjem permutacije s ponavljanjem moglo dogoditi da posao koji zadnji dolazi u sustav bude izabran kao prvi za raspoređivanje. To bi uzrokovalo iznimno loši raspored. Zbog toga se koriste kromosomi odluke i to na takav način da se promatra trenutno vrijeme u kojem se sustav nalazi. Oni poslovi čiji je početak vremenski iza aktualnog trenutka čekaju s raspoređivanjem.

Mana ovakvog pristupa je prerana konvergencija i zarobljavanje rješenja u lokalnom minimumu. [17] Pozitivno je što se mogu primijeniti spomenuti jednostavni operatori križanja jer se radi o brojčanom prikazu. Nema mogućnosti dobivanja nepostojećih rješenja prilikom rada operatora križanja i mutacije pa oporavak od pogreške nije potreban. Ova reprezentacija rješenja se može prevesti u permutaciju s ponavljanjem i takav se pristup koristi u implementaciji koja prati ovaj diplomski rad. Dakle koriste se dva prikaza rješenja: permutacija s ponavljanjem i kromosomi odluke.

4.4 Genetski operatori

4.4.1 Evaluacija

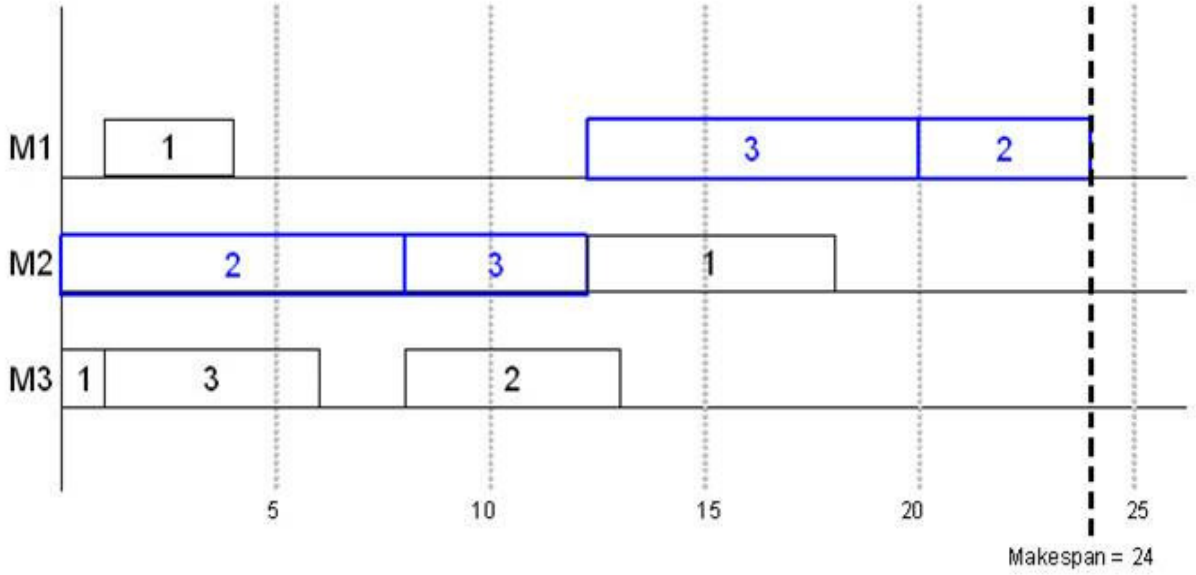
Za ocjenu kvalitete jedinke koriste se funkcije dobrote ili cilja (eng. *fitness function*). Bolja, u ovom slučaju niža vrijednost funkcije cilja znači i veću vjerojatnost prolaska kroz selekciju što omogućuje križanje u kasnijoj fazi i prenošenje genetskog materijala na sljedeću generaciju. Funkcija dobrote treba vjerno prikazivati problem koji se optimira. Od skupa funkcija cilja koje su bile opisane u poglavlju 2.4 za implementaciju u ovom diplomskom radu izabrane su minimizacija ukupnog trajanja dok se završe svi poslovi te također minimizacija kašnjenja poslova.

4.4.1.1 Minimizacija ukupnog trajanja

Traženje rasporeda s najkraćim ukupnim vremenskim trajanjem (eng. *makespan*) je najraširenija funkcija cilja kod problema proizvoljne obrade. Za prikaz rješenja koristi se Ganttov dijagram. Prilikom stavljanja operacije u raspored moraju se poštivati dva uvjeta:

1) Mora se uvažiti pravilo prednosti. Trenutak početka promatrane operacije mora biti veći ili jednak trenutku završetka operacije koja je neposredno ispred nje u pravilu prednosti.
2) Ukoliko su neke druge operacije drugih poslova koje se izvršavaju na istom stroju postavljene prije operacije koja se raspoređuje tada promatrani zadatak mora čekati da se završe zadatci drugih poslova koji su dobili prioritet. Zadatak može početi s obradom onog trenutka kad se svi drugi zadatci koji su već dobili taj stroj završe.

Budući da se moraju promatrati dva trenutka i oba pravila moraju biti zadovoljena kao početak operacije uzima se trenutak koji ima veću vrijednost odnosno koji je vremenski kasniji. Svaki stroj ima svoje vrijeme završetka svih svojih operacija, a uzima se najveće od tih vremena.



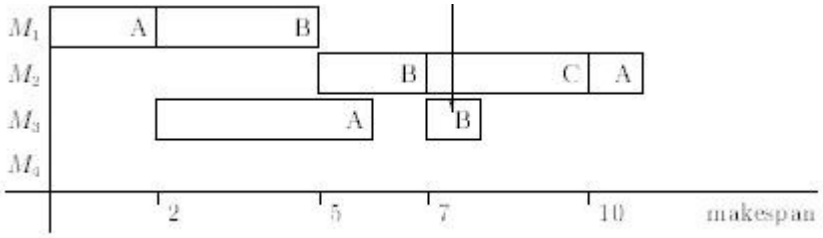
Slika 19 – Prikazan je raspored za 3x3 problem. U retcima su strojevi. Pravokutnici predstavljaju zadatke pojedinih poslova. Njihova širina ovisi o trajanju vremena izvođenja. Primjerice treća operacija trećeg posla koja se obavlja na prvom stroju mora čekati da se operacija istog posla koja se obavlja na drugom stroju jer je postavljena prije nje u pravilu prednosti.

Na slici 19 plavom bojom su predstavljene operacije koje se nalaze na kritičnom putu. Ukoliko se koristi prikaz rješenja preko grafa koji je opisan u poglavlju 4.3.1 tada zbroj trajanja svih operacija koje se nalaze na kritičnom putu predstavlja i ukupno trajanje cijelog rasporeda. Kritični put je onaj put od izvora do ponora grafa koji je najdulji.

Poboljšanje rasporeda pomicanjem operacija ulijevo

Uvažavanjem ova dva uvjeta kako je prethodno opisano dobiva se poluaktivni raspored. Njegova je karakteristika da se ne može poboljšati bez mijenjanja rasporeda. Kako je opisano u poglavlju 2.5, postoji još jedan manji podskup skupa poluaktivnih rješenja u kojem se sigurno nalazi optimalno rješenje. To je skup aktivnih rješenja koji se ne može dalje poboljšati bez da se prekrši pravilo prednosti.

Poluaktivni raspored se može unaprijediti u aktivni pomicanjem operacija ulijevo na Ganttovom dijagramu. Postupak je objašnjen u literaturi [19] a bit će prikazan na primjeru.



Slika 20 – Prikazan je poluaktivni raspored za 3x3 problem. Postoji razlika u prikazu jer poslovi nisu predstavljeni brojevima nego slovima.

Raspored na slici 20 je dobiven postupkom koji je opisan ranije, i u kojem su poštivana dva uvjeta:

1) Uvaženo je pravilo prednosti.
2) Operacije drugih poslova koje su ranije raspoređene nego operacija koja se upravo raspoređuje, a obavljaju se na istom stroju imaju prednost u obavljanju. Time bi trenutni zadatak morao čekati da se obrade svi zadatci koji su postavljeni prije njega da bi on konačno došao na red.

Tim dvama uvjetima dobiven je poluaktivni raspored. Taj se raspored može unaprijediti u aktivni raspored. Pri tome se prvi uvjet mora poštivati, ali drugi ne mora. Ako se promotri slika 20, i gleda se operacija posla C koja se obavlja na M_2 stroju vidi se da je ona postavljena iza operacije posla B i tako bi mogla početi tek u trenutku 7. Ali raspored se može poboljšati pomicanjem operacije posla C ulijevo, tako da ona bude raspoređena prije operacije posla B, da počne prije nje i završi prije nje. Razlog je tome što postoji vrijeme prije operacije posla B (od trenutka 0 do trenutka 5) kada je stroj M_2 neaktivan. Prvi uvjet je zadovoljen jer je to prva operacija posla C i ne mora čekati nijednu operaciju istog posla. Operacija posla C se pomiče ulijevo i počinje u trenutku 0 i traje do trenutka 3.

Pomicanje operacija ulijevo se obavlja prilikom evaluacije. Tada se definira kada koja operacija počinje i završava i može se detektirati da li je moguće pomaknuti pojedini zadatak ulijevo.

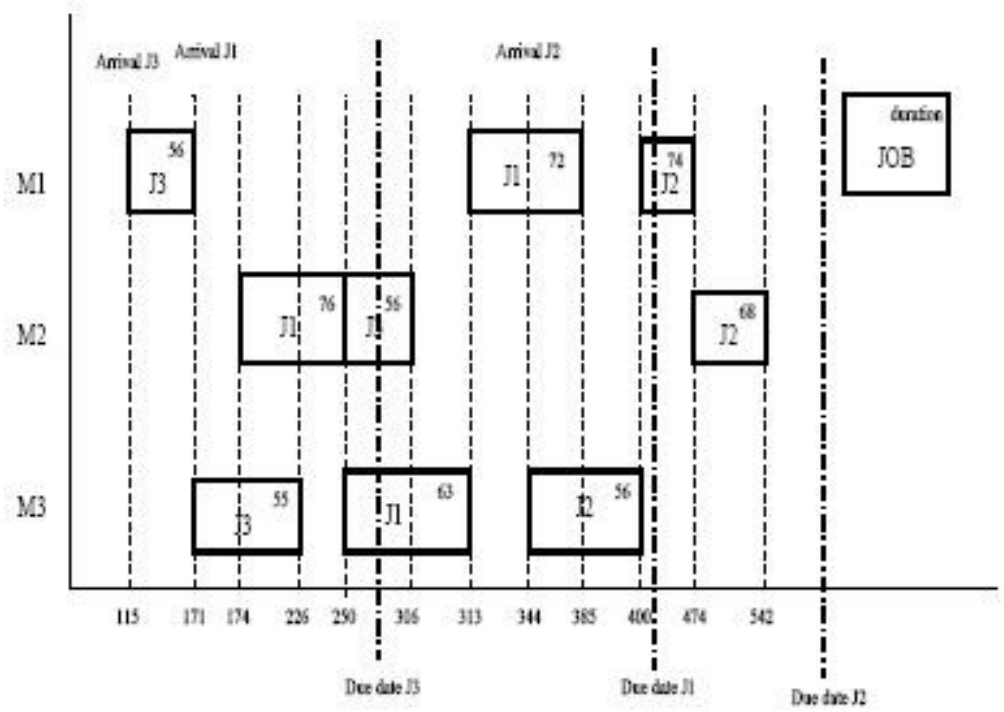
Potrebno je ažurirati reprezentaciju. To je prikazano na slici 21.

Slika 21 – Operacija posla C se pomiče ulijevo tako da se nalazi ispred prve operacije koja joj slijedi u novom rasporedu. To je u ovom primjeru druga operacija posla B.

4.4.1.2 Ukupno kašnjenje svih poslova

Kod dinamičkog problema proizvoljne obrade koji se također obrađuje u ovaj implementaciji ponekad ukupno trajanje nije najbolji pokazatelj kvalitete rješenja. Kod dinamičkog problema ne dolaze svi poslovi u sustav u trenutku t=0, već imaju vremena dolaska u sustav koja su veća od nule. Također često za takve slučajeve poslovi imaju definirane rokove, odnosno vremenske trenutke do kada bi trebali ili morali biti završeni. Ovo se definira često i za statičke probleme.

Jedan od načina kako promatrati kvalitetu rješenja jest definirati termin kašnjenja L_j (eng. *lateness*) gdje je j redni broj posla. Ta vrijednost se određuje za svaki posao i računa se oduzimanjem stvarnog završetka posla sa onim koje je bilo predviđeno. Kada se odrede te vrijednosti za svaki pojedini posao, ukupno kašnjenje se dobije njihovim zbrajanjem, $\sum_{j=0}^{n} L_j$. Najboljim rasporedom se smatra onaj koji ima najmanje ukupno kašnjenje. To je prikazano na slici 22 za jednostavni 3x3 problem.



Slika 22 - Ganttov dijagram za dinamički 3x3 problem sa zadanim vremenima dolaska i završetka poslova. Isprekidanim crta-točka-crta linijama su prikazani rokovi.

4.4.2 Selekcija

Selekcija predstavlja komponentu evolucije kojom se prirodnim odabirom biraju bolje jedinke koje će preživjeti i prenijeti svoje gene na sljedeću generaciju.

Jedna od mogućnosti selekcije je sortirati jedinke u generaciji prema vrijednosti funkcije cilja, i odabrati određeni broj najboljih za reprodukciju te odstraniti preostale. Zamka je ovakvog pristupa što se gubi dobar genetski materijal kojeg sadržavaju loše jedinke, a sve završava preranom konvergencijom prema lokalnom optimumu i zarobljavanje u istome nakon svega nekoliko iteracija [15].

Selekcije mogu biti generacijske i eliminacijske i sukladno tome genetski algoritmi se dijele na generacijske i eliminacijske [18].

1) Generacijske selekcije odabiru iz trenutne generacije najbolje jedinke i kopiranjem istih stvaraju novu generaciju. Nedostatak je što postoji više

potpuno jednakih jedinki u sljedećoj generaciji što smanjuje raznolikost i kvalitetu cjelokupnog genetskog koda.

2) Eliminacijske selekcije ne biraju koje će jedinke preživjeti odabir nego koje će jedinke biti odstranjene. Tu lošije jedinke imaju manju vjerojatnost preživljavanja. Njihova mjesta se popunjavaju proizvodnjom novih jedinki koje se dobivaju križanjem boljih kromosoma.

Postoje razni mehanizmi selekcije.Neki od čestih primjera su: jednostavna, turnirska, eliminacijska i elitizam [15].

1) Jednostavna selekcija (eng. *roulette wheel parent selection*)) se koristi u generacijskom genetskom algoritmu. Jedinke koje će sudjelovati u reprodukciji odabiru se sa vjerojatnošću koja je proporcionalna njihovoj kvaliteti.
2) Turnirska selekcija slučajnim odabirom bira *k* jedinki iz populacije. Ako se radi se o generacijskoj selekciji bolje jedinke preživljavaju odabir, a radi se pak eliminacijskom odabiru onda najslabija jedinka nestaje. Budući da je turnirska selekcija jedina selekcija koja se koristila u programskoj implementaciji vezanoj za ovaj diplomski rad, bit će podrobnije objašnjena kasnije.
3) Eliminacijska selekcija je naprednija verzija prethodno spomenute jednostavne selekcije. Razlika je što se ne biraju kromosomi koje će preživjeti, nego oni koji će biti eliminirani. Sukladno tome, definira se funkcija kazne tako da lošije jedinke sa većom vrijednošću tog izraza imaju veću vjerojatnost za uklanjanje iz procesa.
4) Elitizam se uvodi kod onih selekcija kod kojih nije sigurno da će najbolja jedinka preživjeti odabir. Primjeri takvih su jednostavna i eliminacijska selekcija gdje doduše najkvalitetniji kromosom ima najveću vjerojatnost prelaska u sljedeću generaciju, ali ona nije 100%-tna. Da se ne bi izgubilo takvo dragocjeno rješenje, može se primijeniti elitizam koji u svakoj generaciji traži najbolju jedinku i automatski je kopira u sljedeću iteraciju. Ovim postupkom genetski algoritam se približava globalnom ekstremu, ali povećava se računalna složenost postupka. Kod turnirske selekcije koja se koristi u ovom diplomskom radu elitizam nije potreban jer najbolja jedinka uvijek prolazi selekciju.

4.4.2.1 Turnirska selekcija

Kod genetskih algoritama sa složenijim prikazom rješenja, kakav je i permutacija s ponavljanjem te kromosomi odluke, jednostavna selekcija bi bila računski prezahtjevna. Zbog toga se rodila ideja o turnirskoj selekciji jer u njoj ne sudjeluju sve jedinke što podrazumijeva manji broj kopiranja i sortiranja elemenata [16].

U postupku se slučajnim odabirom bira *k* jedinki koje će sudjelovati u turniru. Ovisno o broju *k* selekcija se zove k-turnirska selekcija. Ukoliko se radi o generacijskoj turnirskoj selekciji u turniru se traži najbolji kromosom i on se kopira u sljedeću generaciju. Postupak se ponavlja onoliko puta koliki je broj jedinki u populaciji (veličina populacije). Ako se radi o eliminacijskoj turnirskoj selekciji, onda se pak bira najlošija jedinka koja se odstranjuje. Na njezino mjesto dolazi nova jedinka koja će nastati kao produkt križanja između druge dvije jedinke koje su također sudjelovale u turniru. One se biraju slučajnim odabirom.

U implementaciji koja prati ovaj diplomski rad korištena je 3-turnirska eliminacijska selekcija. Od tri jedinke koje su odabrane najlošija otpada, a preostale dvije križanjem proizvode novu jedinku.

4.4.3 Križanje

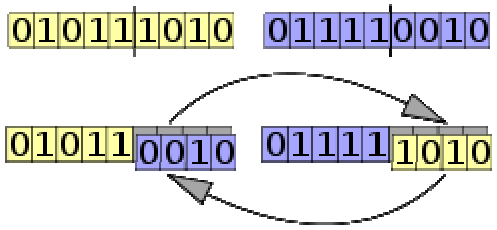
Operator križanja se smatra okosnicom genetskog algoritma [19]. Njime se određuje kako se prenosi genetski kod sa roditelja na djecu. U njemu sudjeluju dva roditelja koji oboje prenose svoje karakteristike na potomke. Potomci bi vjerojatno trebali biti jednako dobri kao i roditelji, ako ne i bolji.

U implementaciji su korišteni operatori jednostavnog križanja od kojih je odabrano uniformno križanje, te operatori koji rade na permutaciji s ponavljanjem kao prikazom rješenja, a zovu se poopćeno redoslijedno križanje te križanje s očuvanjem pravila prednosti.

4.4.3.1 Jednostavni operatori – uniformno križanje

Oni se mogu koristiti ukoliko je kromosom niz bitova ili kao u slučaju prikaza kromosoma odluke, niz brojeva sa vrijednostima između 0 i 1. Najčešći jednostavni operatori križanja su križanje s jednom točkom prekida, križanje s dvije točke prekida i uniformno križanje [15].

1) Kod križanja s jednom točkom prekida slučajnim odabirom se traži mjesto između dva gena gdje će biti točka prekida. Dio kromosoma prije točke se nasljeđuje od jednog roditelja, a drugi dio od drugog.



Slika 23- Križanje s jednom točkom prekida

2) Križanje s dvije točke prekida je slično prethodnome s razlikom da postoje dvije točke prekida koje dijele kromosom na tri dijela. Svako dijete dobiva jedan dio od prvog roditelja, a dva od drugog ili obrnuto.
3) Uniformno križanje je križanje sa *b-1* točaka prekida (*b* je broj gena).Ukoliko se radi o binarnom prikazu kromosoma može ga se vrlo jednostavno i učinkovito implementirati kao logičku operaciju.

$$DIJETE = A * B + R(A \oplus B) \tag{3}$$

Jednadžba 3 prikazuje izraz koji predstavlja uniformno križanje. Prvi član $A * B$ osigurava prenošenje gena koji su jednaki u oba roditelja. R je slučajni niz bitova jednake duljine kao i kromosomi i njime se bira koji će se gen naslijediti od onih koji se razlikuju od oba roditelja. Ovakva implementacija preko logičkih operatora je moguća jedino ako su kromosomi binarni nizovi. Kod dvaju prikaza rješenja koje se koriste u ovom diplomskom radu to nije slučaj. Ipak se uniformno križanje koristi kod reprezentacije rješenja kao kromosoma odluke gdje se također generira slučajni vektor R koji se sastoji od nula i jedinica. Vrijednosti u tom vektoru odlučuju od kojeg će se roditelja naslijediti pojedini gen. Na slici 24 je prikazano kako je to implementirano na primjeru iz literature [5].

Coin toss	*H*	*H*	*T*	*H*	*T*
Parent 1	*0.57*	*0.93*	*0.36*	*0.12*	*0.78*
Parent 2	*0.46*	*0.35*	*0.59*	*0.89*	*0.23*
Offspring	*0.57*	*0.93*	*0.59*	*0.12*	*0.23*

Slika 24- Prvi redak zauzima niz vrijednosti koji predstavlja binarnu odluku koja može biti glava(eng.Head, H) ili pismo(eng.Tail,T). Ukoliko novčić padne na glavu odabire se gen prvog roditelja (kao što je u slučaju prvog gena, 0.57), a ukoliko padne na pismo bira se pripadajući broj iz drugog roditelja.

Križanja s jednom i dvije prekidne točke se koriste pri velikim populacijama kad je prisutna veća raznolikost rješenja. Uniformno križanje je suprotnost koja se koristi pri malim populacijama [15].

4.4.3.2 Poopćeno redoslijedno križanje

Ovo križanje (eng. *Generalized Order Crossover, GOX*) je usko vezano za prikaz rješenja kao permutacije s ponavljanjem. Ova reprezentacija rješenja ima dodatne zahtjeve na operator križanja i mutacije ukoliko se žele izbjeći ilegalni potomci, a to je da produkti križanja moraju imati jednaki broj ponavljanja svakog posla (što mora biti *m* puta, gdje je *m* broj strojeva). Drugi je zahtjev dosljednost kopiranja svojstava s roditelja na djecu što se ne može napraviti samo kopiranjem gena već se treba voditi računa i koje je to ponavljanje gena u kromosomu, jer različita ponavljanja u poretku istog posla znače i različite operacije koje se obavljaju na drugim strojevima.

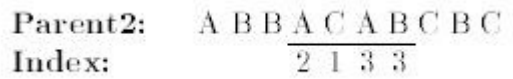
Stoga se u poopćenom redoslijednom križanju osim kromosoma koji ima permutaciju s ponavljanjem definira još jedan niz koji je dugačak *m x n* elemenata. Sadrži brojeve u rasponu od 1 do *n*. Svaka brojka kazuje koje je to po redu ponavljanje svakog posla. Tako da prva operacija po pravilu prednosti ima redni broj 1, druga 2, itd. To je prikazano na slici 25.

Parent1:	B	A	B	B	C	A	C	C	B	A
Index:	1	1	2	3	1	2	2	3	4	3

Slika 25- Slika pokazuje kromosom koji umjesto rednih brojeva poslova sadrži slova abecede tako da je prvi posao predstavljen slovom A, drugi sa B, itd. Ispod kromosoma je još jedan niz koji govori o kojoj se operaciji pojedinog posla radi. Tako da prva operacija ima indeks 1, druga 2, itd.

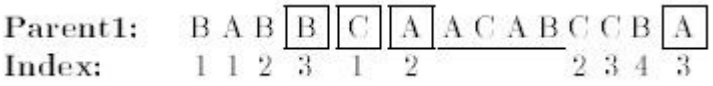
Jedan roditelj se ponaša kao primatelj, a drugi kao davatelj. U kromosomu davatelju slučajnim odabirom se određuje pozicija onih gena koji će biti darovani. Duljina tog niza se također određuje slučajno, i to između trećine i polovine duljine cjelokupnog kromosoma. Određeni niz za donaciju može cijeli ležati u jedinci neprekinut, ili može jedan dio biti na početku kromosoma, a drugi na kraju.

1) Ukoliko je cijeli niz unutar davateljskog kromosoma postupak je sljedeći.
   Prvo se izračunaju indeksi ponavljanja onih gena koji će biti donirani. To je prikazano na slici 26.



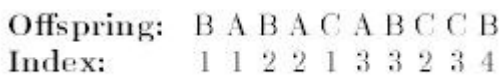
Slika 26- Izračunavaju se indeksi ponavljanja za one gene koji će biti donirani. To su geni počevši sa četvrtim (posao A, drugo ponavljanje) i završno sa sedmim(posao B, treće ponavljanje)

Sada se svi geni koji se pojavljuju u nizu u donatorskom kromosomu koji će se koristiti u križanju označuju u primateljskom kromosomu. Pozicija gdje će donirani niz biti postavljen u primateljski kromosom određuje se tako da se promatra koji je prvi gen u donatorskom nizu. Pronalazi se pozicija istog gena s istim indeksom ponavljanja u primateljskom kromosomu. Donatorski niz će biti stavljen neposredno iza toga gena. Oni geni koji su bili označeni u primateljskom kromosomu će biti izbrisani. To je prikazano na slici 27.



Slika 27- Donatorski niz ACAB je postavljen u primateljski kromosom tako da je postavljen iza istog gena s istim indeksom ponavljanja kao što je prvi gen u donatorskom kromosomu. Pripadni geni koji se pojavljuju u primateljskom kromosomu, a dio su donatorskog niza, označeni su i bit će obrisani.

Uklanjanjem gena koji se pojavljuju u primateljskom kromosomu, a dio su donatorskog niza, dobiva se gotovi potomak na slici 28.



Slika 28- Konačni izgled kromosoma potomka

Bitno je napomenuti da ovo križanje ima jednu manu. Ako se zamisli konfiguracija u kojoj se gen *C* u primateljskom kromosomu pojavljuje dva puta prije prekidanja kromosoma da bi se umetnuo donorski niz. Budući da donorski niz sadrži gen *C* s indeksom ponavljanja 1, jedan od ta dva gena koji

se pojavljuju prije točke prekida će biti izbrisan, ali jedan će ostati, što znači da će gen *C* u donorskom nizu zapravo dobiti indeks 2, jer će biti drugi takav gen u cijelom kromosomu. To podrazumijeva gubitak informacije jer se radi o drugoj operaciji što rezultira drugačijim rasporedom.

Navedeno se može izbjeći implicitnom mutacijom gdje će se sporne operacije zamijeniti. Izbor mjesta u kromosomu gdje će se rezati donorski niz također smanjuje vjerojatnost za ovakvim iznimkama.

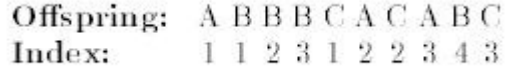
2) Druga je mogućnost da je donatorski niz zamotan oko krajnjih točaka donatorskog kromosoma kao na slici 29 .

Slika 29- Drugi slučaj kada donorski niz nije neprekinuti i cijeli unutra kromosoma nego se obavija oko rubova kromosoma

Sad je cijela procedura jednostavnija. Nakon označavanja gena koji su dvostruki, oba dijela donorskog niza se dodaju u primateljski kromosom na ista mjesta koja su zauzimala u donorskom kromosomu. Prikaz je na slici 30.

Slika 30- Dijelovi donorskog niza su dodani na početak i kraj primateljskom kromosoma. Dvostruki geni koji se pojavljuju u obje instance označeni su i bit će izbrisani.

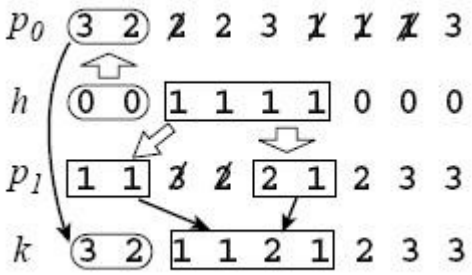
Konačni izgled kromosoma dan je na slici 31.



Slika 31- Završna reprezentacija kromosoma

4.4.3.3 Križanje s očuvanjem pravila prednosti

Ovaj operator (eng. *precedence preservative crossover - PPX*) je, kao i prethodni, usko vezan za prikaz rješenja kao permutacije s ponavljanjem. Prije križanja se generira niz *h* duljine *broj_strojeva* x *broj_poslova*. Niz se sastoji od nula i jedinica. Ako je generirana nula potomak nasljeđuje gen od prvog roditelja, a ako je generirana jedinica nasljeđuje taj gen od drugog roditelja. Jedan primjer križanja je prikazan na slici 32.



Slika 32- Križanje sa očuvanjem pravila prednosti(PPX)

Kao što se vidi na slici 32 u nizu *h* prvo su generirane dvije nule što znači da se prva dva gena nasljeđuju od prvog roditelja p_0. To su poslovi 3 i 2. Ti se poslovi uništavaju u drugom roditelju p_1. Zatim su u pomoćnom nizu *h* generirane 4 jedinice. To znači da se iduća 4 gena nasljeđuju od drugog roditelja. To su poslovi 1,1,2,1. Poslovi 3 i 2 na trećem i četvrtom mjestu u roditelju p_1 se ne nasljeđuju jer su već naslijeđeni u prethodnom koraku. Zbog istog razloga se uništavaju geni 2,1,1,1 u prvom roditelju. Konačno generirane su tri nule. Od prvog roditelja se nasljeđuju tri gena i to su jedina tri preostala gena koji nisu već naslijeđeni ili uništeni. To su geni 2,3 i 3. Završni izgled potomka *k* je {3,2,1,1,2,1,2,3,3}.

4.4.4 Mutacija

U prirodnom procesu evolucije uloga je mutacije dobivanje novog genetskog materijala i povećavanje raznolikosti. Većina mutacija su negativne za jedinke koje zbog toga ne preživljavaju prirodnu selekciju. Dobre mutacije pak donose bolju prilagodbu koja se nakon nekoliko generacija širi i postaje dominantna u vrsti.

Mutacija u genetskom algoritmu ima funkciju izbjegavanja lokalnih optimuma. Algoritam koji ima dovoljno visoku vjerojatnost mutacije i dovoljan broj iteracija pronaći će se u blizini globalnog optimuma. Ako vjerojatnost mutacije teži prema 100%, genetski algoritam postaje slučajno pretraživanje. Ako je spomenuti parametar blizak nuli, postupak će zaglaviti u nekom lokalnom optimumu [15].

Operator mutacije je unarni operator što znači da djeluje na jedan kromosom, za razliku od operatora križanja koji radi na dvije jedinke.

Radi se o slučajnoj promjeni u gena u kromosomu. Najjednostavnije ga je implementirati ukoliko se radi o binarnom prikazu kromosoma. Tada se samo jedan gen promijeni iz nule u jedinicu ili obrnuto.

Ponešto je drugačije kod permutacije s ponavljanjem i kromosomima odluke koji se koriste u ovom diplomskom radu. Tu se ne radi o promjeni samo jednog gena, nego dva ili višekratnika broja dva (4,6,itd.). Naime ukoliko se promjeni samo jedan gen na nekoj poziciji, primjerice iz posla rednog broja 3 u posao rednog broja 4, tada raspored više nije izvediv. Došlo je do viška operacija četvrtog posla, a manjka operacija trećeg posla. Iz tog razloga, traže se uvijek dva gena, i na njima se obavlja operacija zamjene. Ako se primjerice radi o reprezentaciji {3,2,1,1,2,1,2,3,3} i trebaju se zamijeniti treći i sedmi gen reprezentacija nakon mutacije izgleda: {3,2,2,1,2,1,1,3,3}. Koji će geni biti zamijenjeni odlučuje se slučajnim odabirom.

Potrebno je još i napomenuti kako se računa koliko će parova biti zamijenjeno u procesu mutacije. Parametar vjerojatnosti mutacije jednog gena se zadaje na početku programa. Tipično se kreće od 0.001 do 0.01. Zatim se određuje broj parova koji će biti mutirani. On se dobije kako je prikazano u jednadžbi 3

$$BrojParova = n * m * p_m/2 \quad (3)$$

n je broj poslova, *m* je broj strojeva, p_m je vjerojatnost mutacije jednog gena. Sve se dijeli sa faktorom 2 jer se traži broj parova gena koji će biti mutirani, a ne broj pojedinačnih gena.

Ukoliko se dobije broj koji nije cijeli postupak je sljedeći.

- Cijeli dio broja (primjerice 3, ukoliko je dobiven 3.33) predstavlja broj zamjena koje će sigurno biti obavljene
- Decimalni dio broja (0.33) predstavlja vjerojatnost za još jednu dodatnu promjenu. Programski se generira decimalni broj između nula i jedan. Ukoliko je manji od ostatka broja (0.33) tada se obavlja još jedna zamjena.

Mutacija se obavlja u algoritmu neposredno nakon operacije križanja.

5. Analiza rezultata

5.1 Opis načina pokretanja programa i zadavanja parametara

Programsko ostvarenje podržava statički i dinamički deterministički problem proizvoljne obrade. Metoda rješavanja je genetski algoritam.

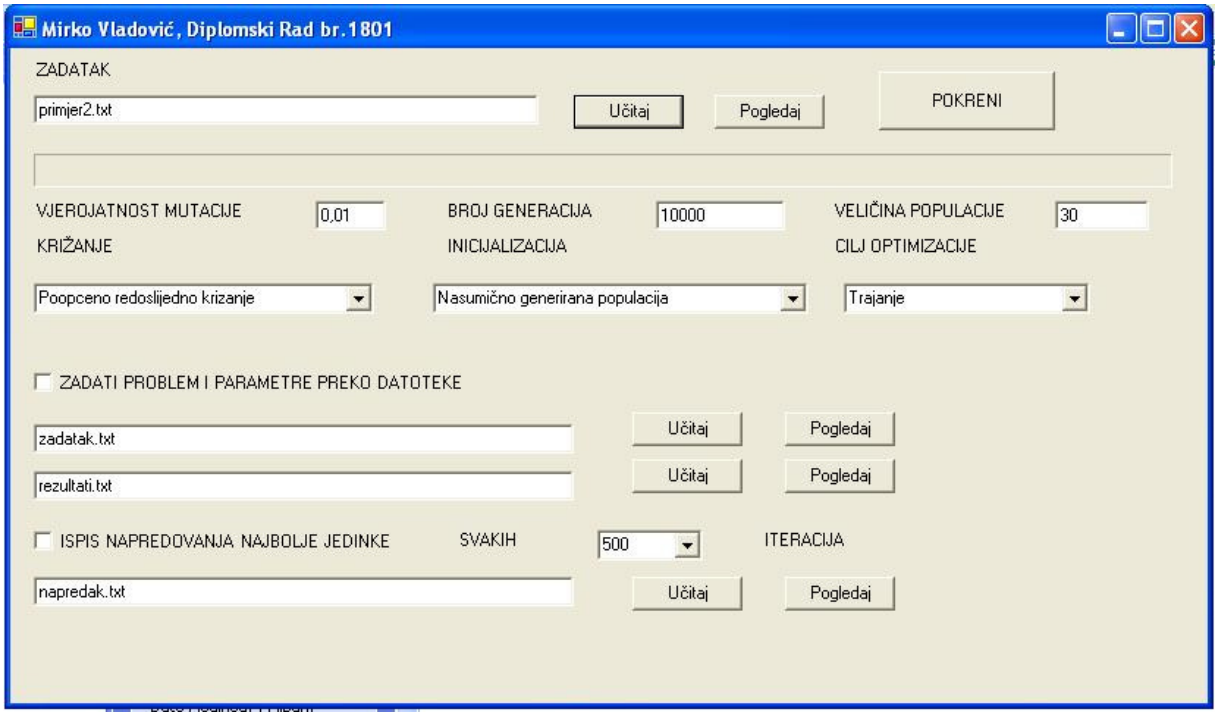
Koriste se dva prikaza rješenja: permutacija s ponavljanjem te kromosomi odluke. Podržani su tri operatora križanja:

- Poopćeno redoslijedno križanje
- Križanje s očuvanjem pravila prednosti
- Uniformno križanje

Uniformno križanje je podržano samo ako se koriste kromosomi odluke kao prikaz rješenja.

Koristi se 3-turnirska eliminacijska selekcija.

Rad je programiran u programskom jeziku C# i izvršnoj datoteci je za izvođenje potreban .NET Framework 1.1 ili viši. Pokretanje se obavlja dvoklikom na izvršnu datoteku *JobShop.exe.*

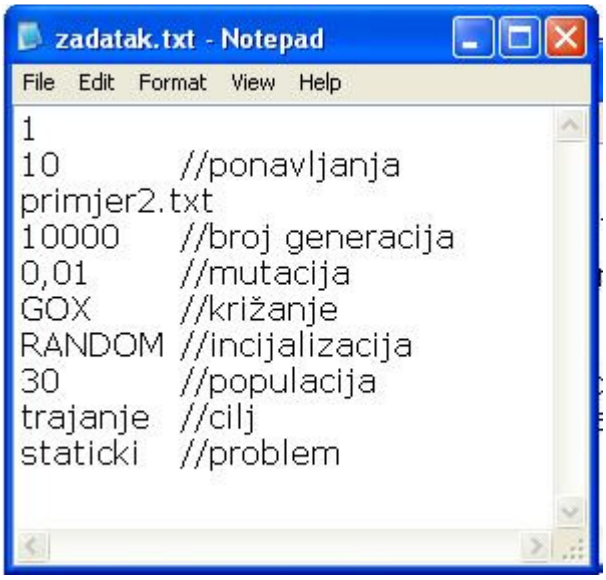


Slika 33-Prozor programa

Kao što je vidljivo na Slici 33 nakon pokretanja otvara se prozor sa programom. U prvom okviru za tekst (eng. *text box*) ispod oznake *ZADATAK* nalazi se ime datoteke gdje se nalazi zadan problem proizvoljne obrade problem u tekstualnom obliku. Pritiskom na tipku *Pogledaj* može se u tekstualnom pregledniku Notepad vidjeti kako izgleda zadani problem. Tipkom *Učitaj* koja se nalazi lijevo od tipke *Pogledaj* može se učitati neki drugi problem u okvir za tekst. Ispod toga se nalaze tri okvira za tekst i to su parametri koji se zadaju prije pokretanja programa. To su vjerojatnost mutacije koja je broj između 0 i 1. Inicijalno je zadana na vrijednost 0.01 ili 1%. Zatim je tu broj generacija koji je inicijalno zadan na 10 000 te veličina populacije što je broj jedinki koji se generira slučajnim odabirom i taj broj jedinki sudjeluje u evoluciji. Inicijalno je postavljen na 30. Korisnik programa može mijenjati parametre upisom drugih vrijednosti u okvire za tekst.

Ispod tih okvira postavljene su strukture za odabir operatora križanja, prikaza rješenja te cilja optimizacije. Bitno je napomenuti da program ima dva načina rada. Prvi je način prethodno opisan gdje se zadaju parametri preko okvira za tekst. U ovoj radnoj metodi algoritam se izvršava samo jednom. Ukoliko se želi statistički testirati program potrebno je više ponavljanja. Drugi je način pogodan za to. Aktivira se

ukoliko je potvrdni okvir (eng. *check box*) u prozoru kraj kojeg je oznaka *ZADATI PROBLEM I PARAMETRE PREKO DATOTEKE* potvrđen. Tada se problem, broj ponavljanja i parametri zadaju preko datoteke. Na slici 34 je dan izgled datoteke.

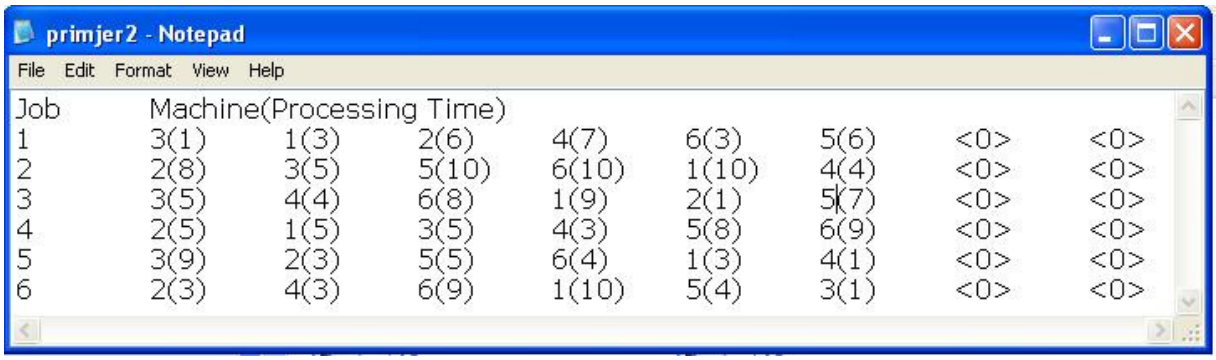


Slika 34- Problem, broj ponavljanja, parametri, prikaz rješenja, cilj optimizacije zadani preko datoteke

Ime datoteke se zadaje preko okvira za tekst koji se može uređivati preko tipke *Učitaj.* Također se na isti način zadaje i datoteka u koju će biti spremljeni rezultati.

Kvaliteta algoritma se može ispitivati praćenjem načina na koji se najbolja jedinka napreduje kroz iteracije. To se sprema u posebnu datoteku.

Potrebno je prikazati i kako treba izgledati datoteka u kojoj je zadan problem. To je pokazano na slici 35.



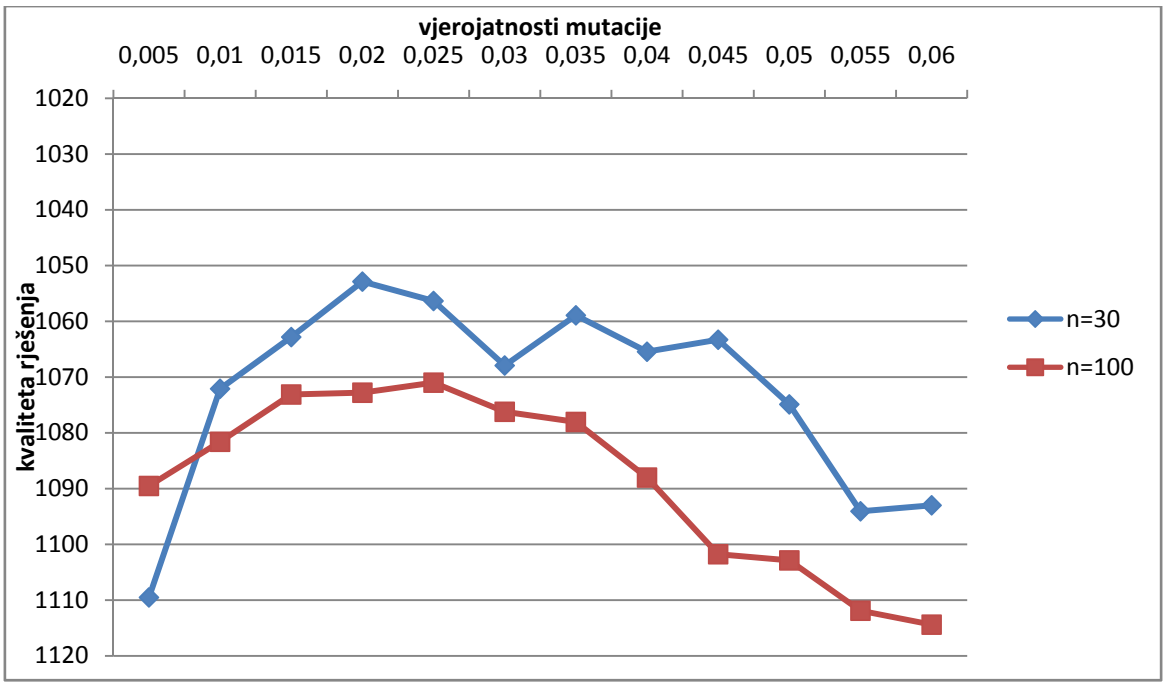
Slika 35- Datoteka sa 6x6 problemom

Na slici 35 je prikazan sadržaj tekstualne datoteke *primjer2.txt* gdje je zadan problem 6x6 (šest poslova na šest strojeva). U redcima su zadani poslovi i njihove operacije. Prva operacija prvog posla je na trećem stroju i traje jednu vremensku jedinicu. Zatim prvi posao ide na prvi stroj i to tri vremenske jedinice. Prvi redak u datoteci služi za opis stupaca datoteke. Prvi stupac su redni brojevi poslova (eng. *job*) dok su idućih šest redaka redni brojevi strojeva na kojima se operacije tog posla i u zagradama vremena trajanja obrade operacija na strojevima. U redoslijedu operacija zadano je pravilo prednosti. Primjerice, prvi posao mora ići prvo na stroj 3, pa na stroj 1 a nakon toga na stroj 2 i tako dalje. Stupci u datoteci moraju biti odvojeni tabovima. Zadnja dva stupca predstavljaju parametre dolaska posla u sustav (ta je vrijednost 0 ukoliko se radi o statičkom problemu, a veća od nule ukoliko se radi o dinamičkom problemu) te predviđeno vrijeme do kada bi se posao trebao završiti. Program se pokreće pritiskom na veliku tipku *Pokreni*. Kada je optimizacija gotova, korisnik je obaviješten preko poruke (eng. *Message Box*) s tekstom "ZAVRŠIO".

5.2 Osjetljivost kvalitete rješenja i vremena izvođenja o izboru parametara

5.2.1 Osjetljivost kvalitete rješenja o vjerojatnosti mutacije

Vjerojatnost mutacije je najvažniji parametar genetskog algoritma i kvaliteta rješenja je najosjetljivija na njegovu promjenu [16]. Vrijednost tog parametra je vezana uz količinu jedinki u populaciji. Kod većeg broja kromosoma (npr.100) dobra vrijednost vjerojatnosti mutacije se smatra 0,001, dok je kod manje populacije (npr.30) potrebna veća vrijednost mutacije.



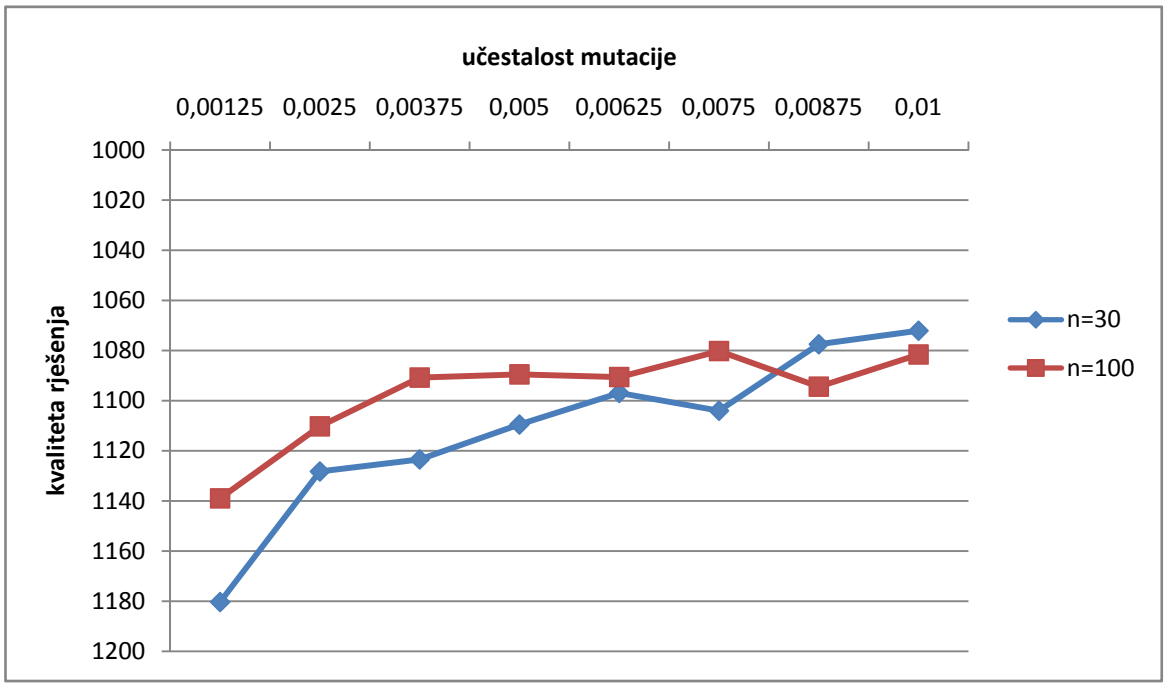
Slika 36-Graf ovisnosti kvalitete rješenja o vjerojatnosti mutacije

Na slici 36 su prikazani grafovi promjene kvalitete rješenja o promjeni mutacije. Prikazana su dva grafa. Parametri algoritma se razlikuju samo u veličini populacije. Crveni ima 100 jedinki u generaciji, a plavi 30. Na ordinati je prosjek najboljih rješenja za 40 ponavljanja. Operatori križanja se mijenjaju između poopćenog redoslijednog križanja i križanja s očuvanjem pravila prednosti. Također se mijenja inicijalizacija. U jednom slučaju je nasumično generirana populacija, a u drugom se populacija stvara posredstvom kromosoma odluke. Dakle, 2 varijacije križanja i 2 varijacije

inicijalizacije daju ukupno 4 različita načina. Algoritam se zaustavlja nakon 100 000 iteracija. Svaki se način ponavlja 10 puta, što daje 40 ukupnih ponavljanja. Testni primjer je veličine 10x10. Optimalno mu je rješenje 930 vremenskih jedinica, a to je primjer spomenut u poglavlju 2.2 koji je bio neriješen 25 godina.

Promatrajući graf može se zaključiti da na višim vjerojatnostima mutacije (veće od 0.005) bolje rezultate daje manja populacija. To je očekivano jer je široko prihvaćeno u literaturi [15]. Najbolji prosječni rezultat je postignut pri učestalosti mutacije od 2%.

Ako bi promatralo drugi skup vjerojatnosti mutacije, onaj s manjim vrijednostima i to između 0.001 i 0.01, rezultati bi bili drugačiji.

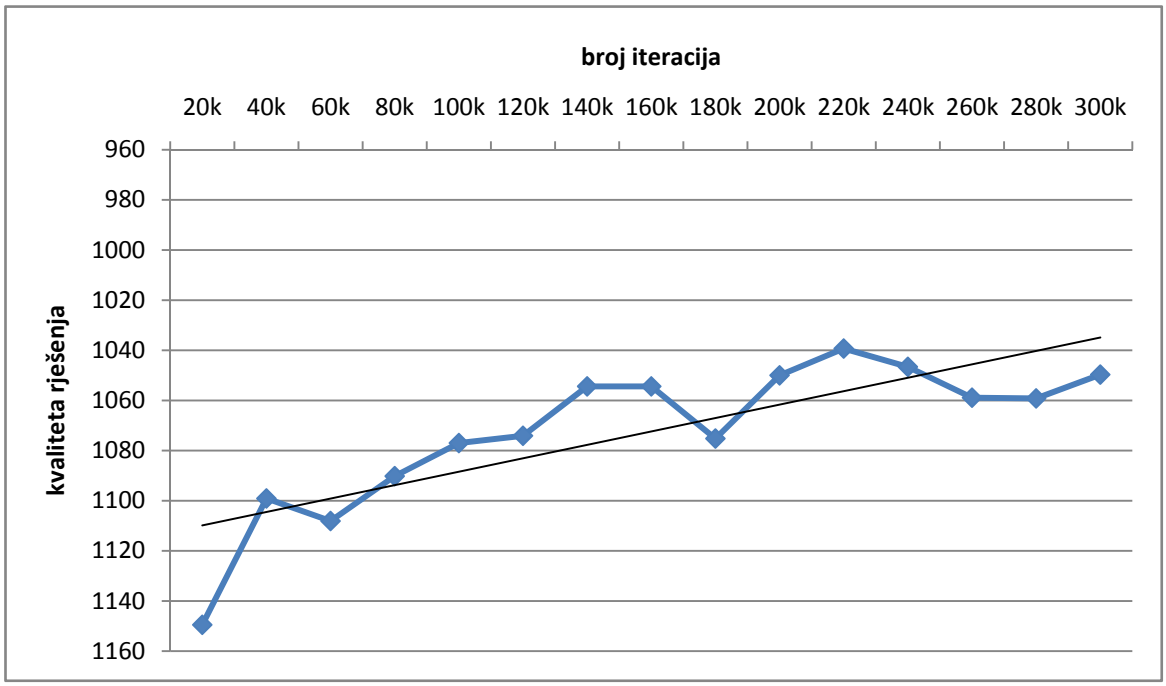


Slika 37- Usporedba male i velike populacije na manjim vrijednostima parametra vjerojatnosti mutacije

Kao što se vidi na slici 37, manja vrijednost parametra mutacije znači i dominaciju velike populacije nad malom. Ostali parametri su ostali isti kao u primjeru koji je prethodio ovome.

5.2.2 Utjecaj broja iteracija na kvalitetu rješenja i vrijeme izvođenja

Kao što je bilo i očekivano, rezultati su pokazali da porast broja iteracija znači i rast kvalitete rješenja. Na slici 38 je prikazana ovisnost broja iteracija i kvalitete rješenja.

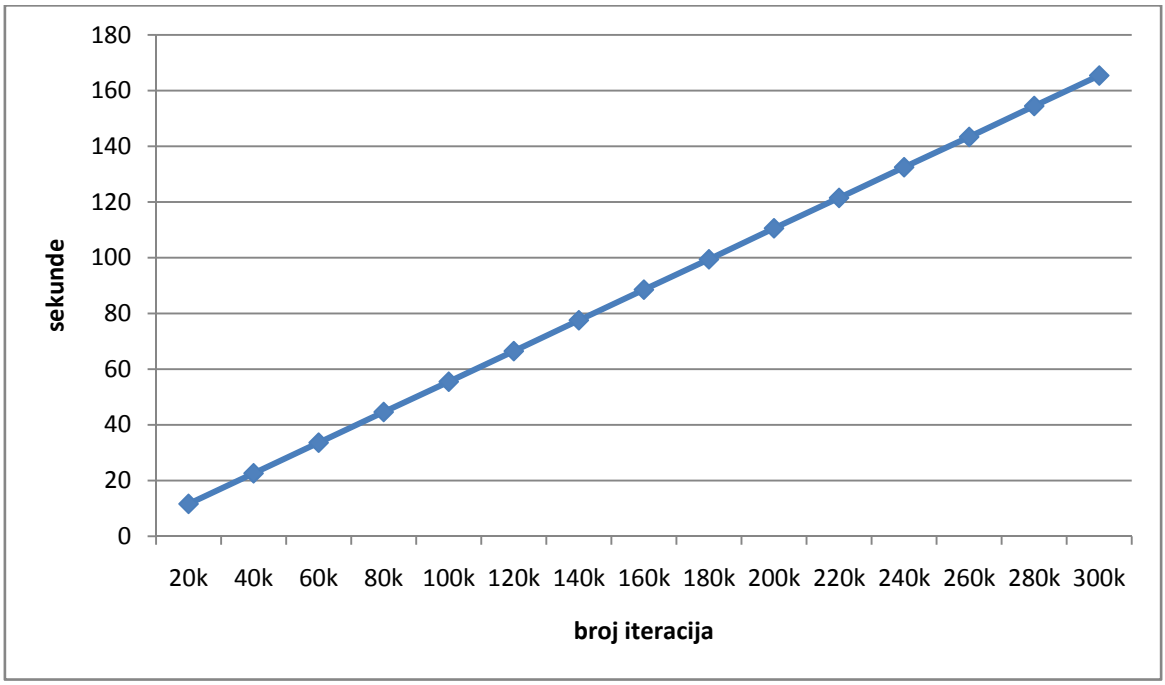


Slika 38 – Graf koji pokazuje porast kvalitete rješenja s porastom broja iteracija algoritma

Broj iteracija je varirao od 20 000, pa sve do 300 000, s korakom od 20 000. Svaka varijacija je imala 10 ponavljanja, i prikazane su prosječne vrijednosti. Sve varijacije su imale jednake vrijednosti sljedećih parametara. Korišteno je križanje s očuvanjem pravila prednosti,populacija je inicijalno stvorena slučajnim odabirom, broj jedinki u populaciji je bio 60 te je vjerojatnost mutacije je bila 0.01. Testni primjer je bio 10x10.

Na grafu se vidi da veći broj iteracija daje i veću kvalitetu rješenja. Prikazan je i linearni trend rasta. Bitno je i napomenuti da se pažljivijom analizom došlo do podatka da je prilikom maksimalnog broja iteracija od 300 000, zadnjih 250 000 generacija nije bilo poboljšanja najbolje jedinke. Algoritam je zapeo u lokalnom optimumu i nije uspio izaći iz njega. Ukoliko algoritam ima sreće, pronaći će bolje rješenje i u manjem broju iteracija, a ukoliko radi s lošom populacijom ni veliki broj iteracija neće mu pomoći.

Sljedeći podatak je također očekivan. Na slici 39 prikazana je ovisnost trajanja vremena izvođenja algoritma u odnosu na broj iteracija.

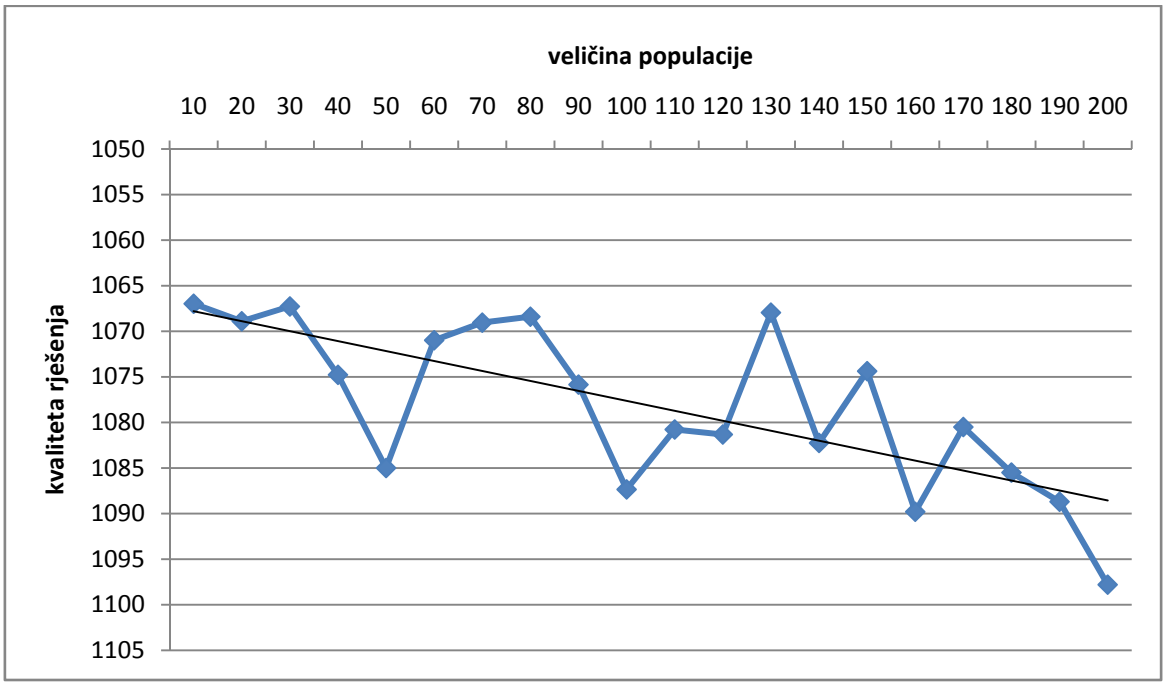


Slika 39- Graf porasta vremena izvođenja u odnosu na rast broja iteracija

Na slici 39 se vidi linearna ovisnost računske složenosti algoritma o broju iteracija. Trajanje algoritma najviše ovisi upravo o broju iteracija tako da, iako poboljšava kvalitetu rješenja, veći broj generacija ima i svoju negativnu stranu.

5.2.3 Utjecaj veličine populacije na kvalitetu rješenja i vrijeme izvođenja

Analiza utjecaja rasta broja jedinki u populaciji na kvalitetu rješenja nije dala tako jasne zaključke kao i prethodna ispitivanja. Na slici 40 je prikazan odnos kvalitete rješenja i veličine populacije.

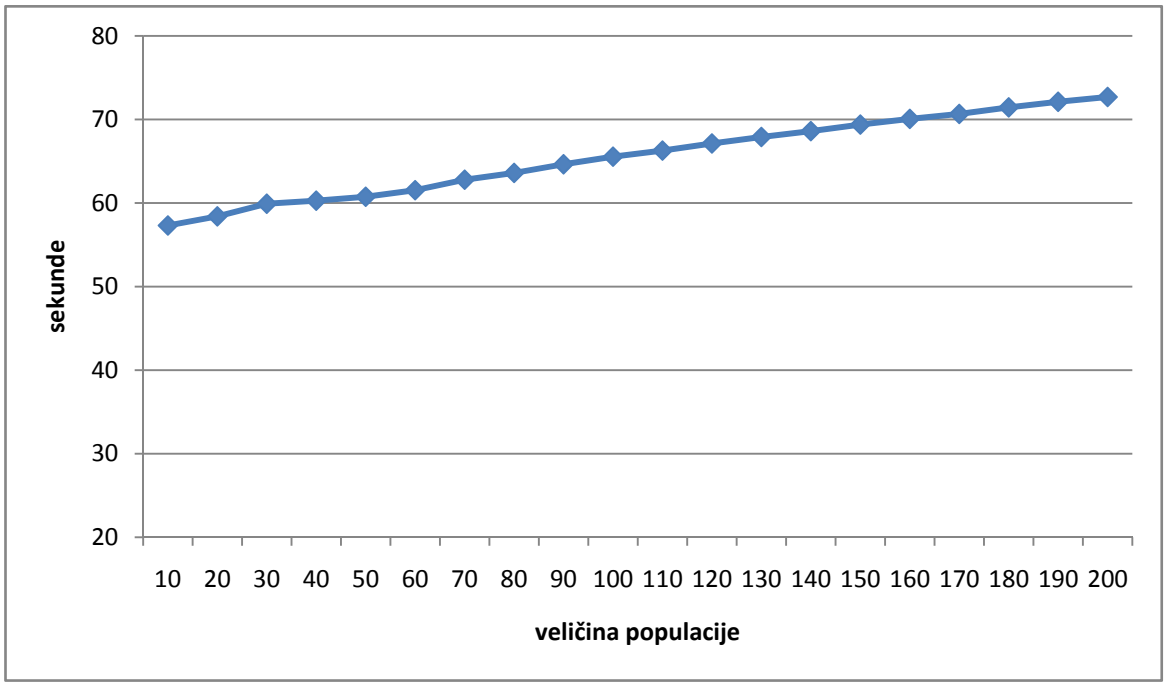


Slika 40 – Graf ovisnosti kvalitete rješenja o veličini populacije

Testni primjer je 10x10, kao i prije. Za sve varijacije sljedeći parametri su isti. Broj iteracija je 100 000. Vjerojatnost mutacije 0.01. Broj ponavljanja je 10. Operator križanja je križanje s očuvanjem pravila prednosti. Inicijalizacija je slučajnim odabirom. Na grafu su prikazane prosječne vrijednosti.

Iako se može povući linearni trend pada kvalitete rješenja, postoje velika odstupanja od te linije.

Analiza računalne složenosti o veličini populacije nije tako zbunjujuća. Na slici 41 je prikazan graf ovisnosti trajanja izvođenja o porastu populacije.

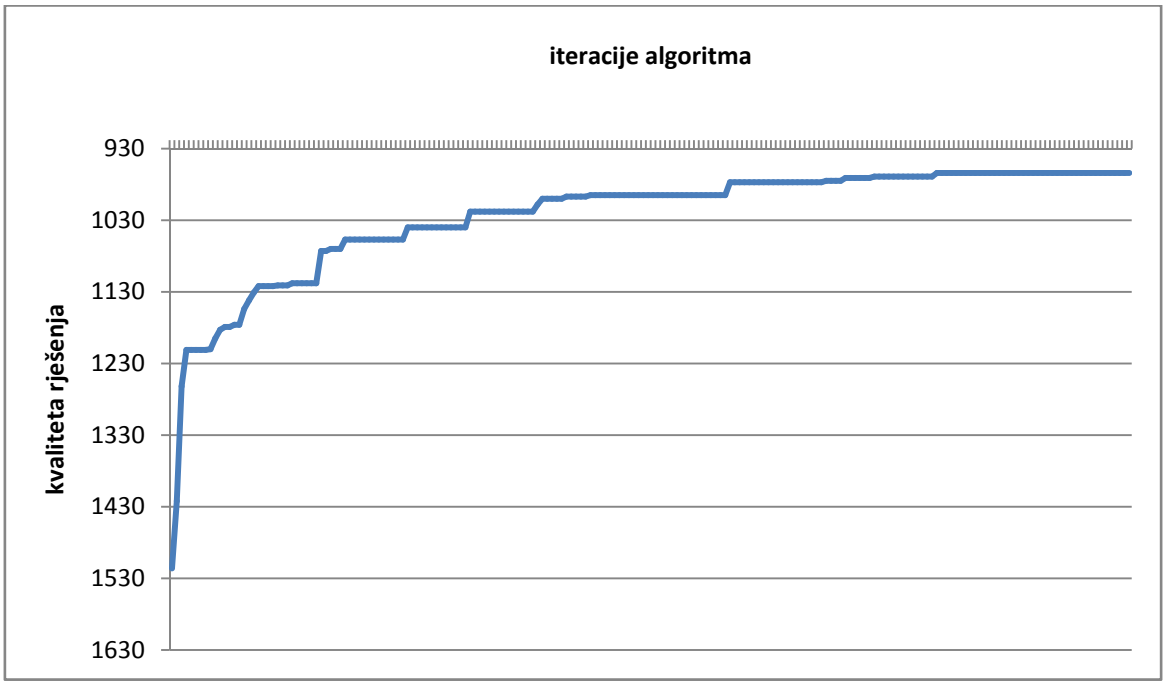


Slika 41-Graf utjecaja parametra veličine populacije na trajanje izvođenja algoritma

Na slici 41 se vidi linearni rast vremena izvođenja s porastom veličine populacije. Iako računalna složenost raste s većom populacijom, to nije tako strašni porast kao što je bio u slučaju porasta broja iteracija (koeficijent rasta je znatno manji). Porast populacije ima najviše utjecaja na složenost inicijalizacije, što malo usporava algoritam.

5.2.4 Praćenje napretka najbolje jedinke kroz iteracije algoritma

Graf praćenja napretka najbolje jedinke u populaciji kako prolaze generacije algoritma služi da bi se vidjela kvaliteta cijelog postupka. U idealnom slučaju prikaz sliči grafu eksponencijalne funkcije. U tom slučaju radi se o temeljitom pretraživanju prostora stanja. [16] Na slici 42 dan je prikaz napretka najbolje jedinke u izvedbi algoritma koja je dala najbolje rješenja za problem 10x10 (u iznosu od 964 vremenske jedinice).



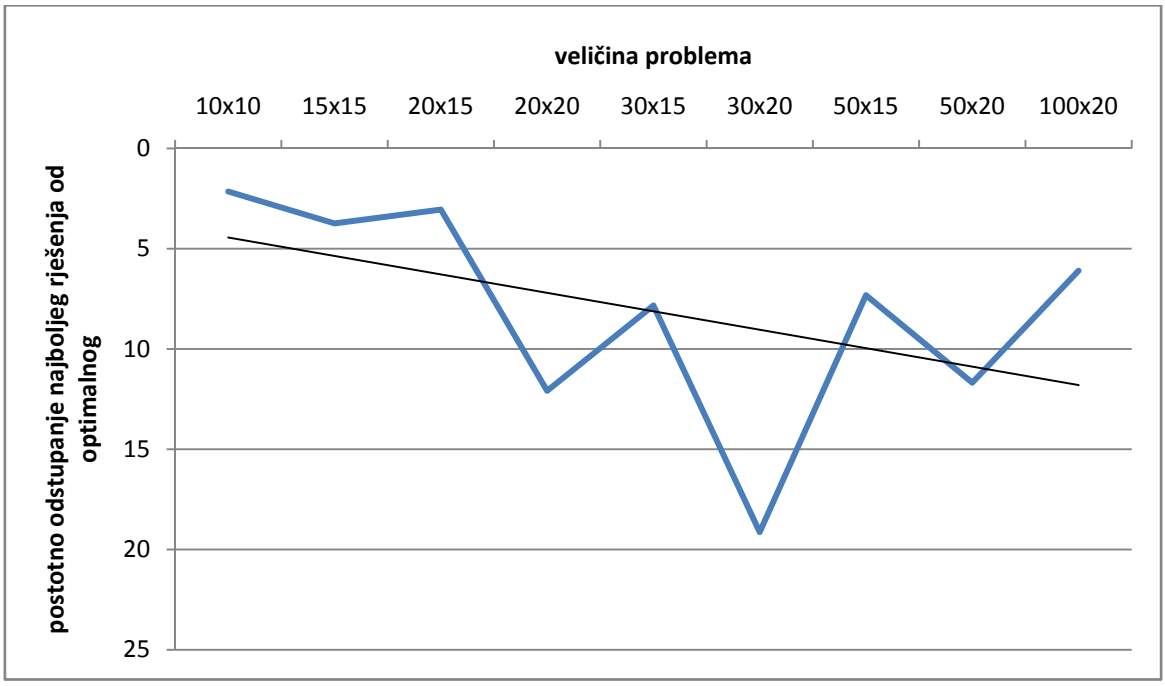
Slika 42 – Praćenje napretka najbolje jedinke kroz iteracije algoritma

Prikazana izvedba je koristila križanje s očuvanjem pravila prednosti, 100 000 iteracija, 60 jedinki u generaciji, vjerojatnost mutacije od 0.01 te nasumičnu inicijalizaciju početne generacije.

Analizom grafa vidi se da ima dosta skokova, što znači da je daleko od idealnog slučaja. Što je više skokova, to je naznaka prevelike učestalosti mutacije i zapravo slučajnosti da je pronađeno bolje rješenje [16].

5.3 Utjecaj veličine problema na kvalitetu rješenja i vrijeme izvođenja

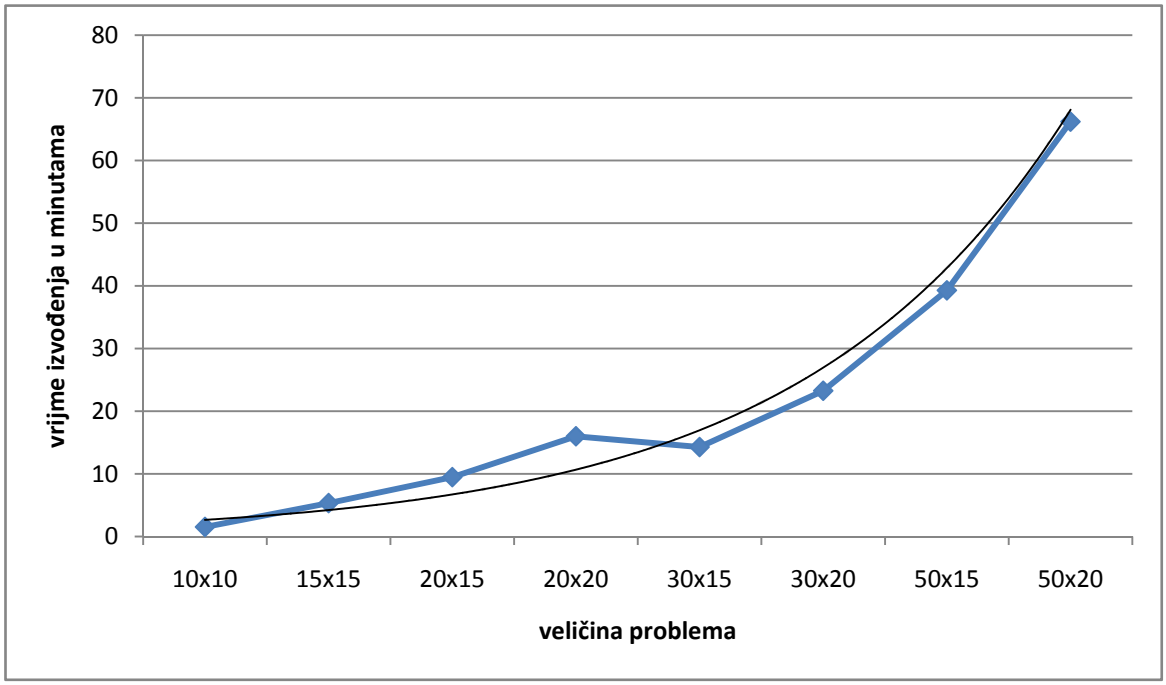
Veličina problema je zadana preko dva broja, broja poslova (*n*) te broja strojeva (*m*). Veća vrijednost tih brojeva znači i veći veličinu problema. Kod manjeg problema prostor pretraživanja je manji pa je lakše u istom vremenu postići dobar rezultat nego što je to u slučaju većeg problema. Na slici 43 prikazano je odstupanje kvalitete rješenja koje je postignuto ovim algoritmom od optimalnog rješenja ili najboljeg poznatog rješenja, ukoliko za veće probleme optimalnost nije dokazana. Mjeri se u postotcima i računa se po formuli $\frac{postignuto - optimalno}{optimalno}$.



Slika 43 – Graf ovisnosti kvalitete rješenja s povećanjem veličine problema

Na grafu se vidi trend pada kvalitete rješenja s povećanjem veličine problema.

Rast veličine problema ima utjecaj i na vrijeme izvođenja. Kako su kromosomi veliki $n * m$, veći problemi znače i veće kromosome. Potrebno je duže vremena za inicijalizaciju, evaluaciju i križanje. Na slici 44 je prikazan graf koji pokazuje taj rast.

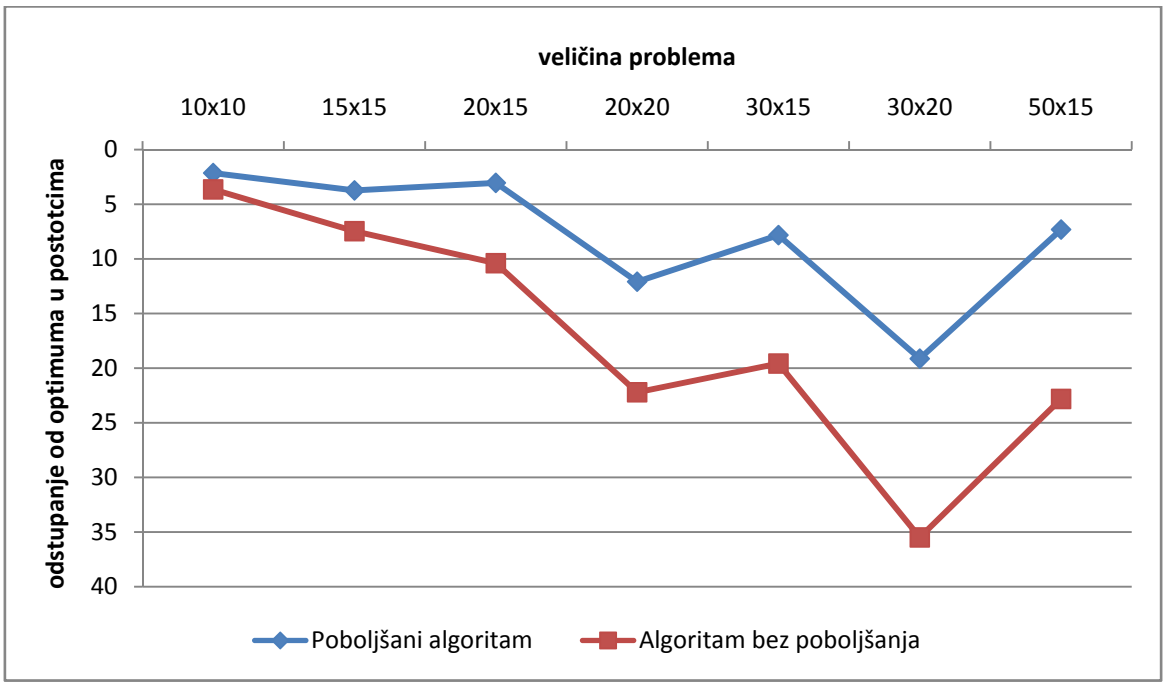


Slika 44- Utjecaj veličine problema na računalnu složenost

Na slici 44 je prikazan rast vremena izvođenja s porastom veličine problema. Veličina problema je dana umnoškom broja poslova i broja strojeva. Iako veličine na apscisi ne rastu potpuno linearno, to je približno tako. (100, 225, 300, 400, 450, 600, 750, 1000). Može se povući eksponencijalni trend rasta. Bez analize rezultata moglo bi se pretpostaviti da je ovisnost trajanja izvođenja o veličini problema linearna, ali detaljnija analiza pokazuje da je utjecaj veći.

5.4 Utjecaj poboljšanja rasporeda pomicanjem operacija ulijevo na kvalitetu rješenja i vrijeme izvođenja

U poglavlju 4.4.4.1 je opisano kako se raspored može poboljšati pomicanjem operacija ulijevo na Ganttovom dijagramu. To je dovelo do poboljšanja kvalitete rješenja.



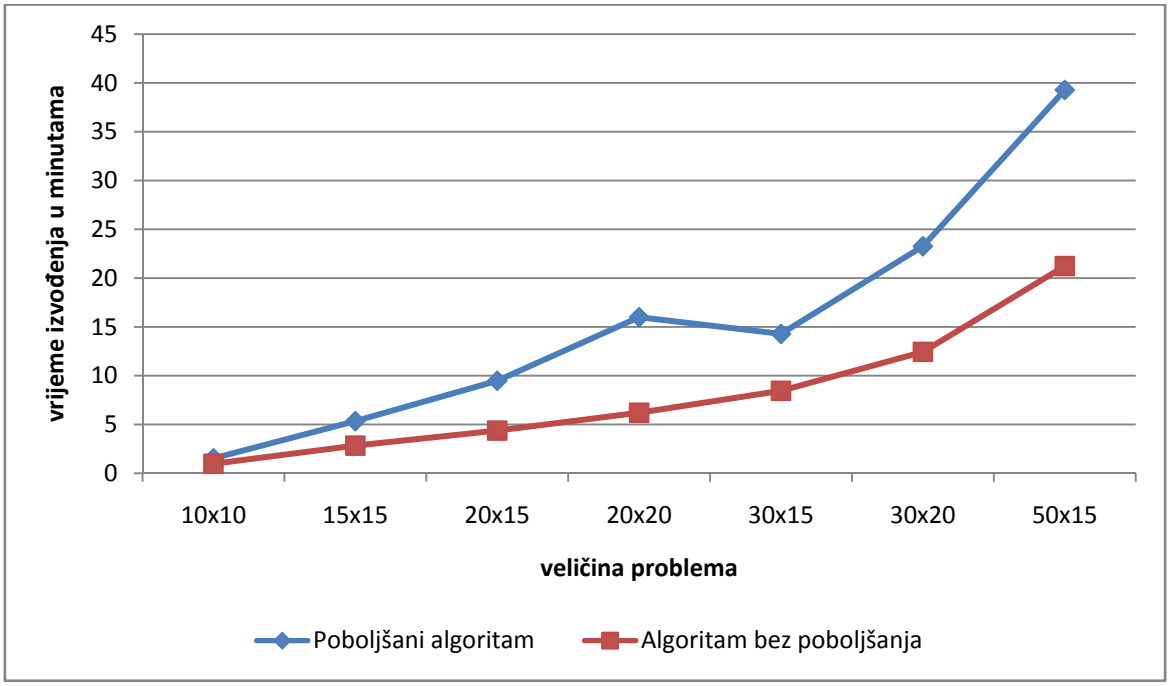
Slika 45 - Usporedba poboljšanog algoritma u odnosu na onaj bez poboljšanja

Na slici 47 su predstavljena dva grafa. Plavi predstavlja algoritam koji koristi pomicanje operacija ulijevo, a crveni predstavlja algoritam koji to ne koristi. Na apscisi su veličine problema, a na ordinati je odstupanje najboljeg rješenja od optimalnog u postotcima. U oba slučaju za svaki od 7 primjera problema je korišteno 40 ponavljanja. Vjerojatnost mutacije je jednaka, 0.01. Populacije ima 30 jedinki. Broj generacija je 100 000.

Poboljšani algoritam daje bolje rezultate. Prosječno su bolji za 9.5%. Na manjim inačicama problema razlika je manja. Kako raste veličina problema, kvaliteta algoritma dolazi do izražaja.

Slijedi analiza koliko je skupo to poboljšanje, odnosno za koliko je povećana računalna složenost.

Na slici 48 je prikazana dva grafa. Plavi predstavlja algoritam koji koristi pomicanje operacija ulijevo, a crveni onaj koji to ne koristi. Na apscisi su veličine problema, a na ordinati vrijeme izvođenja u minutama.



Slika 46 - Utjecaj poboljšanja algoritma na računalnu složenost

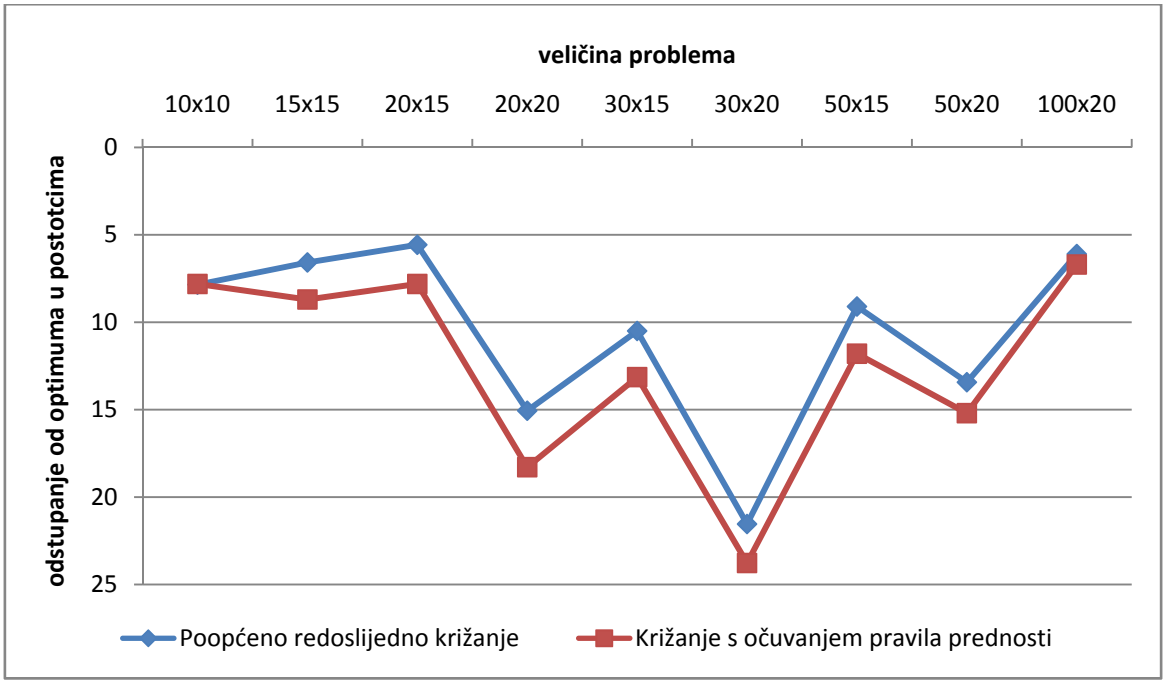
Sa slike 48 se vidi da je povećanje složenosti algoritma uzrokovalo povećanje vremena izvođenja za otprilike dvostruko.

Poboljšanje algoritma donosi veću kvalitetu generiranog rasporeda, ali pod cijenu dužeg trajanja izvođenja programa. Potrebno se odlučiti koje je mjerilo učinkovitosti važnije, i u skladu s time odabrati jednu od dvije verzije algoritma.

5.5 Usporedba različitih operatora križanja i prikaza rješenja

Analiza kvalitete rješenja kod većih problema je pokazala kako poopćeno redoslijedno križanje daje nešto bolje rezultate od križanja s očuvanjem pravila prednosti.

Odnos je prikazan na slici 45.



Slika 47 – Odnos kvalitete rješenja s obzirom na izbor operatora

Poopćeno redoslijedno križanje daje malo bolje rezultate. Razlika je vrlo mala, otprilike 2%, ali je prisutna.

Ako se promatra vrijeme izvođenja, oba križanja su ravnopravna.

Uniformno križanje se pak pokazalo uvjerljivo najlošije od ova tri operatora, i sa strane kvalitete rješenja i sa strane računalne složenosti.

Različiti načini inicijalizacije su se pokazali jednako dobrima. Stvaranje početne populacije preko slučajnog odabira te to isto posredstvom kromosoma odluke daju podjednako dobre rezultate te imaju slična vremena izvođenja.

5.6 Ispitivanje konkurentnosti s drugim metodama rješavanja problema proizvoljne obrade

U proteklih pola stoljeća bilo je mnogo pokušaja rješavanja problema proizvoljne obrade. Algoritam koji bi jamčio optimalno rješenje još ne postoji, a postoji mišljenje da neće niti biti pronađen. Genetski algoritmi kao samostalna metoda se ne smatraju osobito dobrima u ovoj domeni. U literaturi [11] su navedeni rezultati testiranja na problemu 10x10 kojemu se ovaj algoritam podvrgnuo. Iz tog rada je preuzeta tablica na slici 46.

	Makespan Achieved			CPU Time (secs)			
Author	FT 06	FT 10	FT 20	FT 06	FT 10	FT 20	Machine Used
Namada & Yakano ('91) - NY'91	55	965	1215	NG	NG	NG	NG
Yamada & Nakano ('92) - NY'92	55	930	1184	NG	600	NG	SUN Sparcstation 2
Davidor *et al.* ('93) - DYN'93	55	930	1181	NG	300	1800	SUN Sparcstation 2
Fang *et al.* ('93) - FRC'93	55	949	1189	NG	1500	1500	SUN 4
Storer *et al.* ('93) - SWP'93	55	954	1180	NG	55	75	CDC 4340 - R3000
Pesch ('93) - 2J-GA	55	937	1193	8.1	100.0	95.3	VAX 8650
Pesch ('93) - 2JCP-GA	55	937	1175	11.4	146.9	121.1	VAX 8650
Pesch ('93) - 1MCP-GA	55	930	1165	8.5	104.4	101.0	VAX 8650
Mattfeld *et al.* ('94) - MKB'94	55	930	1165	NG	138	138	SUN Sparcstation 10
Della Croce *et al.* ('95) - DTV'95	55	946	1178	223	628	675	PC 486 (25MHz)
Bierwirth ('95) - B'95	55	936	1181	NG	135	147	SUN Sparcstation 10
Dorndorf and Pesch ('95) - DP'95	55	938	1178	19.7	106.7	95.7	DEC Station 3100
Yamada & Nakano ('95b) - YN'95	55	930	1165	NG	699	654	SUN Sparcstation 10
Mattfeld ('96) - Mf'96	55	930	1165	6	40	47	SUN 10/41
Yamada & Nakano ('96b,c) - YN'96	55	930	1165	NG	88	55	Dec Alpha 600 5/266
Norman & Bean ('97) - NB'97	55	937	1165	NG	300	300	SUN Workstation
Shi ('97) - S'97	55	930	1165	NG	NG	NG	SONY NWS - 3460

NG - This information is not given

Slika 48 – Rezultati testiranja različitih implementacija genetskog algoritma

U prvom stupcu tablice su dana imena autora te godina testiranja. Sljedeća dva stupca daju rezultate testiranja na probleme 6x6 (na slici 46 FT 06) i 10x10 (na slici FT 10) koji su bili dostupni na internetu i kojima je implementacija u ovom radu podvrgnuta. U slučaju manjeg problema 6x6 postignuto je optimalno rješenje od 55 vremenskih jedinica, što je slučaj i s ostalim implementacijama. Kod većeg problema 10x10 nije postignuto optimalno rješenje od 930, nego je najbolje rješenje imalo trajanje od 950 što je lošije od implementacija prikazanih u tablici.

Iz toga se zaključuje da program nije konkurentan i da postoje bolje implementacije. Iako treba podsjetiti da ostale implementacija ne koriste samo genetski algoritam nego su hibridne tehnike koje imaju objedinjuju više metoda. One se i inače smatraju boljima od samostalnog genetskog algoritma.

6. Zaključak

Problem proizvoljne obrade je popularan u literaturi i ima svoju praktičnu primjenu. Budući da je NP-težak heurističke metode su dobar izbor za njegovo rješavanje, jer egzaktni postupci ne mogu dati rješenje u realnom vremenu. Genetski algoritam je jedna takva metoda, zanimljiv jer koristi prirodnu metaforu i učinkovit u davanju rezultata. Definiran kao u ovome radu, usko je vezan za problem i ne može se koristiti za druge primjene.

Genetski operatori križanja odabrani u ovom radu imaju pozitivno svojstvo kojim uvijek daju izvedivog potomka, pa je implementiranje lakše jer nema oporavka od pogreške, ali negativno je što se više reprezentacija može prevesti u isti raspored. Operator poopćenog redoslijednog križanja se pokazao nešto malo bolji od križanja s očuvanjem pravila prednosti.

Kromosomi odluke se mogu eventualno koristiti kao sredstvo inicijalizacije, a zatim se prevesti u permutaciju s ponavljanjem. Korištenjem kromosoma odluke kao jedinog prikaza rješenja se pokazalo poražavajućim, i u pogledu kvalitete rješenja i računalne složenosti.

Poboljšanje rasporeda pretvaranjem iz poluaktivnog u aktivni pomicanjem operacija ulijevo ima svojstvo veće kvalitete, ali ima cijenu veće računalne složenosti. Tu se potrebno odlučiti koje je mjerilo važnije.

Programska implementacija je bila jednostavna. Korišten je C# programski jezik, pa je negativno što je za izvođenje potreban .NET Framework 1.1 ili viši, što donekle otežava primjenu, iako je spomenuti .NET Framework prilično popularan.

Dok algoritam manje probleme rješava optimalno (primjerice 6x6), povećanjem veličine problema raste i postotno odstupanje od optimalnog rješenja ili najboljeg poznatog.

Rezultati koje je algoritam polučio nažalost nisu konkurentni s drugim implementacijama koje koriste genetski algoritam za rješavanje istog problema , a kako općenito genetski algoritmi kao samostalna metoda nisu stavljeni visoko u domeni ovog problema, definitivno postoje bolji postupci za rješavanje istoga.

7. Literatura

[1] Takeshi Yamada i Ryohei Nakano, **Genetic Algorithms for Job-Shop Scheduling Problems**, 18-19.ožujka 1997, *Proceedings of Modern Heuristic for Decision Support*,22.listopada 2008.

[2] J.F.Muth i G.L.Thompson (1963.), *Industrial Scheduling,* Prentice-Hall, Englewood Clifs, N.J.

[3] Hsiao-Lan Fang, Peter Ross i Dave Corne, **A Promising genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems**, *Proceedings of the Fifth International Conference on Genetic Algorithms,* S.Forrest(ed.),San Mateo: Morgan Kaufmann,1993, stranice 375-382

[4] Manuel Vazquez i L.Darrell Whitley, **A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem**

[5] Jose Fernando Goncalves, Jorge Jose de Magalhaes Mendes i Mauricio G.C.Resende, **A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem**, *AT&T Labs Research Technical Report TD-5EAL6J,* Rujan 2002.

[6] Wikipedia,the free encyclopedia, 23 Kolovoz 2008 ,http://www.wikipedia.org/JobShop.html, 27.Travanj 2009.

[7] Ana Madureira, Carlos Ramos, Silvio do Carmo Silva, **A Genetic Approach for Dynamic Job-Shop Scheduling Problems**, *MIC'2001 - 4th Metaheuristics International Conference*

[8] Yuri N.Sotskov, Natalia V.Shakhevich, Frank Werner, **Mixed Shop Scheduling Problems**

[9] Christian Blum, **ACO applied to Group Shop Scheduling: A case study on Intensification and Diversification**

[10] Debra J.Hoitomt, Peter B.Luh, Krishna R.Pattipati, **A Practical Approach to Job-Shop Scheduling Problems**, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL.9.,NO.1,VELJAČA 1993

[11] Anant Singh Jain, Sheik Meeran, **A State of the art review of Job Shop scheduling techniques,** 3.Listopad 1993.

[12] Albert Jones, Luis C. Rabelo, **Survey of Job Shop Scheduling Techniques**

[13] Ganapathi Kamath, Kaifu Hong,**The Job Shop Scheduling Problem,** *CIS 675, Design & Analysis of Algorithms Syracuse University*

[14] Goran Radanović, **Pregled heurističkih algoritama**, Seminar, FER 2007.

[15] Golub, M: ***Genetski algoritam – prvi dio***, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf , 2004.

[16] Damjanić, T. **Primjena evolucijskih algoritama za rješavanje aproksimacijskog problema**, diplomski rad, FER, Zagreb, 2008.

[17] James C. Werner, Mehmet E. Aydin,Terence C. Fogarty, **Evolving genetic algorithm for Job Shop Scheduling problems,** *Proceedings of ACDM 2000*

[18] Frankola, T., **Rješavanje problema raspoređivanja aktivnosti projekata evolucijskim algoritmima**, diplomski rad, FER, Zagreb, 2006.

[19] Christian Bierwirth, **A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms**, Department of Economics, University of Bremen, Germany