Primjena evolucijskih algoritama u problemima raspoređivanja

Tehnička dokumentacija

Verzija 1.0

Studentski tim: Petar Čolić
Filip Domazet
Karlo Knežević
Ivo Majić
Luka Marasović
Marin Maržić
Ante Modrić

Nastavnik: Prof.dr.sc. Marin Golub
Doc.dr.sc. Domagoj Jakobović

Sadržaj

Problem krojenja 1D

Problem jednodimenzionalnog rezanja spada u skupinu problema rezanja i pakiranja (*Cutting and Packing Problems* – C&P problemi). Problem rezanja je problem kombinatoričke optimizacije koji uključuje rezanje raspoloživog materijala (*stock*) u potrebne manje komade (zahtjeve za rezanjem - *item*) sa ciljem da se minimalizira broj korištenih komada raspoloživog materijala i količina otpada (materijala koji se više ne može koristiti). Za rješavanje se općenito koriste 2 pristupa: fokusiranje na objekte i fokusiranje na uzorke. Prvi pristup direktno pridružuje zahtjeve za rezanjem komadima raspoloživog materijala, dok drugi pristup prvo generira uzorke rezanja raspoloživog komada, a zatim kombinira dobivene uzorke da bi se ostvarili svi zahtjevi za rezanjem.

U sklopu projekta se rješavao problem rezanja koji se prema tipologiji C&P problema koju je H. Dyckhoff [1] postavio karakterizira sa 1/V/D/M :
1 - jednodimenzionalni problem
V - problem biranja izvornog materijala da se ostvare svi zahtjevi za rezanjem
D - raspoloživo je više veličina izvornog materijala
M - velika količina zahtjeva za rezanjem u puno različitih duljina

Problem se može matematički formulirati na idući način:

$$\text{Minimise } C = \sum_{j=1}^{m} w_j$$

$$\text{gdje je } w_j = L_j - \sum_{i=1}^{n} x_{ij} l_i - \sum_{i=1}^{n} x_{ij} s$$

$$\text{uz } \sum_{j=1}^{m} x_{ij} = N_i \text{ gdje je:}$$

n - broj različitih duljina zahtjeva za rezanjem
m - broj iskorištenih komada raspoloživog materijala
w_j - otpad nakon rezanja j-tog komada materijala
L_j - duljina j-tog komada raspoloživog materijala
x_{ij} - broj komada i-tog zahtjeva za rezanjem koji se režu od j-tog komada materijala
l_i - duljina i-tog zahtjeva za rezanjem
N_i - ukupan broj komada i-tog zahtjeva za rezanjem

Također je zadano da se preferiraju rješenja koje koriste raspoložive materijale kojih ima konačan broj (ostatci od prijašnjih rješavanja zadataka).
Zbog očekivanja više različitih duljina raspoloživog materijala, pristup fokusiranjem na uzorke nije prikladan za rješavanje problema zbog količine uzoraka koje bi se trebalo generirati. Zbog toga je za rješavanje problema odabran pristup fokusiranjem na objekte.

1.1. Optimiranje algoritmom MinMaxAS (Filip Domazet)

1.1.1. Primjena algoritma na zadani problem

Značenje tragova i računanje vjerojatnosti odabira

Korišten optimizacijski algoritam kolonije mrava bazirao se na Max-Min Ant Systems verziji. U Max Min Ant System algoritmu postoji donja i gornja granica na vrijednosti tragova, te da tragove postavlja samo najbolji mrav [3], no u ovom slučaju gornja granica tragova se neće primjenjivati.

Koriste se dva skupa tragova, jedan koji se koristi pri određivanju duljine materijala koju treba uzeti, a drugi koji pri određivanju izreska kojeg treba dodati u plan rezanja već odabranog materijala.

Što se tiče tragova među izrescima, značenje tragova i formule računanja vjerojatnosti se oslanjaju na [2]. Trag između izreska duljine *i* te izreska duljine *j* , tj. τ(i, j) se koristi kao mjera povoljnosti da se u planu rezanja materijala duljine *x* nalaze izresci duljina *i* i *j* (vrijedi τ(i, j) = τ(j, i)).

Prema tragovima među izrescima koji su u planu rezanja (označimo taj skup sa *s*) i izrescima koje još možemo dodati računa se vjerojatnost odabira pojedinog izreska koristeći sljedeću formulu:

$$p(i,s) = \frac{\tau_b(i)}{\sum_{g \in J(x,s)} \tau_b(g)},$$

gdje J(x, s) skup izrezaka koji još „stanu“ u plan rezanja materijala. Gornja formula vrijedi ako je $i \in J(x,s)$, u protivnom je p(i, s) = 0. $\tau_b(i)$ je prosječna vrijednost tragova između izreska duljine i i svih izrezaka trenutno u planu rezanja (s), formalno:

$$\tau_b(i) = \begin{cases} \frac{\sum_{j \in s} \tau(i,j)}{|s|}, & ako\ s \neq \emptyset \\ 1, & inače \end{cases}$$

Što se tiče tragova za duljine materijala, oni predstavljaju koliko je određena duljina materijala povoljna za koristiti. Vjerojatnost odabira materijala duljine x se računa prema formuli:

$$p(x) = \frac{\tau(x)\eta(x)}{\sum_{y \in L(k)} \tau(y)\eta(y)},$$

gdje L(k) predstavlja skup raspoloživih duljina u koraku k (nakon k odabira), a η(x) predstavlja heurističku povoljnost odabira materijala duljine x. Heuristička povoljnost se zadaje kao parametar koji daje prednost duljinama materijala koje su u ograničenim zalihama (pretpostavlja se da su to ostaci prijašnjih rezova kojih se želi riješit).

Funkcija dobrote

Prema [2] funkcija dobrote rješenja S računa se po sljedećoj formuli:

$$f(S) = \frac{\sum_{i=1}^{N} \left(\frac{Fi}{x_i}\right)^k}{N},$$

gdje je N broj korištenih materijala u rješenju S, F_i zbroj izrezaka u planu rezanja i-tog materijala, a x_i ukupna duljina i-tog materijala. Pomoću eksponenta k može se podešavati kakva se rješenja vrednuju više. Ako je k = 1, onda se gleda samo prosječna iskorištenost, a za k > 1, bolja se smatraju rješenja koja imaju mješavinu dobro iskorištenih materijala i manje dobro iskorištenih materijala. Prilagodba koja se koristi pri ovom algoritmu jest da se materijal smatra potpuno iskorištenim ako je ostatak manji od duljine koja se smatra prihvatljivom (označimo s *d*), tj. ako $x_i - F_i < d$. onda se umjesto $\left(\frac{Fi}{x_i}\right)$ stavlja 1.

Postavljanje tragova

Tragove postavlja najbolji mrav, i to naizmjence najbolji u iteraciji i globalno najbolji, pri čemu globalno najbolji tragove postavlja svakih γ iteracija. Vrijednosti tragova vezanih uz duljine materijala se povećaju za „prosječnu“ iskorištenost materijala određene duljine (računa sa prema formuli funkcije dobrote, ali samo za materijale te duljine) unutar rješenja.

Tragovi među izrescima *i* i *j* u za materijal duljine x, se povećaju za $m \cdot f(S)$, pri čemu je m broj pojavljivanja materijala duljine i zajedno sa materijalom duljine j unutar istog plana rezanja materijala duljine x. Nakon postavljanja tragova, primjenjuje se isparavanje, tako da se vrijednosti svih tragova pomnože sa parametrom isparavanja ρ, $\rho \epsilon (0,1)$.

Zbog navedene definicije tragove i ovakvog načina računanja tragova, nije postavljena gornja granica na vrijednost tragova. Donja granica vrijednosti tragova se zadaje kao parametar algoritma.

1.2. Optimiranje algoritmom YAEAuzLS (Marin Maržić)

1.2.1. Primjena algoritma na zadani problem

Program koristi evolucijske strategije kao tehniku optimizacije pri rješavanju problema krojenja 1D (engl. cutting-stock problem ili CSP) s različitim veličinama zaliha i rezova (uključujući beskonačne zalihe). Također periodično primjenjuje varijantu lokalnog pretraživanja nad jedinkama populacije u nastojanju poboljšanja rješenja.

Evolucijske strategije su optimizacijske tehnike iz područja evolucijskih algoritama bazirane na prilagođavanju i evoluciji. Njihova središnja ideja je stvaranje jednog ili λ potomaka operacijom mutacije iz jednog ili μ roditelja. Roditelji i potomci predstavljaju potencijalno rješenje optimizacijskog problema. [4]

YAEA primjenjuje evolucijsku strategiju varijante (μ + λ)-ES gdje je μ broj roditelja koji mutacijom stvaraju λ potomaka. Selekcijom se iz skupa roditelja i potomaka odabire μ najboljih jedinki koji prelaze u sljedeću generaciju kao novi budući roditelji. U ovoj implementaciji je μ proizvoljan broj jedinki početne populacije roditelja. λ je cjelobrojni višekratnik od μ (λ = n * μ, n = 1, 2, ...). Npr. μ = 4 pa λ može poprimiti vrijednosti 4, 8, 12, ...

Zapis jedinke rješenja je baziran na redoslijedu nizova zalihe i rezova [5]. Redom rezovi "ulaze" u zalihu tj. bivaju izrezani iz nje dok ju ne "popune" tj. potroše (sljedeći rez u nizu ne može biti izrezan iz preostale zalihe). Kad je zaliha potrošena sljedeći rezovi u nizu ulaze u sljedeću zalihu u nizu. Ukoliko je rez veći od zalihe ta se zaliha preskače kao da je nije bilo. Npr. neka je niz zaliha [10, 5, 8, 9], a niz rezova [4, 4, 4, 9]. Tada će se iz zalihe 10 rezati dva reza 4, iz zalihe 5 jedan rez 4, a zaliha 9 će biti rez 9.

Mutacija proizvoljan broj puta radi zamjenu elemenata nizova zaliha i rezova u nasumične tri točke (3 elementa niza zamjene pozicije). Ta zamjena je izvedena nasumičnim odabiranjem elemenata niza indeksa i, j, k i njihovim preuređenjem u niz k, i, j. Npr. ukoliko mutacija radi jednu zamjenu i imamo niz rezova [1, 2, 3, 4, 5], a nasumično su odabrani indeksi 2, 4, 5, onda će niz nakon mutacije biti [1, 5, 3, 2, 4].

Selekcija rangira skup roditelja i potomaka po funkciji dobrote što postavlja μ bolje rangiranih jedinki kao roditelje iduće generacije.

Funkcija dobrote jedinku rješenja vrednuje bolje po kriterijima [6]:

- manje gubitaka pri rezanju
- više savršenih rezova tj. unutar granice dovoljno dobrog
- manje korištenja beskonačnih zaliha
- ekstremniji raspored rezova tj. gubitaka (jako veliki i jako mali gubitci)

$$\min \frac{1}{J+2}\left(\sum_{j=1}^{J}\sqrt{\frac{T_j}{L_j}}+\frac{1}{J}\sum_{j=1}^{J}t_j+\frac{1}{K}\sum_{j=1}^{J}u_j\right)$$

gdje je:

- J = broj zaliha korišten u rješenju
- K = broj beskonačnih zaliha korišten u rješenju
- Tj = gubitak pri rezanju j-te zalihe
- Lj = veličina j-te zalihe
- tj = 1 ukoliko je j-ta zaliha dovoljno dobro iskorištena, inače 0
- uj = 1 ukoliko je j-ta zaliha beskonačna, inače 0

YAEAuzLS radi slično kao YAEA. Razlika je u periodičnom provođenju varijante lokalnog pretraživanja (LS) između mutacije i selekcije. Frekvencija provođenja LS je proizvoljna. LS se provodi nad skupom roditelja i potomaka.

LS radi na način da "oslobodi" neke zalihe i pripadajuće rezove tako da zalihe stavi na kraj redoslijeda, a rezove izbaci te ih pokuša efikasnije ubaciti u rješenje.

Oslobođeni rezovi se pokušavaju ubaciti natrag u rješenje počevši od najvećih. Prvo se za rez pokuša naći potpuno slobodno mjesto. Ukoliko to ne uspije pokuša se zamijeniti s nekim manjim rezom tako da još uvijek "stane" u zalihu (ne zahtjeva korištenje nove zalihe), dok se manji rez oslobađa. Ukoliko ni to ne uspije rez se stavlja na kraj redoslijeda na svoje staro mjesto tj. u staru zalihu.

1.3. Optimiranje grupirajućim genetskim algoritmom (Ante Modrić)

1.3.1. Primjena algoritma na zadani problem

Genetski algoritmi (GA) su skupina optimizacijskih metoda gdje se do rješenja dolazi pomoću imitiranja prirodne evolucije nad populacijom kodiranih rješenja (kromosomi). Evolucija se imitira postupcima selekcije, križanja (reprodukcije) i mutacije. Selekcijom, u kojoj se prednost daje boljim jedinkama (kodiranim rješenjima), se biraju jedinke od kojih će se križanjem i mutacijom kromosoma generirati nove jedinke.
Problem rezanja se može interpretirati kao problem grupiranja elemenata skupa (zahtjevi za rezanjem) u podskupove (komade raspoloživog materijala od kojih će se rezati). Zbog toga za rješavanje zadanog problema rezanja korišten je grupirajući genetski algoritam.
Grupirajući genetski algoritam (GGA) se razlikuje od tradicionalnih GA, u kojima svaki gen kromosoma predstavlja jedan element i njegova pozicija ili poredak u odnosu na ostale gene u kromosomu je bitan, po tome što u GGA gen predstavlja grupu elemenata, te njegova pozicija ili poredak u odnosu na druge gene u kromosomu nije bitan. Također, za razliku od tradicionalnih GA, gdje se nove jedinke generiraju kombinacijom križanja i mutacije, kod GGA se jedinke generiraju ili križanjem ili mutacijom.

Grupiranje
Problem prilagođavanja Falkenauerovog GGA (konstruiranog za rješavanje problema pakiranja) za rješavanje problema rezanja gdje se očekuje više različitih duljina raspoloživog materijala se svodi na problem pridruživanja duljine komada materijala od kojeg se grupa elemenata rezati svakoj grupi materijala [3]. Taj problem je riješen proširivanjem zapisa grupe duljinom prikladnog komada materijala.

Generiranje početne populacije
Generiranje početne populacije je izvedeno slučajnim odabirom komada raspoloživog materijala, te zatim pridruživanjem u pripadnu grupu neostvarene zahtjeve za rezanjem korištenjem *First-Fit* algoritma (FF)[8]. Takav način generiranja daje rješenja koja su relativno dobra (tj. nisu jako loša) a i dalje su međusobno vrlo različita.

Selekcija
Za selekciju se koristi proporcionalna selekcija, što znači da je vjerojatnost da neka jedinka bude odabrana za reprodukciju jednaka njenoj relativnoj dobroti u odnosu na cijelu populaciju. Selekcijom se odabiru dvije jedinke roditelja od kojih će se generirati dvije jedinke nove generacije.

Križanje
Za križanje se koristi GGA operator križanja [7] koji je modifikacija standardnog križanja sa 2 točke prekida za GGA kromosome. Nova jedinka se generira iz odabranih roditelja na način da se iz prvog roditelja kopira dio grupa, zatim se iz drugog roditelja kopiraju sve grupe kojima se ne prelazi broj zahtjeva za rezanjem ni broj raspoloživih materijala, te se na kraju od preostalih nesvrstanih zahtjeva za rezanjem generiraju nove grupe koristeći algoritam FF.

Mutacija
Za mutaciju se koristi GGA operator mutacije [7] koji funkcionira na način da se određeni broj grupa obriše te se zatim iz oslobođenih elemenata generiraju nove grupe koristeći algoritam FF.

Funkcija dobrote

Za funkciju cijene se koristi sljedeća formula [3]:

$$C = \frac{1}{n}\left(\sum_{i=1}^{n}\frac{w_i}{l_i} + \frac{ws}{n}\right)$$

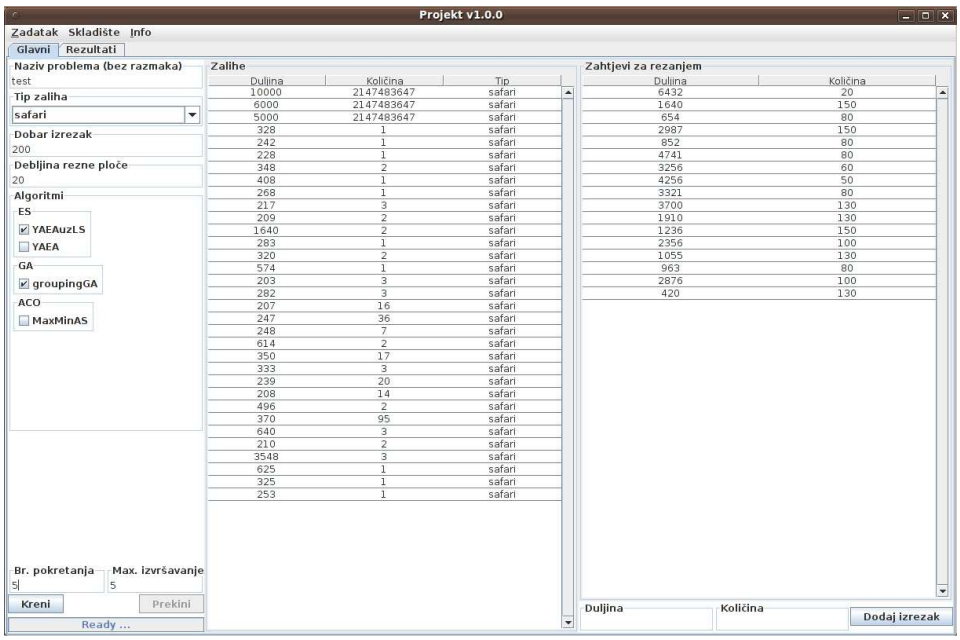
gdje je:

C = funkcija "cijene"
w_i = otpad nakon rezanja i-tog komada materijala
l_i = duljina i-tog komada materijala
ws = ukupan broj neiskorištenih komada materijala (komadi materijala kojima je ostatak veći od maksimalnog dopuštenog)
n = ukupan broj korištenih komada materijala

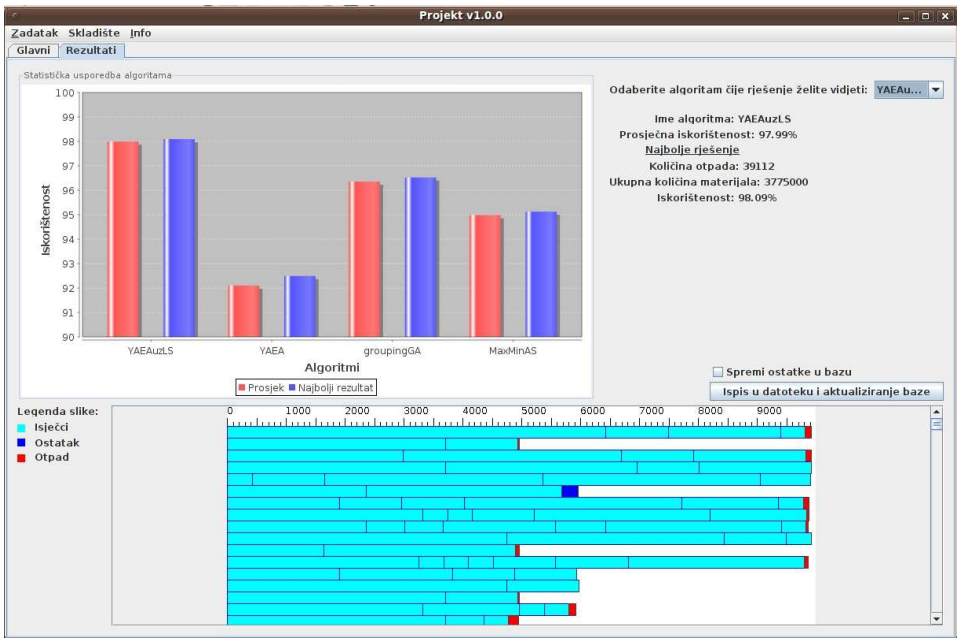
Dobrota pojedine jedinke se zatim računa kao $D_i = C_{max} - C_i$

1.4. Upute za korištenje

Za izvršavanje programa je potreban Perl interpreter i Java JRE. Postoje različite verzije programa za operacijske sustave Windows i Linux. Pokretanjem izvršne datoteke CSP-1D.jar otvara se prozor sa panelom za unos zadatka, te odabir algoritama s kojima se zadatak treba rješavati. Nakon pokretanja rješavanja zadatka, rezultati se mogu pratiti na drugom panelu. Rješenja se prikazuju sa 2 grafa: jedan za usporedbu rješenja svih algoritama, drugi za grafički prikaz najboljeg rješenja odabranog algoritma. Program odabrano rješenje ispisuje u .txt i .pdf datoteke.



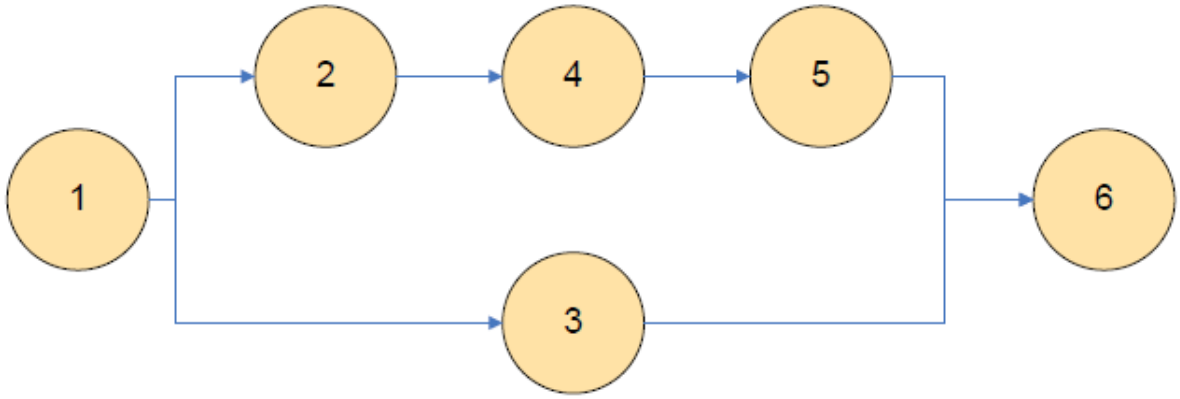
Slika 1. Prikaz prozora za unos zadatka



Slika 2. Prikaz prozora za praćenje rezultata

2. Problem raspoređivanja projekata s ograničenjem u sredstvima

Projekt je privremena aktivnost koja se provodi kako bi se napravio neki proizvod, usluga ili neki drugi rezultat. To je dakle svaka aktivnost koja ima točno određen početak i kraj, te definiran rezultat. Disciplina upravljanja projektima je primjena znanja, vještina i alata kako bi projektne aktivnosti dale očekivani rezultat. Voditelj projekta mora uspješno balansirati trokutom koji se sastoji od sredstava, rezultata projekta i vremenskog plana. Sredstva mogu biti: ljudi, strojevi, novac itd. Ako je cilj projekta izraditi računalni program, rezultati su svojstva proizvoda koji isporučujemo.[9]



Slika 2.1. Primjer projekta sa 6 aktivnosti [9]

Projekt s ograničenim sredstvima može se opisati na sljedeći način:

Neka postoji projekt s n+2 aktivnosti koje se moraju izvršiti prije nego se projekt završi. Definirajmo skup poslova J={0, 1, …, n, n+1} i skup sredstava K={1, 2, …, k}. Početna aktivnost koju označavamo s indeksom 0 i krajnja aktivnost koju označavamo sa indeksom n+1 su zapravo fiktivne aktivnosti čije je trajanje 0 vremenskih jedinica. Te dvije aktivnosti za izvođenje ne trebaju sredstva. Za sve ostale aktivnosti definirane su dvije skupine ograničenja:

- ograničenje slijeda aktivnosti: aktivnost ne može početi prije nego što završe sve aktivnosti koje joj prethode. Aktivnosti prethodnice definirane su usmjerenim grafom.
- izvođenje aktivnosti može započeti samo ako je dostupno dovoljno sredstava koja su potrebna da bi se aktivnost izvršavala.

Tijekom izvođenja, jednoj je aktivnosti j potrebno $r_{j,k}$ jedinica sredstva k∈K tijekom svakog vremenskog trenutka u kojem se izvodi. Trajanje aktivnosti je d_j, a čim ona započne ne smije se prekidati do završetka.[9]

Sredstvo k ograničeno je količinom R_k tijekom cijelog izvođenja projekta. Primjerice ako se naš projekt izvodi na nekom stroju, taj stroj može biti dostupan 8 sati dnevno i potrebno je aktivnosti projekta organizirati u skladu s tim ograničenjem. Svi ovdje spomenuti parametri $r_{j,k}$, d_j i R_k pozitivne su vrijednosti koje su utvrđene prije početka projekta. Kako je prethodno navedeno uvijek vrijedi $d_0=0$, $d_{n+1}=0$ i $r_{0,k}=r_{n+1,k}=0$ za svaki k∈K. Cilj optimizacije je pronaći projektni raspored za koji je trajanje projekta najkraće, a da su pritom zadovoljena sve gore spomenuta ograničenja.[9]

2.1. Optimiranje genetskim algoritmom uz OX operator križanja (Petar Čolić)

2.1.1. Primjena algoritma na zadani problem

Algoritam može raditi u jednom od dva glavna načina rada, generacijski i iteracijski.

Generacijski

Generira se određeni broj generacija. Svaka nova generacija nastaje od jedinki prošle generacije koje se međusobno križaju i mutiraju. Koristi se i elitizam, tj. čuvaju se najbolje jedinke iz prošle generacije. Da bi se elitizam mogao koristiti, potrebno je evaluirati svaku novu generaciju. Tako se mogu odrediti najbolje jedinke i njih se prebaci u novu generaciju.

Iteracijski

Stvori se početna populacija, te se obavlja turnirsko križanje. Nasumično se odaberu tri jedinke iz populacije te se križaju dvije bolje jedinke. Nastalo dijete prolazi mutaciju i zamijeni najlošiju odabranu jedinku. Jedna iteracija odgovara jednom križanju. Iteracijski način rada se može obaviti sa jednim od tri različita kriterija zaustavljanja. Algoritam se može ograničiti vremenski, brojem iteracija i pametno – praćenjem napretka u zadnjih nekoliko iteracija.

Vremensko ograničenje je najsporije jer zahtjeva čestu provjeru vremena što dodatno usporava program. Da bi se pronašlo optimalno ili dovoljno dobro rješenje, kod lakših test primjera potrebno je samo par sekundi dok je za teže primjere potrebno desetak sekundi.

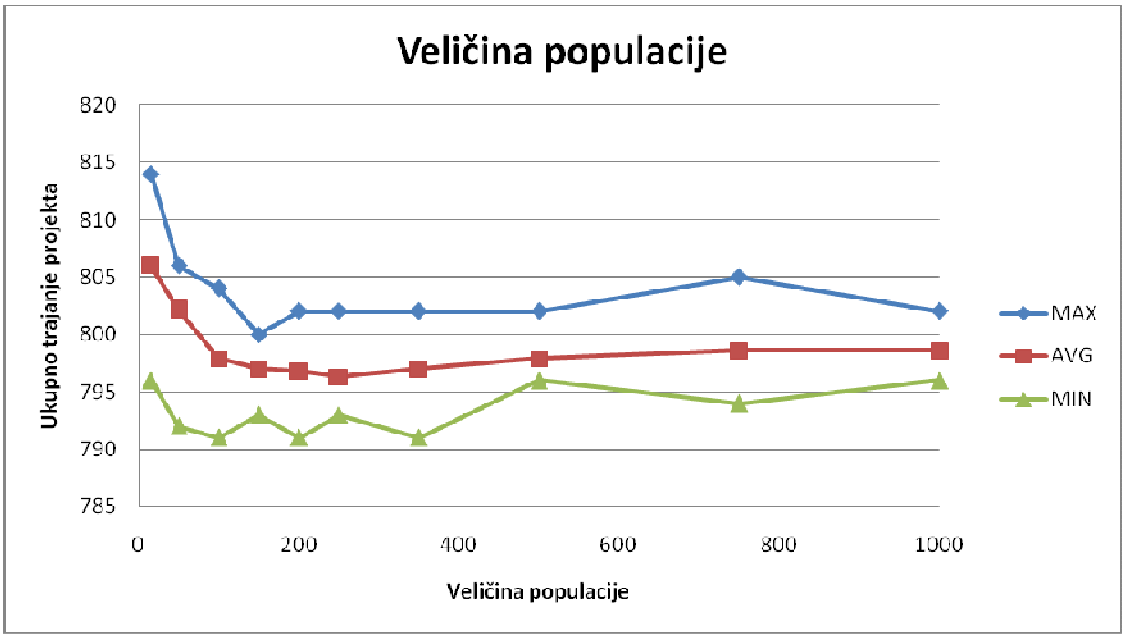
Na prosječnom računalu u desetak sekundi se može izvršiti oko 20-25k iteracija što je i više nego dovoljno za pronalazak dobrog rješenja. Dovoljno je uzeti 15k iteracija da se pokriju svi test primjeri.

Pametno zaustavljanje radi tako da prati napredak u određenom broju prošlih iteracija. Ako se u tom broju iteracija, najbolje rješenje ne promijeni, algoritam se zaustavlja. Rezultati su generalno loši ako se prati manje od 1000 iteracija unazad. Poboljšanja se vide do praćenja 6-7k iteracija, iznad toga poboljšanja su neznačajna. Ovim načinom teži test primjeri se prosječno zaustave na oko 9-10k, a lakši test primjeri na 4-6k iteracija.

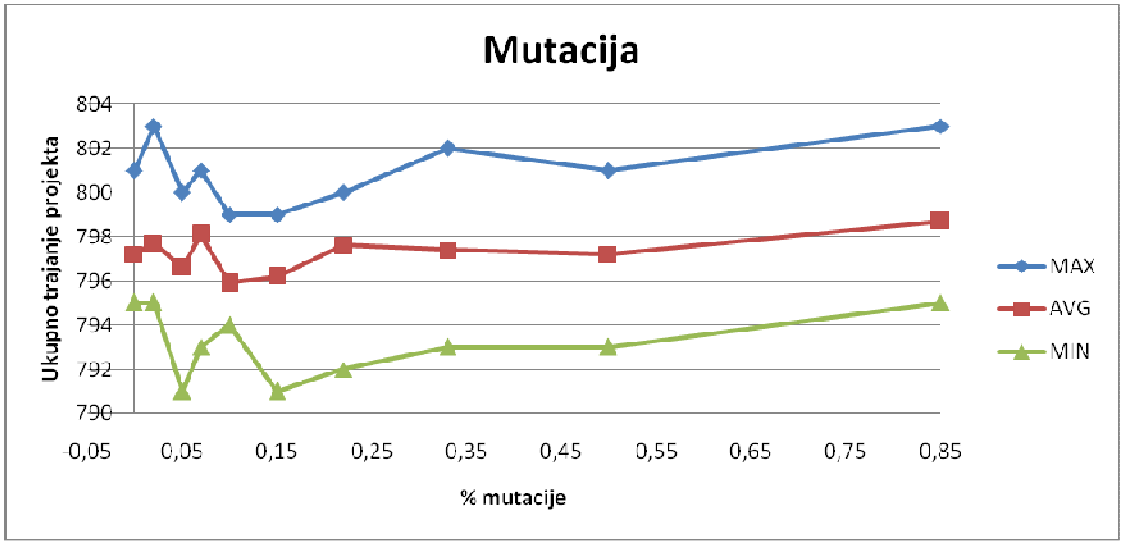
2.1.1. Analiza rezultata

Iteracijski način rada postiže bolje rezultate u kraćem vremenu, te se on koristio pri testiranju uz pametni način zaustavljanja. Kao kriterij zaustavljanja, pratilo se zadnjih 6000 iteracija. Različiti rezultati pri odabiru različitih vrijednosti parametara mogli su se vidjeti samo na težim test primjerima. Optimalna veličina početne populacije se kreće između 100 i 350 jedinki. Najbolji rezultati su dobiveni sa postotkom mutacije od 5% i 15%.

Algoritam na početku relativno brzo konvergira prema optimalnom rješenju, pogotovo ako se uzme mala početna populacija. Kasnije su pomaci sve manji i manji. Nakon 30-60 sekundi, pomaka gotovo i nema, jer će algoritam ili naći optimalno rješenje ili biti jako blizu.



Graf 2.1.1. Ovisnost trajanja projekta o veličini populacije



Graf 2.1.2. Ovisnost trajanja projekta o postotku mutacije

2.1.3. Upute za korištenje

Izvorni jezik: Java

Program je preveden u .jar format te se može pokrenuti iz komandne linije. Nikakvi argumenti se ne zadaju pri pokretanju programa, nego se unose postepeno, kada program to zatraži. Korisnik odabire način rada algoritma te unosi parametre i zadaje ulaznu datoteku. Ispis programa također je vidljiv na komandnoj liniji. Na računalu je potrebno imati instaliranu Javu.

2.2. Optimiranje algoritmom kolonije mrava (Karlo Knežević)

2.2.1. Primjena algoritma na zadani problem

Za rješavanje zadanog problema koristi se algoritam kolonije mrava. Algoritam kolonije mrava je metaheuristički algoritam koji imitacijom ponašanja mrava u prirodi vrlo efikasno pronalazi rješenja za određene probleme. Pri pokretanju algoritma najprije se postavljaju inicijalni feromonski tragovi, izračunavaju se heurističke vrijednosti te se stvaraju i postavljaju mravi u aktivnosti. Funkcija postavljanja mrava u aktivnosti je modulo funkcija koja mrave, logički, kružno postavlja u čvorove aktivnosti započevši od aktivnosti s indeksom 1 pa na više. Prilikom inicijaliziranja mrava, inicijaliziraju se i njihove tabu liste – tablica u kojoj piše koje aktivnosti do sada nisu posjetili.

Zatim počinju iteracije kretanja mrava. Nakon *n* iteracija svi mravi imaju pune tabu liste te se postavlja trag feromona samo *najboljeg* mrava. Za izračun najboljeg mrava koristi se funkcija koja izračunava, na temelju vektora posjećenih aktivnosti, koliko bi trajalo izvođenje projekta. Mrav, ili mravi, koji sadrži(e) vektor koji rezultira najkraćim trajanjem projekta, postavlja jačinu feromonskog traga koja je jednaka recipročnoj vrijednosti trajanja projekta određena prioritetnim vektorom.

Trajanje algoritma traje tako dugo dok se ne iziteriraju svi mravi, po svim generacijama, ili, ukoliko je korisnik odabrao vremensku kontrolu toka izvođenja, dok ne istekne vremensko ograničenje trajanja algoritma. Algoritam kolonije mrava ostvaren je kroz sljedeće funkcije:

```
dok (brojac_ciklusa < MAX_broj_ciklusa){
        brojac_ciklusa++;
        Odradi_Ciklus();
        Provjeri_Najbolju_Rutu();
        Ispari_Trag();
        Nadodaj_Trag();
        Resetiraj_Mrave();
}
```

Slika 2.1. Funkcije algoritma kolonije mrava

Za izračun trajanja projekta na temelju prioritetnog vektora koristi se *fitness* funkcija. [11]

2.2.2. Analiza rezultata

U analizi su ispitivana 4 faktora:

- *alfa* parametar
- *beta* parametar
- *ro* parametar
- *brojnost mrava*

Kao inicijalne vrijednosti prilikom ispitivanja promjene parametara bile su:

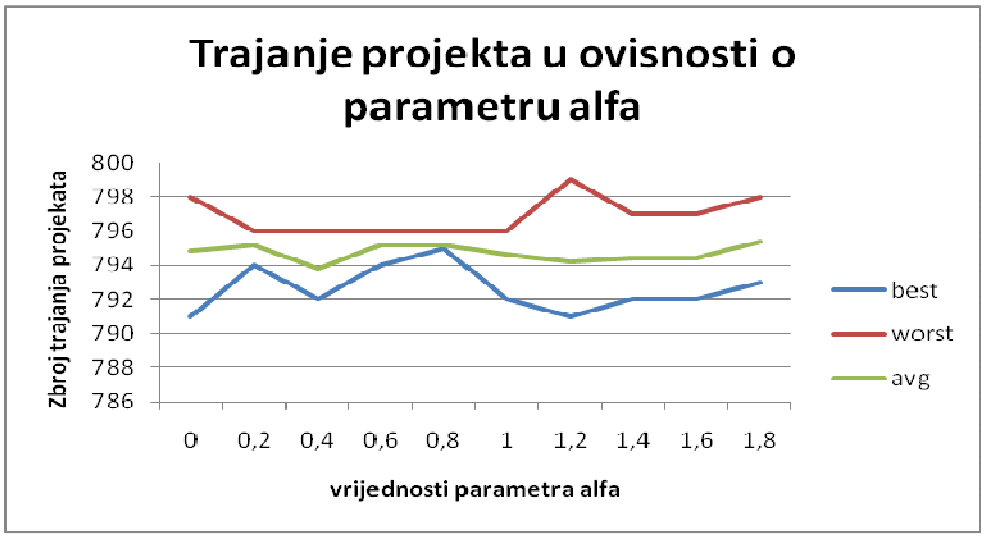
alfa	beta	tau	ro	Broj mrava	Trajanje izvođenja
1.3	1.0	0.7	0.5	n	600 s

Tablica 1. Inicijalne vrijednosti parametara

Te vrijednosti [Tablica 1.] dobivene su eksperimentalno pretraživanjem prostora najboljih rješenja. Broj mrava bio je jednak broju aktivnosti u projektu.
Procesor na kojem se vršilo ispitivanje je Athlon 64,dvojezgreni procesor tvrtke AMD, takta frekvencije 2.7GHz; RAM 2GB.
Izvođenje programa omeđeno je vremenskim izvođenjem da bi se egzaktno moglo odrediti izvođenje programa. Također, samo je mrav s najboljim rješenjem osvježavao feromonski trag.

Alfa parametar

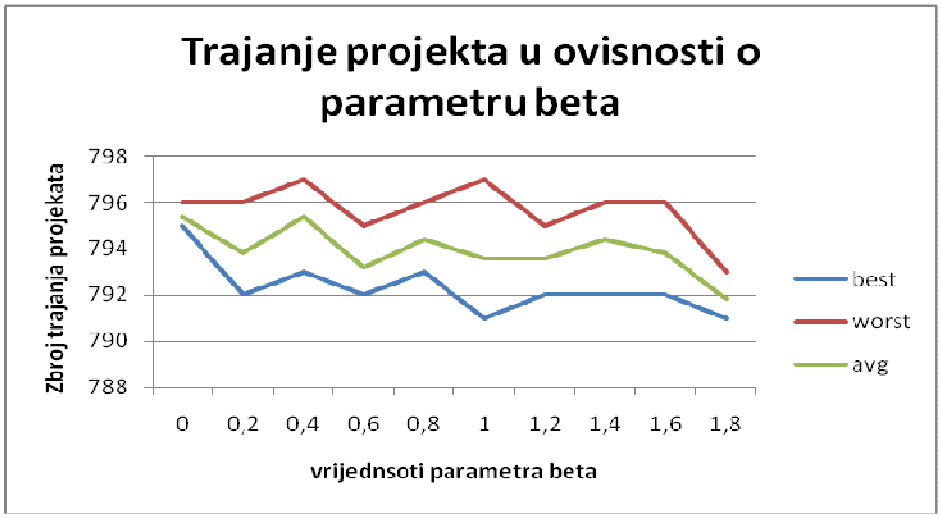
Alfa parametar određuje koliki će utjecaj, prilikom odluke o odlasku u sljedeći čvor aktivnosti, imati feromonski trag. Iskustva govore da bi alfa parametar trebao biti manji od *beta* parametra (heurističkog parametra), međutim u ovim ispitivanjima pokazalo se da se bolji rezultati dobivaju ukoliko je *alfa* parametar veći od *beta*. Rezultati ispitivanja su očekivani: što je *alfa* veća, rezultati postaju sve lošiji. Takvo ponašanje opravdava se činjenicom da ukoliko najbolji mrav prođe lošom rutom, većina mrava započinje slijediti tog mrava I događa se konvergencija u suboptimalno rješenje. Na grafu *2.2.1.* prikazane su dobiveni rezultati.



Graf 2.2.1. Trajanje projekta u ovisnosti o parametru alfa

Beta parametar

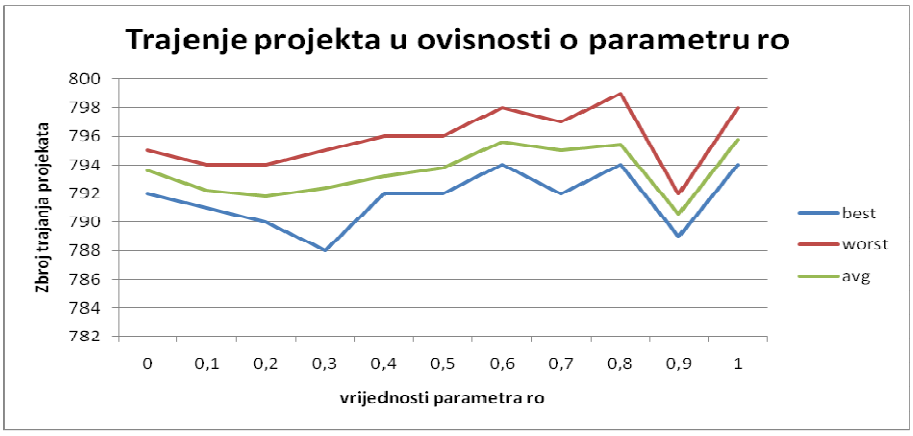
Beta parametar određuje koliki će utjecaj imati heuristička informacija prilikom odluke o odlasku u sljedeći čvor aktivnosti. U rješavanju problema, autor je kao heurističku informaciju uzeo recipročnu vrijednost zbroja trajanja aktivnosti i svih resursa koje ta aktivnost zauzima tijekom izvođenja. Upravo činjenica odabira heurističke informacije ovisi o tome hoće li *beta* parametar biti veći od *alfa*, ili obrnuto. Rezultati ispitivanja, također, su očekivani. Što je heuristička informacija o odluci o odlasku u sljedeću aktivnost veća, rezultati su bolji. Na grafu *2.2.2.* prikazani su dobiveni rezultati.



Graf 2.2.2. Trajanje projekta u ovisnosti o parametru beta

Ro parametar

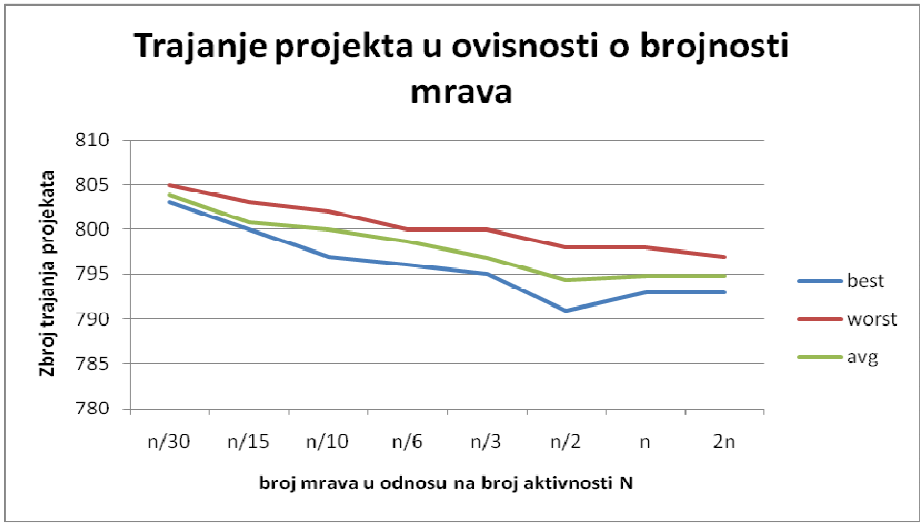
Ro parametar je broj koji određuje brzinu isparavanja te se on nalazi u intervalu [0,1>. Vrijednost parametra *ro* ovisi o vrijednostima *alfa* i *beta*. U ovisnosti o dobroti tih parametara, eksperimentalno se, pretraživanjem prostora rješenja, pronađe optimalan iznos. Na grafu *2.2.3.* prikazani su dobiveni rezultati.



Graf 2.2.3. Trajanje projekta u ovisnosti o parametru ro

Utjecaj brojnosti mrava

Broj mrava (*N*) koji pretražuju prostor rješenja je važan jer odlučuje o tome koliko će dugo trajati izvođenje programa i koliko dobra rješenja će mravi pronaći. Iz iskustva je poznato da je optimalan broj mrava jednak broju čvorova grafa kojim prolaze. Međutim, ispitivanja su pokazala zanimljiv rezultat: optimalan broj mrava za rješavanje ovog problema je *N/2*, gdje je *N* broj aktivnosti nekog projekta. Takav rezultat može se opravdati činjenicom da već broj mrava, ukoliko parametri *alfa*, *beta* i *ro* nisu optimalni, može dovesti do suboptimalnog rješenja. Na grafu *2.2.4.* prikazani su dobiveni rezultati.



Graf 2.2.4. Trajanje projekta u ovisnosti o brojnosti mrava

2.2.3. Upute za korištenje

Izvorni jezik: C++

Pokrenuti aplikaciju i slijediti upute. Korisnik prvo odabire način rada programa. Korisniku se nude 2 opcije:

- Jednovarijabilni način rada
- Viševarijabilni način rada

U jednovarijabilnom načinu rada korisnik može odabrati da mu se inicijalno postave unaprijed definirane, optimalne, vrijednosti parametara ili može vrijednosti parametara upisivati samostalno. Nakon odabira vrijednosti parametara, korisnik upisuje broj mjerenja za učitanu datoteku. Nakon odabira broja mjerenja za učitanu datoteku, korisniku se nudi mogućnost vremenskog navođenja (vremenskog ograničenja trajanja testiranja jednog mjeenja učitane datoteke).

U viševarijabilnom načinu rada korisnik sam odabire interval ispitivanja neke varijable. Interval se odabire upisivanjem početne i konačne vrijednosti intervala te korak, odnosno vrijednost za koju će povećavati iznos varijable. Ukoliko korisnik ne želi ispitivati interval neke varijable, već želi imati fiksnu vrijednost, tada se za početnu i konačnu vrijednost upisuje ista vrijednost. U viševarijabilnom načinu rada ne postoji vremensko ograničenje izvođenja programa, nego program se izvodi dok se ne ispitaju sve vrijednosti intervala neke varijable.

2.3. Optimiranje CLONALG algoritmom (Ivo Majić)

2.3.1. Primjena algoritma na zadani problem

Algoritam CLONALG spada u skupinu algoritama AIS (Artificial Immune System) koji oponašaju rad imunološkog sustava kod sisavaca. Za optimiranje ovog problema je jedna jedinka (antitijelo) prikazana kao vektor prioriteta koji nam govori u kojem redoslijedu se aktivnosti trebaju izvršiti. Algoritam prvo stvara inicijalnu populaciju jedinki (prikazanih kao klasa antitijelo koja sadržava vektorski zapis te podatak o afinitetu) određenu brojem d te zatim ide kroz generacije tako da u svakoj klonira jedinke proporcionalno njihovom afinitetu te ih zatim mutira obrnuto proporcionalno istome. U sljedeću generaciju prelazi n najboljih jedinki iz te populacije te im se dodaje d novostvorenih jedinki kako bi svaka generacija imala određenu količinu svježeg genetskog materijala. [10] Postupci kloniranja i mutacije su opisani u nastavku.

Kloniranje

Funkcija kloniranja svaku jedinku u populaciji klonira proporcionalno njezinom afinitetu (bolje jedinke s kloniraju više puta, lošije jedinke manje) pri čemu broj klonova pojedine jedinke x određuje formula:

$$brojKlonov(x) = 1 + (velicinaPopulacije - rangAntitijela(x)) * beta$$

gdje je beta faktor vrijednost koja određuje učestalost kloniranja pojedine jedinke.

Mutacija

Funkcija mutacije svaku jedinku u kloniranoj populaciji mutira obrnuto proporcionalno njezinom afinitetu (bolje jedinke se mutiraju manje od lošijih jedinki) pri čemu učestalost mutacije jedinke x određuje formula:

$$ucestalostMutacije(x) = 1 + (rangAntitijela(x) * p)$$

gdje je p proizvoljna vrijednost od <0, 1].

Izračun afiniteta

U inicijalnoj verziji programa funkcija koja računa afinitet jedinki (antitijela) je bila pisana u Pythonu. Kako je to zbog složenosti izračuna afiniteta u Pythonu bilo sporije od očekivanog napisana je nova funkcija u programskom jeziku C (modul) koja se može pozivat direktno iz Pythona. To je donijelo značajno ubrzanje u izračunu afiniteta te cjelokupnim performansama programa što se vidi u tablici 2.3.1.

Python	C	Ubrzanje
35.6 sekundi	2.2 sekunde	16,2 puta

Tablica 2.3.1. Usporedba brzine izračuna afiniteta 5000 jedinki

2.3.2. Analiza rezultata

Analizirana su 3 parametra algoritma:

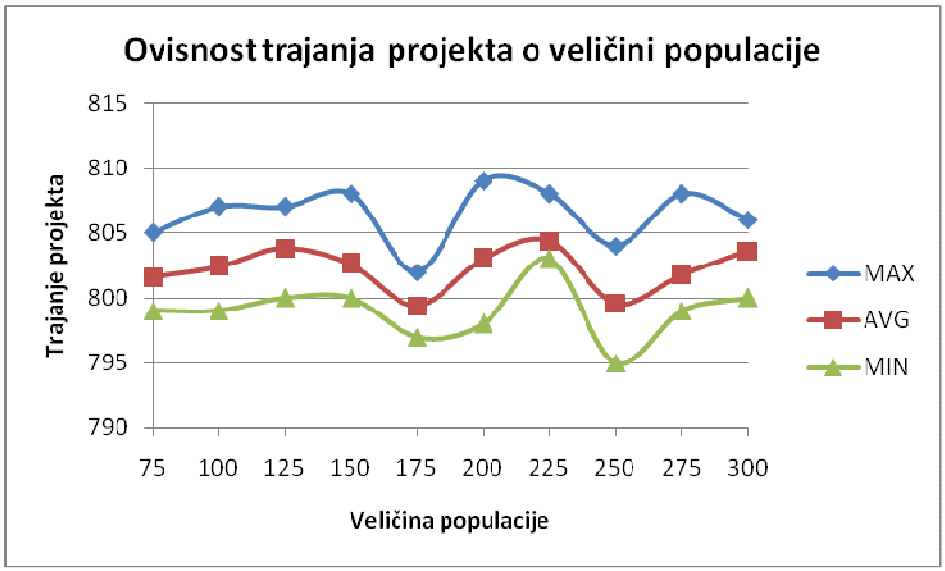
- veličina populacije
- veličina inicijalne populacije
- beta faktor

Veličina populacije

Veličina populacije određuje koliko će najboljih jedinki u svakoj generaciji biti uzeto te zajedno sa novonastalim jedinkama kako bi dobili svježi genetski materijal (određeni parametrom d) biti prebačeno u sljedeću generaciju. Ovdje vidimo da algoritam najbolje rezultate postiže sa vrijednostima 175 te 250 gdje se onda primjećuje da nema daljnjih poboljšanja. Inicijalne vrijednosti algoritma se vide u tablici 2.3.2. dok su rezultati mjerenja prikazani u grafu 2.3.1.

Veličina inicijalne populacije	Beta faktor	Trajanje
50	0.5	5 min

Tablica 2.3.2. Inicijalne vrijednosti algoritma



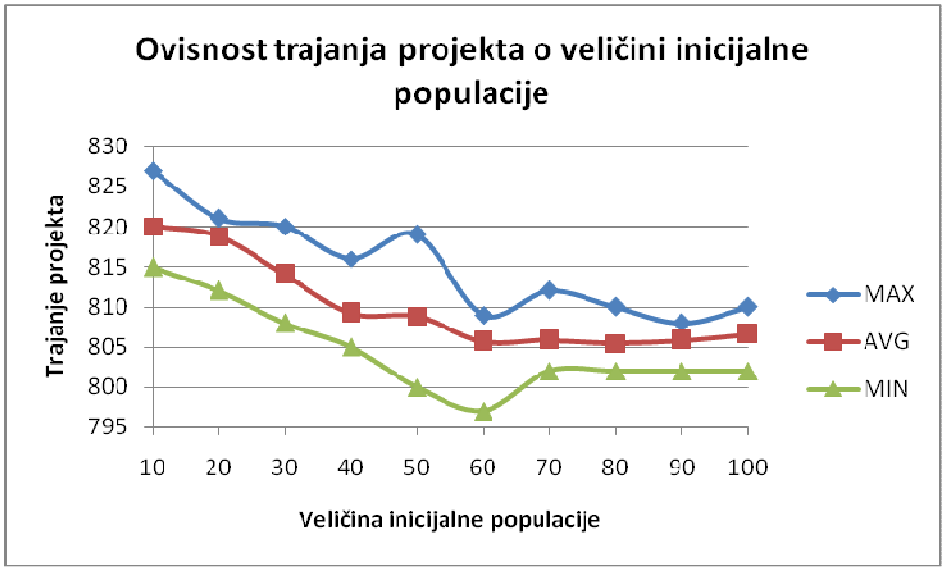
Graf 2.3.1. Ovisnost trajanja projekta o veličini populacije

Veličina inicijalne populacije

Veličina inicijalne populacije je vrijednost koja određuje koliko će slučajnih jedinki biti stvoreno na početku algoritma te ujedinio određuje koliko će slučajnih jedinki na kraju svake generacije biti dodano u populaciju kao svježi genetski materijal. Inicijalne vrijednosti algoritma se vide u tablici 2.3.3. dok su rezultati mjerenja prikazani u grafu 2.3.2. Rezultat pokazuju da algoritam postiže sve bolje rezultate povećanjem inicijalne populacije prema vrijednosti 60 nakon čega za sve daljnje vrijednosti ne primjećuje poboljšanje rezultata.

Veličina populacije	Beta faktor	Trajanje
175	0.5	5 min

Tablica 2.3.3. Inicijalne vrijednosti algoritma



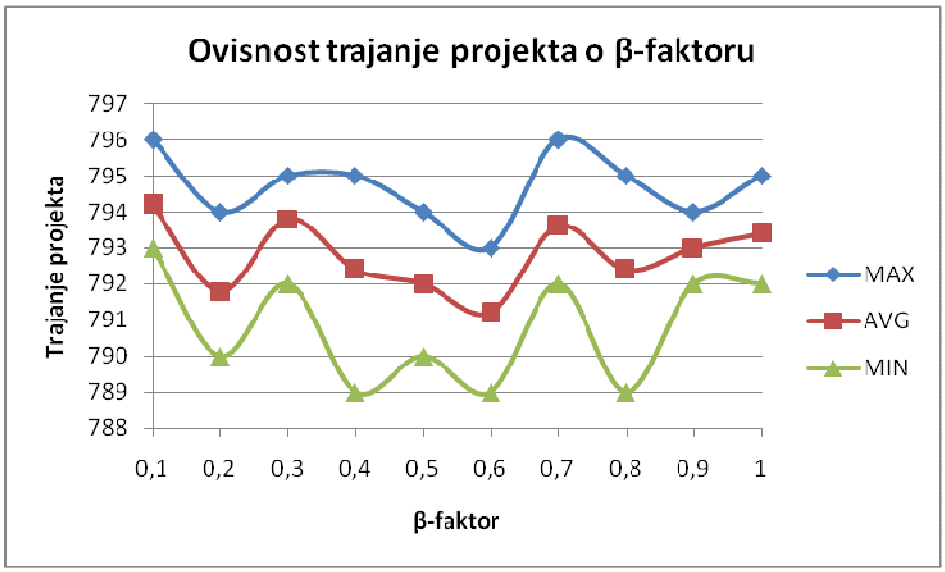
Graf 2.3.2. Ovisnost trajanja projekta o veličini inicijalne populacije

Beta faktor

Beta faktor određuje broj klonova koje će jedinka imati sukladno njezinom afinitetu, pri čemu će bolje jedinke imati više klonova a lošije jedinke manje klonova. Inicijalne vrijednosti algoritma se vide u tablici 2.3.4. dok su rezultati mjerenja prikazani u grafu 2.3.3. Iz rezultata zaključujemo da algoritam najbolje ukupne vrijednosti daje za vrijednost beta faktora 0.6 te ja ta vrijednost korištena prilikom zajedničkih mjerenja.

Veličina populacije	Veličina inicijalne populacije	Trajanje
175	50	10 min

Tablica 2.3.4. Inicijalne vrijednosti algoritma



Graf 2.3.3. Ovisnost trajanja projekta o beta faktoru

2.3.3. Upute za korištenje

Izvorni jezik: Python

Za pokretanje aplikacije treba imati instaliran Python interpreter. Aplikacija se pokreće iz komandne linije uz odgovarajuće argumente kao:

python projekt_rcpsp.py test_datoteka broj_generacija velicina_pop init_pop beta

pri čemu je:

test_datoteka – putanja do .sm datoteke za koju tražimo optimum

broj_generacija – koliko generacija će algoritam proći

velicina_pop – veličina populacije koja će se koristiti u algoritmu

init_pop – veličina inicijalne populacije koja se stvara na početku algoritma

beta – beta faktor, trebao bi biti realan broj između <0,1]

2.4. Optimiranje genetskim algoritmom uz PMX operator križanja (Luka Marasović)

2.4.1. Primjena algoritma na zadani problem

Reprezentacija genoma

Genom je prikazan pomoću jednodimenzionalne liste brojeva od kojih svaki broj predstavlja prioritet za i-tu aktivnost (i je broj lokacije u jednodimenzionalnom nizu). U programu, genom se nalazi unutar klase Jedinka, a ona se pak nalazi u listi objekata Jedinke.

Selekcija

Selekcija je ostvarena nasumičnim odabirom triju jedinki, od kojih dvije bolje postaju roditelji novoj, a treća se odbacuje. Ovim putem su najbolje jedinke uvijek očuvane, a postoji mogućnost da se prekopiraju u cijelosti čime one dobivaju prednost u nasumičnom odabiru.

Križanje

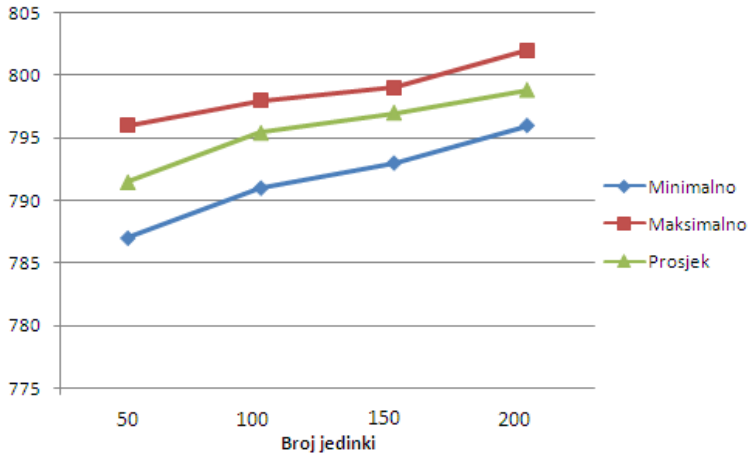
U ovoj inačici programa križanje se izvodi svaki put, iako je dodana mogućnost smanjivanja parametra na određeni postotak. Ako je vjerojatnost križanja manja od 100, te u slučaju da se križanje ne provodi, na najlošijoj od tri jedinke se izvodi operator mutacije. Ovo je dodano za mogućnost implementacije naglaska na mutaciji u kasnijoj fazi algoritma kada su jedinke poprilično ujednačene po funkciji dobrote. Ovdje implementirano križanje je Partially Matched Crossover (PMX).

Mutacija

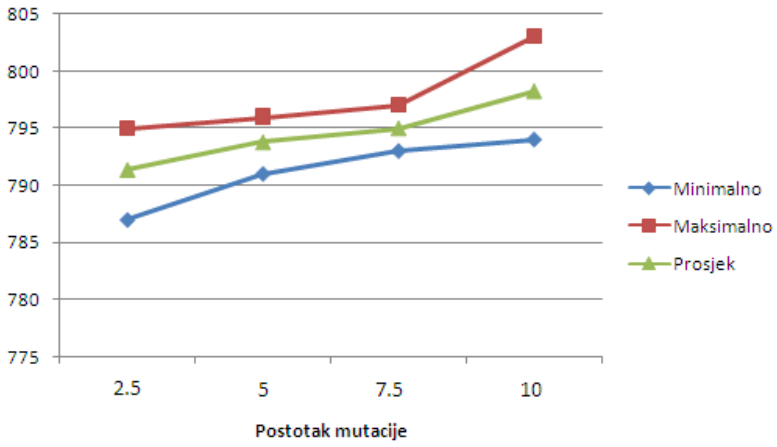
Kako je genom prikazan kao permutirani niz, nije moguće odabrati bilo koju vrijednost nad genom koji želimo mutirati, stoga se izvodi zamjena lokacija dva gena unutar genoma. Vjerojatnost mutacije je moguće postaviti na vrijednosti od 0 do 1000, tj. u promilima. Ako trenutni gen mutira, odabire se nasumični gen s kojim ga zamjenjujemo.

2.4.2. Analiza rezultata

Za testiranje GA nad zadanim problemom, korištene su različite vrijednosti početne populacije i mutacije. Testiranje svakog seta testnih postavki je ponavljano 10 puta, sa ograničenjem 15 minuta. Kako je testna platforma sadržavala dvojezgreni procesor, paralelno su se izvodila 2 programa, što se po testiranju ispostavilo da malo utječe na brzinu izvođenja. Pad performansi je negdje oko 10%.



Graf 2.4.1. Ovisnost trajanja projekta o broju jedinki



Graf 2.4.2. Ovisnost trajanja projekta o postotku mutacije

Iz danih rezultata se da zaključiti da GA najbrže konvergira (gledano vremenski) za mali broj početne populacije, u ovom slučaju 50, i za male vrijednosti mutacije. Moguće je dodatno istražiti područje oko ovdje dobivenih najoptimalnijih vrijednosti za bržu konvergenciju algoritma.

2.4.3. Upute za korištenje

Izvorni jezik: C#

Napravljeno je grafičko sučelje pomoću Windows Form-a unutar Visual Studia 2008, uz .NET Framework 3.5 te C# programski jezik. Sučelje omogućava odabir sljedećih postavki:

- naziv ulazne datoteke
- broj jedinki
- vjerojatnost križanja
- vjerojatnost mutacije
- vrijeme izvršavanja algoritma
- broj izračuna dobrote jedinke
- broj ponavljanja
- učitavanje postavki

Nakon svih obavljenih zadataka, imamo ispis najboljih jedinki (ako je bilo zadano više ponavljanja), broj izračuna dobrote i prosječna dobrota jedinki iz zadnje iteracije.
Da bi se uspješno pokrenuo, program zahtjeva računalo na kojem je instaliran .NET Framework 3.5. Program se sastoji od jedne izvršne datoteke, nazvane "RCPSP-Projekt.exe". Nakon što pokrenemo program, prikazuje nam se sučelje. Putem izbornik opcije "Postavke" možemo dohvatiti i definirati sve postavke rada algoritma. Prilikom prelaska strelice miša nad nekom od postavki, dobivamo novi izbornik u kojem je omogućena izmjena parametra. Ovo je ostvareno upotrebom .NET kontrole TextBox. Svaka izmjena parametra ostaje zapisana unutar TextBox kontrole. Nakon odabranih postavki za rad algoritma, potrebno je kliknuti "Učitaj podatke" unutar padajućeg izbornika "Postavke". Program će dati obavijest jesu li podatci uspješno učitani, ako ne, izbaciva poruku o pogrešci. Ako su podatci uspješno učitani, klikom na kontrolu "GENERIRAJ" algoritam započinje rad. Kada se dvije kontrole Progress Bar-a ispune do kraja, algoritam je završio posao. Prvi odozgo Progress Bar prikazuje izvršavanje jednog zadatka, a drugi ponavljanje, pošto možemo odabrati veći broj ponavljanja algoritma. Nakon svih obavljenih zadataka,

imamo ispis najboljih jedinki (ako je bilo zadano više ponavljanja), broj izračuna dobrote i prosječna dobrota jedinki iz zadnje iteracije.

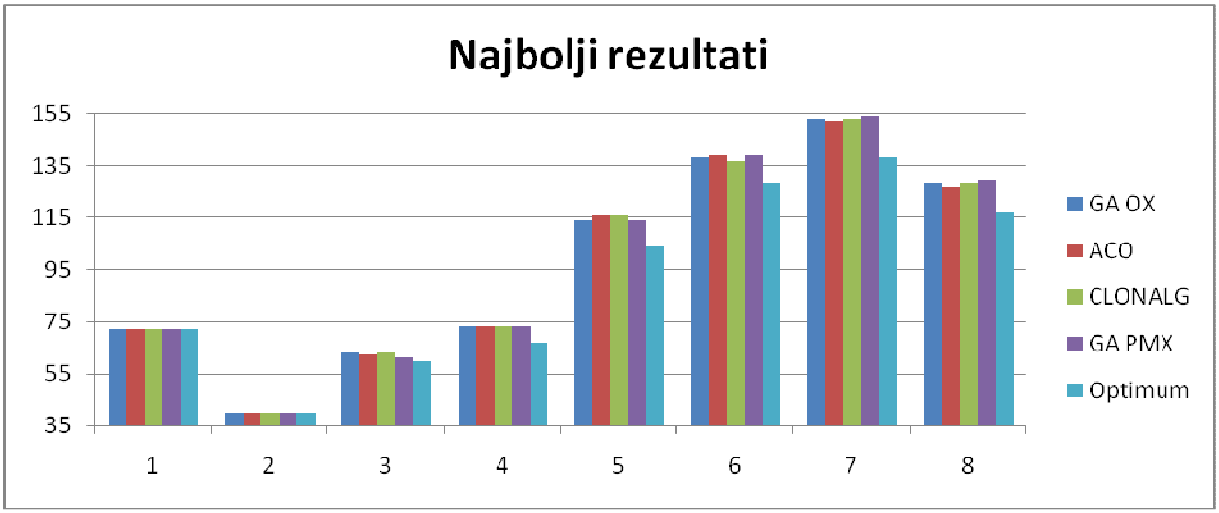
Napomene

- Progress Bar-ovi u nekim situacijama neće predstavljati stvarno stanje ili iteraciju algoritma, ovo je vjerojatno zbog velikih zahtjeva nad procesorom koji ne stigne osvježavati sučelje. Također se u zaglavlju programa pored naziva može pojaviti tekst "(Not responding)". Ovaj artefakt je logičan, jer je program zauzet, i nije u mogućnosti registrirati nove događaje. Moguće rješenje je putem dretvi, ali dodatna implementacija je otežana zbog drugačijeg pristupa programiranju u početnim fazama izrade.
- Ako je pokrenuto više instanci programa, moguće je da će samo jedan od njih osvježavati svoje stvarno stanje.

2.5. Zajednički rezultati

Najbolji rezultati

Na grafu 2.5.2. su prikazani najbolji rezultati koji su algoritmi ikad našli za pojedinu test datoteku.

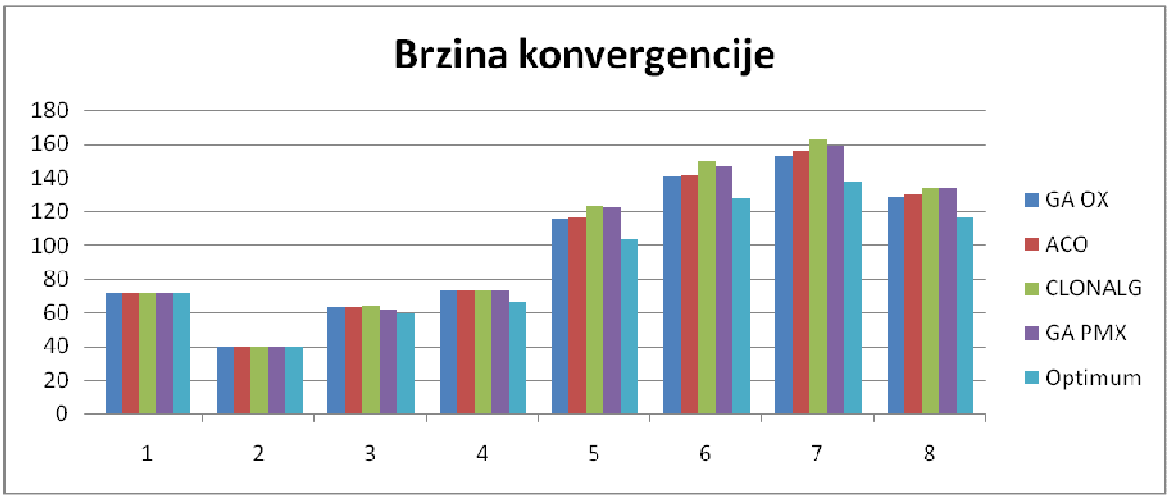


Graf 2.5.1. Najbolji rezultati

	1	2	3	4	5	6	7	8
GA OX	72	40	63	73	114	138	153	128
ACO	72	40	62	73	116	139	152	127
CLONALG	72	40	63	73	116	137	153	128
GA PMX	72	40	61	73	114	139	154	129
Optimum	72	40	60	67	104	128	138	117

Brzina konvergencije

U ovom mjerenju se pokušalo doznati koji algoritam nalazi bolji rezultat u nekom vremenski zadanom ograničenju. Za ovo mjerenje uzeto je ograničenje od 2 min (120 sekundi) te su rezultati vidljivi u grafu 2.5.1.



Graf 2.5.2. Najbolji rezultati dobiveni u 2 min

3. Literatura

[1] H. Dyckhoff, “A typology of cutting and packing problems”, European Journal of Operational Research, vol. 44, pp. 145-159, 1990.

[2] John Levine, Frederick Ducatelle, Ant Colony Optimization and Local Search for Bin Packing and Cutting Stock Problems, Journal of the Operational Research Society 55, 705-716, July 2004.

[3] Thomas Stützle, Holger Hoos, MAX-MIN ant system, Future Generation Computer Systems, 16(8):889–914, 2000.

[4] Monika Čeri, Iva Malović, Evolucijske strategije, http://www.zemris.fer.hr/~golub/ga/studenti/projekt2007/es.html

[5] John Levine, Frederick Ducatelle, Ant Colony Optimization and Local Search for Bin Packing and Cutting Stock Problems, Journal of the Operational Research Society 55, 705-716, July 2004.

[6] Janne Karelahti, Solving the cutting stock problem in the steel industry, Master's thesis, Helsinki University of Technology

[7] R. Hinterding and L. Khan, “Genetic algorithms for cutting stock problems: with and without contiguity,”, Progress in Evolutionary Computation (X. Yao, ed.), vol. 956 of Lecture Notes in Artificial Intelligence, Berlin, pp. 166-186, Springer, 1995.

[8] Falkenauer, E.A, & Delchambre, A., “A Genetic Algorithm for Bin Packing and Line Balancing”, Proceedings of 1992 IEEE International Conference on Robotics and Automation(RA92), pp. 1186-1193, Nice 1992.

[9] Frankola, T., diplomski rad br. 1626, Rješavanje problema raspoređivanja aktivnosti projekata evolucijskim algoritmima, Zagreb, Fakultet elektrotehnike i računarstva, listopad, 2006.

[10] Marko Čupić, Prirodom inspirirani optimizacijski algoritmi, 2008.

[11] Jakobović, D., Frankola, T., Golub, M., Evolutionary Algorithms for the Resource Constrained Scheduling Problem, Cavtat, Int. Conf. On Information Technology Interfaces, lipanj, 2008.

[12] Priyanto, Adiwijaya, Maharani, Implementation of ant colony optimization algorithm on the project resource scheduling problem, Faculty of Informatics, Institute of Technology Telkom

[13] Liang, Chen, Kao, Chyu, An ant colony approach to resource-constrained project scheduling problems, Department of Industrial Engineering and Management, Yuan Ze University