SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1626

RJEŠAVANJE PROBLEMA RASPOREĐIVANJA AKTIVNOSTI PROJEKATA EVOLUCIJSKIM ALGORITMIMA

Toni Frankola

Zagreb, listopad 2006.

Sažetak

U ovom radu su istražene moguće primjene evolucijskih algoritama (genetski algoritam i genetsko programiranje) za optimiranje projekata s ograničenim sredstvima. Raspoređivanje aktivnosti takvih projekata je NP težak problem. Opisani su različiti pristupi rješavanju problema korištenjem genetskih algoritmima i genetskog programiranja, a konačni rezultati su uspoređeni s poznatim heurističkim rješenjima.

Abstract

This diploma thesis investigates possible use of evolution algorithms (genetic algorithm and genetic programming) for optimization of resource constrained projects. Scheduling activities for resource constrained projects is NP hard problem. The work shows different approaches and results achieved with genetic algorithms and genetic programming. Final results are compared with existing heuristic solutions.

Sadržaj

1. Uvod

U ovom su diplomskom radu opisani evolucijski algoritmi i njihova primjena u području upravljanja projektima, točnije za optimiranje projekata s ograničenim sredstvima. Danas se mnoštvo poslova organizira u obliku različitih projekata. Kupci i dobavljači sudjeluju u projektu kako bi ostvarili različite, najčešće materijalne ciljeve. U toj „igri“ suprotnosti kupci najčešće žele za svoj novac dobiti što je više moguće, a dobavljači žele isporučiti projekt s minimalnim ulaganjima, u što kraćem vremenu i s maksimalnom zaradom.

Upravo su razlike između želja kupaca i dobavljača često uzrok kašnjenja i propadanja projekata. Brojni komercijalni alati koji su dostupni na tržištu danas omogućuju bolju kontrolu tijeka pojedinog projekta. Ti alati najčešće rješavaju probleme iz dviju velikih skupina:

a. Planiranje i raspoređivanje projektnih zadataka
b. Komunikacija među članovima tima

Poseban problem u upravljanju projektima predstavljaju projekti iz ICT industrije. Ova je industrija relativno „mlada“ i nije dovoljno sazrela da bi postojalo dovoljno iskustva za upravljanje i privođenje kraju složenih projekata. ICT tehnologije i tržišta se mijenjaju brže nego u bilo kojoj drugoj industrijskoj grani. Iz tih razloga svjedoci smo brojnih problema upravo s ICT projektima.

Projekte s ograničenim sredstvima susrećemo svaki dan. Većina projekata nema dovoljan broj sredstava na raspolaganju. Najčešći je problem nedostatak educirane i kvalitetne radne snage, ali također nedostatna novčana sredstva, strojevi i drugo. U ovom radu je opisano kako problemima planiranja i raspoređivanja doskočiti naprednom znanstvenom metodom, koja će bolje od čovjeka pronaći ono rješenje za koje je vrijeme trajanja projekta najkraće.

Za postavljanje modela projekta s ograničenim sredstvima trebaju unaprijed biti poznata trajanja izvođenja svake aktivnosti i točno trajanje zauzeća pojedinog resursa. Stoga je ovakav model nepraktičan za projekte koji uključuju ljudsku radnu snagu jer je tako precizne procjene u velikom broju slučajeva gotovo nemoguće dati. Međutim, ovakvi modeli se mogu izvrsno prilagoditi onim projektima na kojima a većinu aktivnosti obavljaju strojevi za koje se precizno zna vrijeme izvođenja pojedine projektne aktivnosti. Postavljanjem preciznog modela smanjuje se i prostor mogućih rješenja koja evolucijskom algoritmu omogućuju da pronađe optimalno rješenje za zadanu funkciju cilja.

2. Genetski algoritmi

Genetski algoritam je heuristička metoda optimiranja koja je zasnovana na evoluciji živih bića. Genetski algoritmi predloženi su od strane Johna H. Hollanda još u ranim sedamdesetim godinama prošlog stoljeća. Osnovna je ideja ovih algoritama kopiranje rada prirodnog evolucijskog procesa.

Optimalna rješenja zadanih problema nije moguće pronaći klasičnim metodama pretraživanja jer je prostor rješenja problema najčešće prevelik da bi računalo moglo pretražiti sva rješenja u nekom razumnom vremenu.

Ako evoluciju promatramo kao „algoritam“ koji traži optimalno rješenje (cilj je stvoriti što naprednije organizam), možemo reći da je prirodna evolucija vrlo uspješan „algoritam“. Nakon 4,5 milijardi godina izvođenja taj algoritam i dalje producira kvalitetna rješenja. Živa bića danas rješavaju velik broj problema, uključujući mnoge probleme što ih današnja računala ne mogu riješiti.

Upravo je snaga prirodne evolucije potakla brojne autore da u prirodi potraže inspiraciju za osmišljanje novih računalnih algoritama. Osim genetskih algoritama, tako su nestali genetsko programiranje, evolucijsko programiranje, evolucijske strategije, simulirano kaljenje, neuronske mreže i brojne druge metode umjetne inteligencije i optimiranja.

2.1. Evolucijski algoritmi

Evolucijski algoritmi su skupina algoritama nastalih oponašanjem evolucijskih procesa koje susrećemo u prirodi. Dijele se na četiri najvažnije skupine:

genetski algoritmi (engl. *genetic algorithms*, GA),
genetsko programiranje (engl. *genetic programming*, GP),
evolucijske strategije (engl. *evolution strategies*, ES) i
evolucijsko programiranje (engl. *evolutionary programming*, EP).

U ovom se radu za pronalazak najboljih rješenja promatranih problema koristiti genetski algoritmi i genetsko programiranje.

Genetsko programiranje

Ideja iz koje je nastalo genetsko programiranje (GP) temelji se na ideji genetskog algoritma (GA). Metode se razlikuju u tome što GA pretražuje prostor mogućih rješenja i kao rezultat svojeg rada daje najbolje rješenje, a GP pronalazi program koji može pronaći najbolje rješenje. Genetsko programiranje je detaljno opisano u poglavlju 3.

Evolucijske strategije

Kao i prethodne metode, i evolucijske strategije su tehnika optimiranja temeljena na idejama evolucije. Jedinke koje se koriste za ovu metodu kodiraju se kao realni brojevi. Proces stvaranja no-

vih jedinki ponešto se razlikuje od procesa kojeg koriste GA i GP, ali su temeljeni na sličnim principima: mutaciji, rekombinaciji gena i selekciji.

Evolucijsko programiranje

Inicijalno je zamišljeno za evoluciju konačnih automata, ne uključuje poseban način prikaza jedinke, a od operatora se koristi samo mutacija. Mutacijom se stvaraju nove jedinke od kojih se formira nova populacija [Jak05].

2.2. Evolucija u prirodi

Proučavanju evolucije u prirodi najviše je doprinio britanski znanstvenik Charles Darwin u 19. stoljeću. On je u svom djelu „Porijeklo vrsta“ postavio temelje današnje znanosti o biološkoj evoluciji.

Darwin je u svojoj knjizi predstavio 5 ključnih značajki evolucijskog procesa:

A. Sve vrste produciraju više potomaka od trenutnog broja jedinki u jednoj generaciji
B. Tijekom vremena prosječna veličina populacije neke vrste ostaje uvijek jednaka (postoji samo blagi trend rasta) neovisno o velikom broju potomaka
C. Dostupna hrana (resursi) je ograničena, ali njezina količina je većinom konstantna kroz određeni period.

Iz prethodnih triju tvrdnji vidljivo je da se jedinka pojedine vrste mora boriti kako bi preživjela.

D. Kod vrsta koje se razmnožavaju spolno, općenito gledajući, nema dvije jedinke koje su jednake: jedinke nasljeđuju svojstva od svojih roditelja uz poneku varijaciju (mutaciju).
E. Varijacije su često nasljedne.

Jedinka s najboljim svojstvima imat će i najveću vjerojatnost preživljavanja. Tako će se dobra svojstva nastaviti prenositi dalje na njezino potomstvo. Tijekom vremena dobra svojstva će postati dominantna i proširit će se populacijom.

Evolucija je vođena dvama osnovnim procesima:

Prirodna selekcija
Selekcija je proces kojim se vrši odabir jedinki iz populacije; one će preživjeti i dalje se reproducirati.

Spolna reprodukcija
Proces reprodukcije osigurava da svaka nova jedinka od roditelja naslijedi neka svojstva. Tijekom tog procesa događaju se i mutacije koje pogoduju stvaranju različitosti u genetskom materijalu. Evolucijski proces je znatno kvalitetniji kada u reprodukciji sudjeluju dva

roditelja, nego u slučaju da sudjeluje samo jedan, kao kod nespolnog razmnožavanja. Evolucijski procesi kod organizama koji se razmnožavaju nespolno mogu se vršiti samo mutacijama.

2.3. Struktura jednostavnog genetskog algoritma

Struktura jednostavnog genetskog algoritma prikazana je na slici 2.1.

```
Genetski_algoritam(Velicina_Populacije, t, pm, pc, M)
{
    t = 0;
    generiraj početnu populaciju potencijalnih rješenja P(0);
    sve dok nije zadovoljen uvjet završetka evolucijskog procesa
    {
        t = t + 1;
        selektiraj P'(t) iz P(t-1);
        križaj jedinke iz P'(t) i djecu spremi P(t);
        mutiraj jedinke P(t);
    }
    ispiši rješenje;
}
```

Slika 2.1. Struktura jednostavnog genetskog algoritma

Parametri genetskog algoritma su:

Veličina populacije - broj jedinki u jednoj generaciji

t - uvjet završetka genetskog algoritma; najčešće se iskazuje kao broj iteracija algoritma, vrijeme trajanja ili broj iteracija u kojima nije došlo do promjene najbolje jedinke

p_m - vjerojatnost mutacije

p_c - vjerojatnost križanja

M - mortalitet

2.4. Proces evolucije kao motivacija za genetski algoritam

U ovom poglavlju uspoređeni su osnovni mehanizmi koji se koriste u procesu evolucije s onima koji se koriste u genetskim algoritmima.

Osnovni parametri koji određuju evoluciju, pa tako i genetske algoritme, su: način kodiranja svojstava jedne jedinke, način evaluacije dobrote jedinke, način selekcije, križanja i mutacije.

2.4.1. Kodiranje svojstava jedinke

Kodiranje u prirodi

Danas se pretpostavlja da su sva svojstva živog bića pohranjena u kromosomima. Kromosomi su lančaste tvorevine koje se nalaze u jezgri svake stanice. Djelić kromosoma u kojem je pohranjeno jedno svojstvo, naziva se gen.

Kemijska struktura koja je prisutna u kromosomima je molekula DNK (eng. *deoxyribonucleic acid* – deoksiribonukleinska kiselina). To je dvolančana molekula u kojoj je pohranjen „kod“ (upute) kako izgraditi organizam. Između dvaju lanca te molekule nalaze se četiri komplementarne kiseline: adenin (A) + timin (T) i citozin (C) + guanin (G). Pomoću tih kiselina moguće je ostvariti sistem kodiranja zasnovan na četverobrojnom sustavu. “Brojevi” su A+T, T+A, C+G i G+C.

Pojednostavljeni prikaz molekule DNK je dan na slici 2.2.

Slika 2.2. Pojednostavljeni prikaz molekule DNK

Za dekodiranje neke informacije dovoljno je promatrati samo jedan lanac, jer su kiseline u drugom lancu uvijek komplementarne kiselinama u prvom. Postojanje komplementarnih lanaca omogućava dupliciranje molekule DNK.

Točno određeni dijelovi molekule dekodiraju se u gen koji opisuje neko svojstvo. Istraživanjem nizova dušičnih kiselina utvrđene su kombinacije od četiriju kiselina na jednom lancu, koje odgovaraju jednom od 20 proteina od kojih su izgrađena živa bića. Smatra se da je tek dio kiselina u lancu DNK zaslužan za kodiranje korisnih informacija, a prema sadašnjim istraživanjima velik dio informacija pohranjenih u DNK predstavlja neki oblik redundantnih informacija.

Kodiranje u genetskom algoritmu

Odabir prikaza jedinke u genetskom algoritmu je preduvjet rješavanja nekog problema genetskim algoritmima. O načinu prikaza jedinke uvelike ovisi kvaliteta genetskog algoritma i njegova mogućnost da pronađe najbolje rješenje.

Postoji mnoštvo načina za kodiranje kromosoma u računalu. Najčešći je prikaz kromosoma kao slijeda binarnih brojeva. U praksi se pokazalo da upravo ovaj prikaz daje najbolje rezultate.

Kod binarnog prikaza kromosom je prikazan kao binarni vektor x [dg,gg]. Pritom vektor B = 000… 00 predstavlja donju granicu, a vektor B = 111… 11 gornju granicu. Dužina vektora *n*, definira broj rješenja koja se mogu prikazati. Pretvorba brojeva vrši se sljedećim izrazima [Gol04]:

$$x = dg + \frac{b}{(2^n - 1)}(gg - dg) \qquad b = \frac{x - dg}{gg - dg}(2^n - 1) \qquad (2.1)(2.2)$$

Osim binarnog kodiranja, kod optimiranja problema s realnim brojevima moguće je koristiti i druge prikaze. Primjerice jedan od načina kodiranja je i prikaz realnog broja s pomičnom točkom (npr. prema standardu IEEE 754-1985).

Za rješavanje problema raspoređivanja moguće je koristiti i nizove cijelih brojeva u prikazu kromosoma. Primjerice, za rješavanje problema trgovačkog putnika kromosom se može predstaviti kao niz cijelih brojeva koji predstavljaju slijed gradova što ih putnik mora obići.

Osim ovdje navedenih postoje i mnoge druge reprezentacije, primjerice one zasnovane na poljima, stablima, listama i slično. Ove su detaljnije opisane u poglavlju o genetskom programiranju.

Odabir prikaza kromosoma uvjetuje izvedbu genetskih operatora kao što su križanje i mutacija.

2.4.2. Evaluacija dobrote pojedine jedinke

Evaluacija dobrote u prirodi

Evaluacija dobrote u prirodi ne postoji kao točno definiran matematički izraz. Nemožemo reći da je jedna jedinka u nekoj populaciji u potpunosti bolja od druge. Prirodno okruženje je dinamično i ono stalno evaluira jedinke različitim „izazovima“.

Evaluacija dobrote u genetskom algoritmu

Za ostvarenje simulacije prirodnog okruženja potrebno je prilikom implementacije genetskog algoritma ispravno odabrati način ocjenjivanja dobrote pojedine jedinke. Dobrota neke jedinke je mjera kvalitete toga rješenja u zadanom prostoru rješenja. Kod jednostavnih problema, kao što je pronalaženje minimuma neke funkcije, dobrotu možemo prikazati kao vrijednost koju ta funkcija ima za zadani kromosom. Kod složenijih problema dobrota se može izraziti kao, primjerice, vrijeme trajanja projekta, vrijeme koje je potrebno da se obavi simulacija sustava koji optimiramo i slično.

2.4.3. Način selekcije

Selekcija u prirodi

Selekcija se u prirodi obavlja evaluacijom fenotipa jedinke. Fenotip je skup značajki koje su nastale na temelju genotipa neke jedinke i kao posljedica interakcije jedinke s čimbenicima iz okoline. Prirodno okruženje odabire one jedinke koje imaju najkvalitetniji genetski materijal. Takve, najbolje jedinke najčešće sudjeluju u reprodukciji, tako da se najveći dio tog dobrog genetskog materijala prenosi na njihove potomke. S vremenom prosječna kvaliteta genetskog materijala postaje sve bolja i bolja. U prirodi taj proces ipak nije strogo definiran i može se dogoditi da u nekoj vrsti nakon dugog razdoblja poboljšavanja nastupi period u kojem prosječna kvaliteta genetskog materijala postaje sve lošija. Prirodna selekcija je mehanizam koji se stalno mijenja. U jednoj „iteraciji“ prirodne selekcije jedinke s određenim svojstvima će biti najbolje, a u nekoj drugoj „iteraciji“ evolucija može cijeniti neka posve drugačija svojstva. U prirodnoj selekciji postoji i doza slučajnosti jer nije moguće sa sigurnošću utvrditi na koji način se odabiru pojedine jedinke kao najbolje.

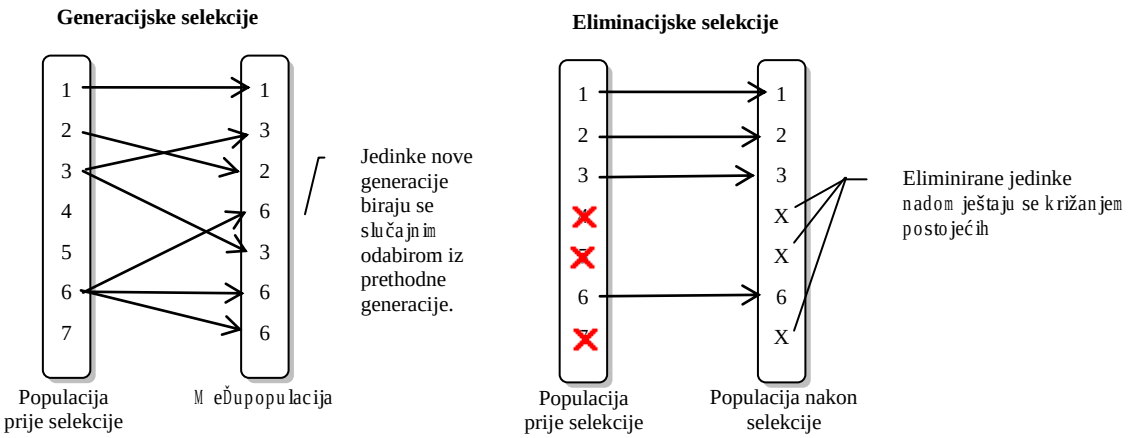
Selekcija u genetskom algoritmu

Selekcija je proces kojim se osigurava prenošenje boljeg genetskog materijala iz generacije u generaciju. Postupci selekcije međusobno se razlikuju po načinu odabira jedinki koje će se prenijeti u sljedeću generaciju.

Prema načinu prenošenja genetskog materijala selekcije se dijele na (Slika 2.3):

Generacijske selekcije
Proces odabire najbolje jedinke i od njih kreira novu generaciju.

Eliminacijske selekcije
Proces selekcije eliminira najgore jedinke iz generacije.



Slika 2.3. Podjela selekcija prema načinu prenošenja genetskog materijala

Generacijske selekcije djeluju tako da iz populacije odabiru određen broj najboljih jedinki i od njih se stvara međupopulacija. Broj jedinki koje prežive selekciju je manji od veličine populacije pa se jedinke koje nedostaju nadomještaju kopiranjem već selektiranih jedinki. Nedostatak ovoga načina selekcije je istovremeno postojanje populacije i međupopulacije u istom spremniku. Drugi nedostatak je proces višestrukog kopiranja istih jedinki, što rezultira nekvalitetnim (vrlo sličnim) genetskim materijalom.

Kod eliminacijskih selekcija nema stroge granice između prethodne i sljedeće generacije. Tijekom procesa eliminacije odstranjuju se slučajno odabrane jedinke (lošije jedinke imaju veću vjerojatnost eliminacije). Eliminirane jedinke nadomještaju se križanjem postojećih.

Selekcije možemo klasificirati i prema načinu odabira pojedinih jedinki [Gol02]. Selekcije u tom slučaju dijelimo na:

Proporcionalne selekcije

- Jednostavna proporcionalna
- Stohastička univerzalna

Rangirajuće selekcije

- Sortirajuće selekcije
  - Selekcije najboljih

    (,) selekcija

    (+) selekcija

Krnja selekcija

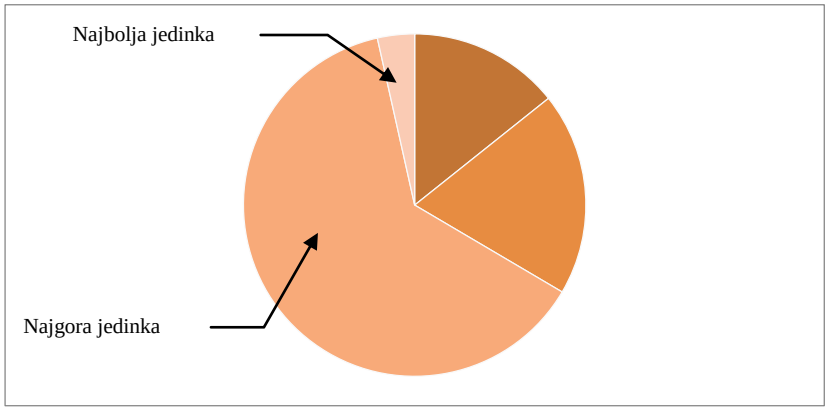
- Linearno sortirajuća selekcija

- Turnirske selekcije
  - K-turnirska selekcija
  - Jednostavna turnirska selekcija

Prilikom izrade ovoga rada korištena je k-turnirska selekcija.

Jednostavna proporcionalna selekcija

Jednostavnu proporcionalnu selekciju najslikovitije možemo prikazati kao kotač ruleta sa različitim veličinama za svaki djelić kotača. Za svaku jedinku naše populacije definirati ćemo dobrotu, a obrnuto proporcionalnu vrijednost ćemo unijeti na kotač za tu jedinku. Tako će najbolja jedinka imati najmanji isječak kotača (postoji najmanja vjerojatnost da će upravo ona biti eliminirana), a najlošija jedinka najveći isječak kotača (Slika 2.4.).

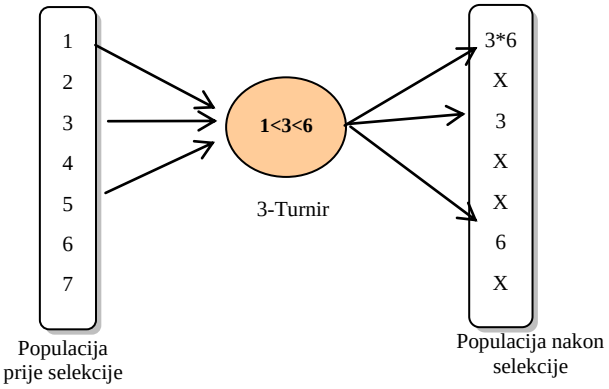


Slika 2.4. Primjer jedinki raspoređenih po kotaču ruleta

K-turnirska selekcija

U k-turnirskim selekcijama odabire se *k* članova populacije koji sudjeluju u turniru. Tijekom turnira uspoređuju se dobrote svih *k* jedinki.

Primjer 3-turnirske selekcije prikazan je na Slici 2.5. Slučajnim odabirom izdvojene su jedinke 1, 3 i 6 za sudjelovanje u turniru. Jedinka 1 ima najmanju dobrotu i ona se eliminira, a jedinke 3 i 6 se križaju i produciraju novu jedinku „3*6“ koja se dodaje u populaciju.

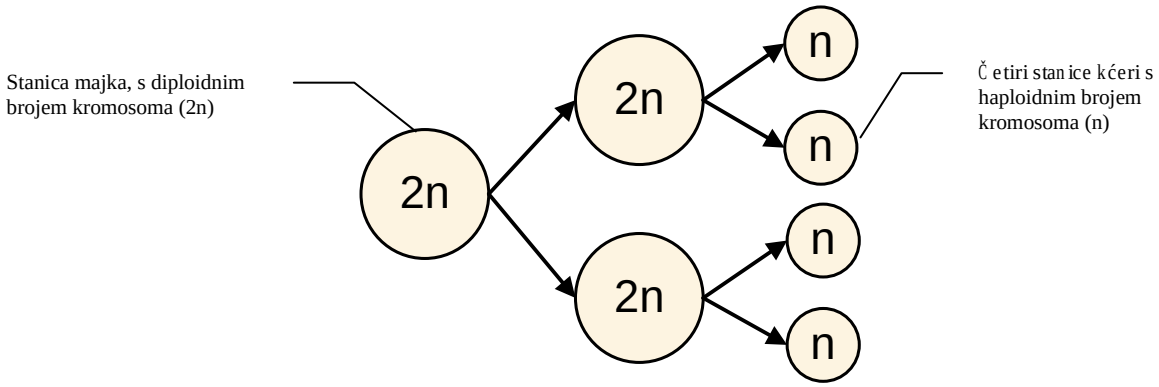


Slika 2.5. Shema 3-Turnirske selekcije

2.4.4. Križanje

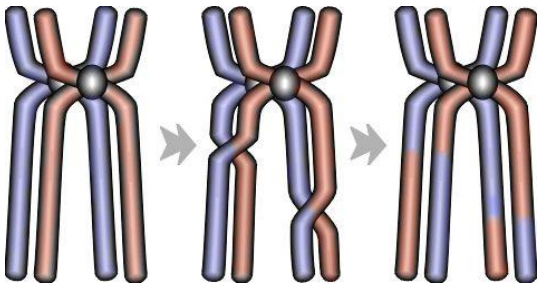
Križanje u prirodi

Križanje u prirodi je vrlo složen proces u kojem najveću ulogu imaju kromosomi. Broj kromosoma jedne jedinke je paran, dakle *2n* (čovjek ima 46 kromosoma). Tijekom procesa mejoze (obavlja se jedino u spolnim stanicama) nizom složenih procesa, nastaju četiri nove stanice s haploidnim (polovičnim) brojem kromosoma (Slika 2.6.).



Slika 2.6. Pojednostavljena shema procesa mejoze

Tijekom procesa mejoze osim stvaranja novih spolnih stanica dolazi i do izmjene genetskog materijala. To je proces u kojem dva kromosoma formirana tijekom rane faze mejoze izmjenjuju dijelove kromosomskih niti, kao što je prikazano na slici 2.7.

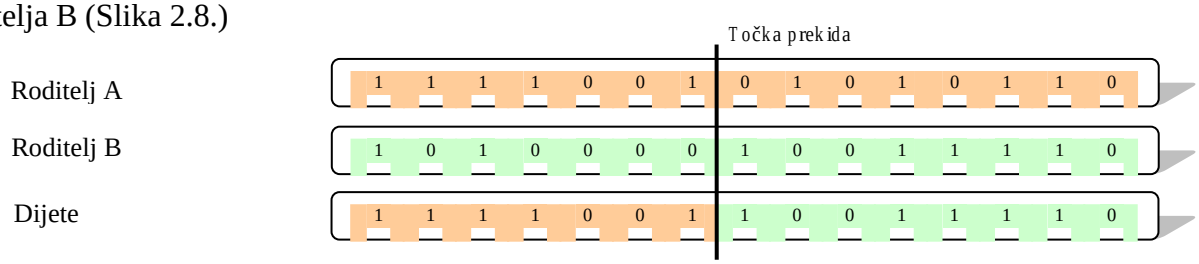


Slika 2.7. Shema procesa križanja prilikom kojeg dolazi do zamjene dijelova kromosomskih niti

Križanje u genetskom algoritmu

Križanje je proces u kojem se od dva roditelja, križanjem njihovih gena, dobije jedna ili dvije jedinke koje predstavljaju njihovo potomstvo. Izvedba križanja u računalu uvelike ovisi o načinu na koji su kromosomi prikazani. Kada je kromosom prikazan kao vektor bitova, može se primjenjivati mnogo načina križanja. Neki od najčešćih načina su:

1. Križanje s jednom točkom prekida
   Slučajnim odabirom određuje se točka prekida i vektori obaju roditelja se prekidaju na tom mjestu. Do točke prekida dijete nasljeđuje svojstva roditelja A, a od točke prekida roditelja B (Slika 2.8.)



Slika 2.8. Križanje s jednom točkom prekida

2. Križanje s dvjema točkama prekida
   Ovo je križanje slično križanju s jednom točkom prekida, s jedinom razlikom što se nakon svake točke genetskim materijalnaizmjence preuzima od drugog roditelja.

3. Uniformno križanje
   Provodi se kao logička operacija nad bitovima s logičkim operatorima I, ILI i XOR. Izraz za uniformno križanje:

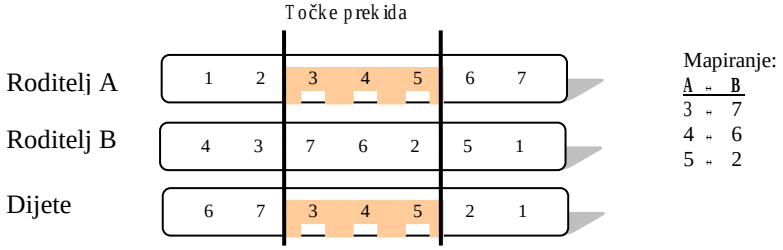
   DIJETE = AB + R (A B) (2.3)

   Pri čemu su *A* i *B* roditelji, a *R* je slučajno generirani vektor jednake duljine.

   Korištenje uniformnog križanja osigurava prijenos onih svojstava koja su kod oba roditelja jednaka.

Ako je kromosom kodiran kao niz cijelih brojeva (bez ponavljanja), koriste se neki drugi načini križanja. U [Mic96] predložen je PMX kao metoda križanja kromosoma koji su prikazani kao nizovi cijelih brojeva. Metoda funkcionira kao križanje s dvjema točkama prekida. Između preki-

dnih točaka uzima se kromosom roditelja *A*, a ostatak se nadom ješta krom osom om roditelja *B*. U slučaju da gen koji se kopira već postoji u djetetu, koristi se mapiranje kao na slici 2.9.



Slika 2.9. Prikaz PM X m etode križanja za krom osom e koji su predstavljeni kao nizovi cijelih brojeva

2.4.5. Mutacije

Mutacije u prirodi

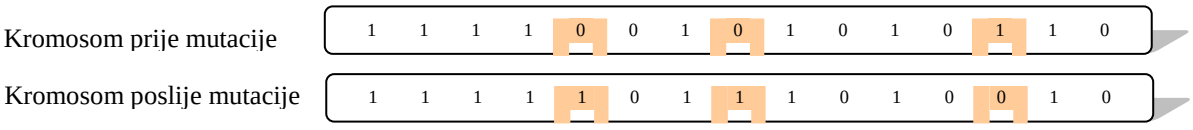
Pod pojmom mutacije u prirodi se podrazumijeva bilo koja promjena genetskog materijala, a najčešće nastaje kao greška u kopiranju prilikom procesa mitoze (dioba stanica) i mejoze (dioba stanica koja rezultira stanicama sa haploidnim brojem kromosoma). N ajveći broj m utacija pokazuje se štetnim za organizam. MeĎutim , da bi proces evolucije bio potpun, potrebne su stalne promjene. Dobre mutacije ugradit će se u organizam i nastaviti prenositi na nove jedinke u sljedećim generacijama. Iznim no loše m utacije m ogu dovesti i do sm rti organizma (primjerice kod pojave tumora).

Mutacije u genetskom algoritmu

M utacije uvelike pom ažu izbjegavanju lokalnih optim um a funkcije cilja koju optim iram o. Primjenom m utacija postiže se raznolikost genetskog m aterijala i om ogućava pretraživanje novih – potencijalno najboljih rješenja. M utacije takoĎer obnavljaju genetski m aterijal. Kad nam sve jedinke na jednom dijelu kromosomu imaju istu vrijednost, ta se vrijednost tijekom križanja nikad neće m ijenjati. T ijekom procesa m utacije postoji m ogućnost da se upravo taj dio kromosoma promjeni.

V jerojatnost m utacije odreĎuje se kao jedan od param etara genetskog algoritm a. Vjerojatnost m utacije jednog bita obično se odabire u intervalu 0,001 – 0,01. Primjerice, ako je p_m = 0,005, znači da m utira 5 bitova na 1000.

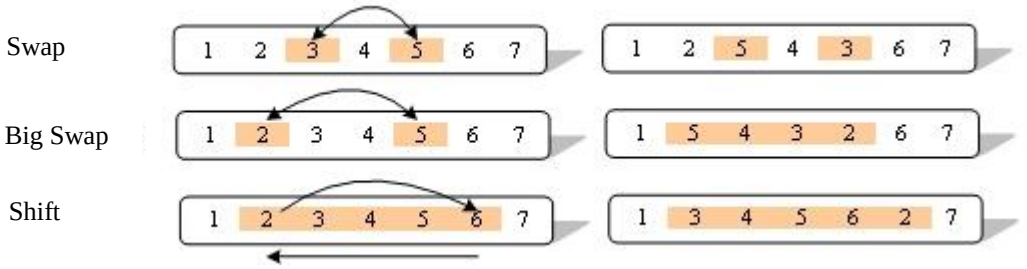
Primjer jednostavne mutacije binarno kodiranog kromosoma prikazan je na slici 2.10. Bitovi krom osom a koji su m utirali označeni su narančastom bojom .



Slika 2.10. – Prikaz mutacija kod binarno kodiranog kromosoma.

Osim jednostavne mutacije, mogući su i drugi načini mutiranja binarnog kromosoma, kao što su: miješajuća mutacija, potpuna miješajuća mutacija i invertirajuća mutacija.

Kad se koriste neke druge reprezentacije kromosoma, mutacije su nešto drugačije. Ako kromosom kodiramo kao niz cijelih brojeva neke od mogućih mutacija su: Swap, Big Swap i Shift. Ove mutacije su prikazane na slici 2.11.



Slika 2.11. Operatori mutacije za kromosome koji su kodirani kao nizovi cijelih brojeva

2.5. Jednostavni genetski algoritam

U velikom broju slučajeva većinu problema moguće je riješiti koristeći samo jednostavni genetski algoritam. Jednostavni genetski algoritam se sastoji od:

1. kromosoma kodiranog binarno,
2. jednostavne selekcije,
3. jednostavnog križanja s jednom točkom prekida i
4. jednostavne mutacije.

2.6. Evaluacija upotrebljivosti genetskog algoritma

Genetske algoritme koji su ostvareni za optimiranje projekata s ograničenim sredstvima, a opisani su u sljedećim poglavljima, potrebno je evaluirati kako bi se utvrdila njihova upotrebljivost za rješavanje zadanih problema. Evaluacija se može provoditi na dva načina:

A. Usporedbom pronađenih optimalnih rješenja s rješenjima koja su dobivena drugim metodama i

B. Usporedbom efikasnosti pronalaženja rješenja s efikasnošću Monte Carlo simulacije.

Prvi je postupak vrlo jednostavan. Ako genetski algoritam sa podešenim parametrima pronalazi isto optimalno rješenje koje je pronađeno nekom heurističkom metodom, možemo zaključiti da ostvareni genetski algoritam daje dobra rješenja za problem koji smo mu zadali.

Monte Carlo simulacija je metoda slučajnog pretraživanja prostora rješenja. Algoritmi Monte Carlo se koriste za simulaciju različitih fizičkih i matematičkih sustava. Rješenja pronalaze slučajnim generiranjem elementa iz skupa mogućih rješenja. Ostvareni genetski algoritam je dobar ako rješenje pronalazi brže (u manjem broju iteracija i/ili u kraćem trajanju) od metode Monte Carlo.

3. Genetsko programiranje

Genetsko programiranje je osmislio John R. Koza. Proučavajući genetske algoritme i druge evolucijske metode, on je odlučio primijeniti te tehnike u novom području. Želio je oblikovati računalni program koji će biti u mogućnosti sam pisati druge računalne programe.

Osnovna ideja za genetsko programiranje proizišla je, kao i za genetske algoritme, iz teorije prirodne evolucije. Cilj optimiranja koje se provodi genetskim algoritmima je pronaći optimalno rješenje za problem koji rješavamo. Kod genetskog programiranja zadatak je evoluiranjem računalnih programa pronaći optimalni računalni program koji može na najbolji način riješiti zadani problem.

U prirodi organizmi „bolje“ strukture imaju veću vjerojatnost preživljavanja i reprodukcije. Biolozi interpretiraju strukture koje promatraju kao posljedicu Darwinove prirodne selekcije tijekom vremenskog perioda. Drugima riječima, struktura je posljedica dobrote. Korištenjem evolucijskih mehanizama (prirodne selekcije, križanja i mutacije) proces evolucije stvara jedinke sa sve boljom i boljom strukturom.

Općenito govoreći, računalni je program također struktura, a genetsko programiranje je evolucijska metoda pomoću koje se pronalaze optimalni programi za rješavanje zadanog problema. Prostor mogućih rješenja, kada tražimo rješenje u obliku strukture, vrlo je velik. Za genetski algoritam kromosom je najčešće niz bitova. Kod genetskog programa kromosom je stablo koje može imati neograničen broj čvorova i listova.

Već se za genetske algoritme pokazalo da pretražuju prostor rješenja mnogo brže nego slijepe metode pretraživanja. Veličina prostora u kojem se traži program daleko je veća od prostora u kojem se traži samo neki optimum, stoga genetsko programiranje ima daleko teži zadatak od genetskih algoritama.

U svojoj osnovi genetsko programiranje se ne razlikuje previše od genetskih algoritama. Za obje metode promatramo iste operatore (križanje, mutacija i selekcija), definiramo populacije, generacije, jedinke, funkcije dobrote. Općenito gledajući, genetsko je programiranje samo specijalizirana verzija genetskih algoritama koja ne pronalazi najbolje rješenje jednog problema, nego izraz koji bi se mogao koristiti u većem broju slučajeva.

3.1. Prikazivanje jedinke

Odabir ispravnog načina prikazivanja pojedine jedinke je najvažniji preduvjet da bi neki GA ili GP mogao pronaći optimalno rješenje zadanog problema. Prikaz jedinke mora s minimalnim brojem gena predočiti sva važna svojstva jedne jedinke. Genetski zapis jedinke mora biti takav da prilikom križanja i mutacije ne dolazi do nemogućih rješenja. Jedinka nastala križanjem ne smije kršiti unaprijed definirana ograničenja. Takve jedinke nepotrebno usporavaju proces evaluacije jedinki i pronalaženje optimuma, jer nepotrebno proširuju prostor mogućih rješenja na dio u kojem se sigurno ne može pronaći optimalno rješenje.

Genetsko programiranje za razliku od genetskih algoritama ne traži optimalno rješenje, već optimalnu strukturu (program) koja će izračunati rješenje problema. Zbog povećane kompleksnosti je stoga potrebno odabrati i odgovarajuću strukturu.

Kako bismo kreirali računalni program potrebni su nam skupovi završnih i nezavršnih znakova. Nezavršni znakovi su funkcije koje primaju jedan ili više argumenata, a završni znakovi su argumenti koji se predaju tim funkcijama.

Nezavršni znakovi su najčešće [Koz98]:

Aritmetičke operacije (+, -, *, itd.)
Matematičke funkcije (sin, cos, exp i log)
Booleove funkcije (AND, OR, NOT)
Uvjetni operatori (If-Then-Else)
Operatori iteracije (kao Do-Until)
Funkcije rekurzije
Neki drugi operatori koji su povezani s problemom koji se rješava

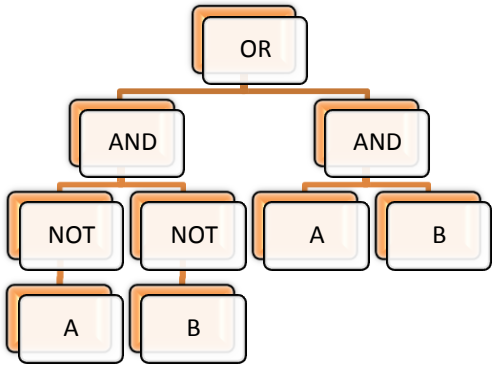
Završni znakovi su najčešće cijeli, realni brojevi ili logičke vrijednosti. Odabir tih znakova uvelike ovisi o problemu u koji rješavamo. Ako primjerice izgrađujemo genetski program koji pronalazi optimalnu funkciju nekog logičkog sklopa, skupovi završnih i nezavršnih znakova bi mogli izgledati ovako:

Skup nezavršnih znakova: *F={AND, OR, NOT}*
Skup završnih znakova: *T={A, B}*

Jedan od mogućih izraza mogao bi biti:

f = NOT A AND NOT B OR A AND B.

Gornji izraz možemo prikazati i kao stablo sa svim znakovima. Stablo je prikazano na slici 3.1.



Slika 3.1. Prikaz gornjeg izraza u obliku stabla

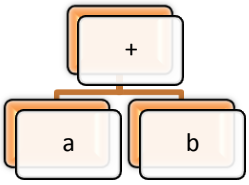
Jedinke koje se koriste u GP-u prikazivat će se u obliku stabla radi pojednostavljenja procesa križanja i mutacije, što će biti objašnjeno u sljedećim poglavljima.

Koza je u svom radu [Koz98] odabrao programski jezik LISP kao jezik u kojem su bile zapisane jedinke njegovih programa. GP može za zapisivanje jedinke koristiti bilo koji programski jezik, međutim jednostavnost LISP-a i način zapisivanja njegovih izraza su prednost zbog lakše implementacije genetskih operatora. Primjerice, izraz na slici 3.2. je klasičan LISP S-izraz.

```
(+ 1 2)
```

Slika 3.2. LISP S-izraz

Gornji izraz je zapisan u poljskoj notaciji, jer se operator + nalazi zapisan prije operanada 1 i 2. Zapravo se radi o izrazu *1+2*. Ovakav se izraz može vrlo jednostavno prikazati i kao stablo (Slika 3.3.).



Slika 3.3. Prikaz jednostavne LISP formule kao stabla

Upravo jednostavnost konvertiranja LISP S-izraza u stabla i obrnuto je prednost LISP-a kao programskog jezika pomoću kojeg se zapisuju GP jedinke.

Preciznost pronađenih rješenja

Genetski program koji pretražuje prostor rješenja može pronaći najbolje moguće rješenje. Zbog nesavršenosti računala i nemogućnosti apsolutne preciznosti računalnih programa, moguće je da ćemo ponekad dobivati rješenja koja nisu matematički ispravna ali daju identične rezultate.

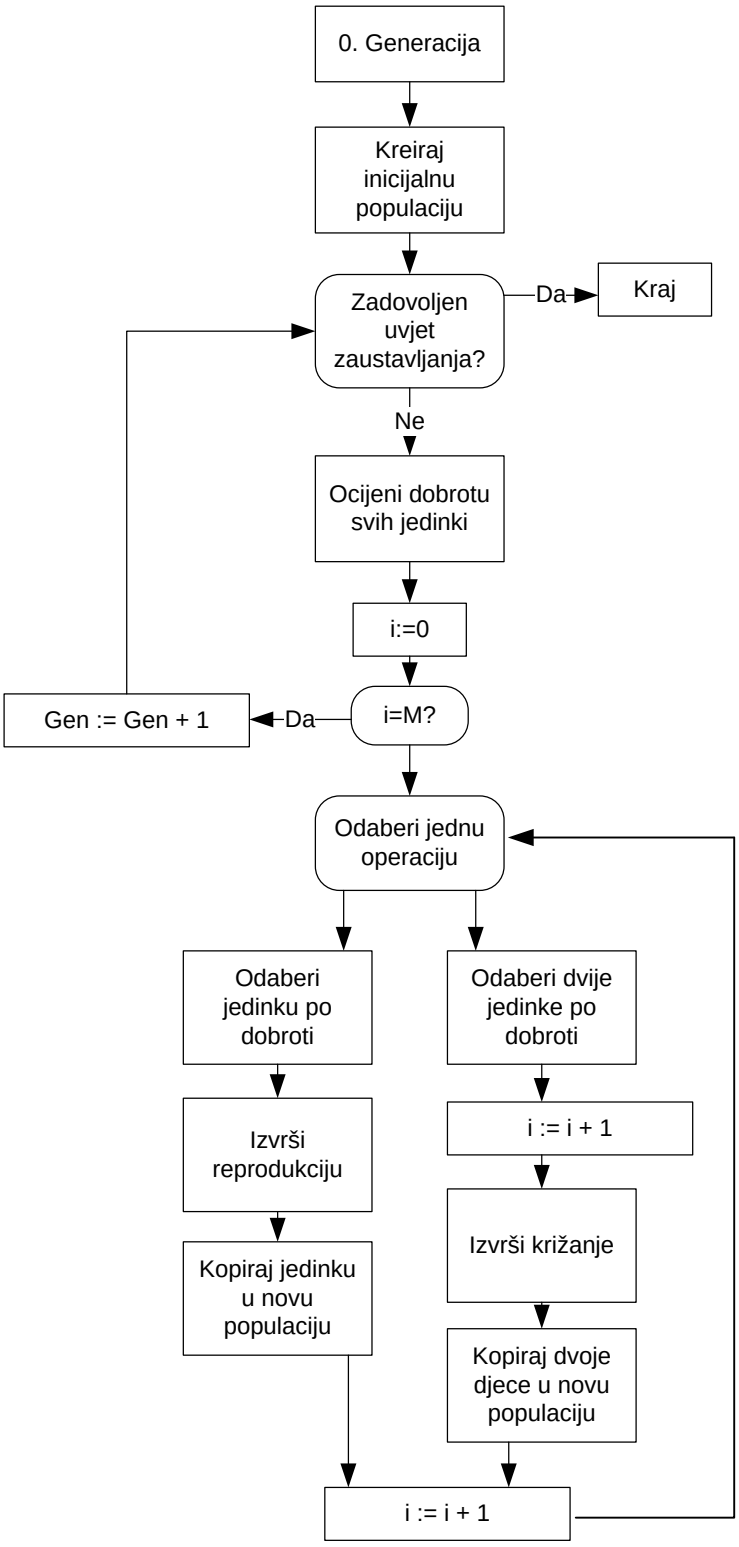
U [Koza98] dan je primjer za problem rješavanja kvadratne jednadžbe $ax^2 + bx + c = 0$, čije je matematičko rješenje: $x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$. Autor je korištenjem genetskog programa dobio nje: $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} + 0.000000000000001a^3bc$. Gledano strogo matematički, ovo rješenje nije točno jer postoji pogreška od $10^{-15}a^3bc$. Međutim, budući da se radi o izrazito maloj šci rješenja koja ćemo dobiti ovakvim programom mogla bi, ovisno o postavljenim zahtjevima, biti zadovoljavajuća.

Preciznost pojedinih rješenja uvelike ovisi o problemu koji rješavamo. Ako izrađujemo program koji pronalazi idealne težine atoma koji sudjeluju u nuklearnoj reakciji pogreška od $10^{-15}kg$ može dovesti do propasti. S druge pak strane, pogreška u trajanju projekta od 10^{-15} $dana$ ne znači ništa, stoga takvo rješenje možemo prihvatiti bez problema.

3.2. Tijek rada genetskog algoritma

Dijagram toka genetskog algoritma je prikazan na slici 3.4. Slika prikazuje tijek programa kako je predložen u [Koz98]. Prema Kozi, proces se odvija u sljedećim fazama:

1. Kreiraj inicijalnu populaciju sastavljenu od programa
2. Ponavljaj dok nije zadovoljen uvjet zaustavljanja
    A. Pokreni svaki program u populaciji i izračunaj njegovu dobrotu. Dobrotu izračunaj tako da pokreneš taj program za zadani problem i provjeriš rješenje.
    B. Kreiraj novu populaciju jedinki korištenjem sljedećih operatora:
        i. Kopiraj postojeći program u novu generaciju
        ii. Kreiraj novi program križanjem dvaju postojećih programa.
3. Najbolji računalni program koji se pojavio tijekom izvoĎenja je rješenje. Rješenje može biti matematički točno ili približno točno.



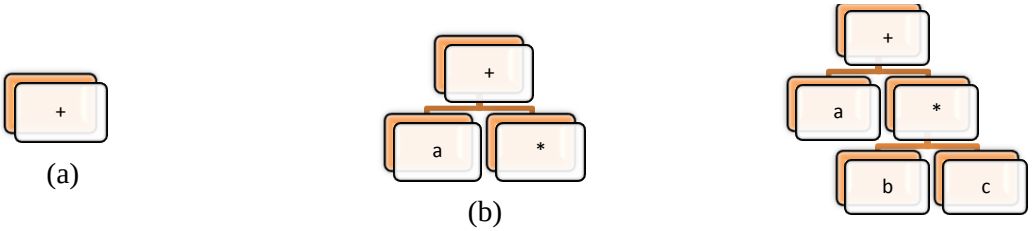
Slika 3.4. Tijek obavljanja optimiranja genetskim programiranjem

Na gornjoj slici indeks *i* označava jedinku iz populacije veličine *M*. Varijabla *GEN* je broj trenutne generacije. Slika je prilagođena prema [Koz03].

3.3. Stvaranje početne populacije

Nakon što smo odabrali gene koji će opisivati našu jedinku, potrebno je stvoriti početnu - slučajno odabranu populaciju. Kreiranje inicijalne populacije je jedan od četiriju trenutaka u kojima se u procesu optimiranja koristi slučajni odabir. O sim ovdje, slučajni odabir se koristi i pri selekciji jedinki za sljedeću generaciju, odabiru točke prekida kod križanja i za mutiranje gena jedinke.

Prikaz jednog mogućeg načina stvaranja početne jedinke prikazan je na slici 3.5.



Slika 3.5. Kreiranje jedne jedinke za genetski program

Na gornjoj slici prikazano je stvaranje jedinke za genetski program. Slika (a) prikazuje stvaranje početnog čvora. Čvor je nezavršni, dakle proces se nastavlja. U drugom koraku, na slici (b), stvaraju se dva nova čvora. Proces se nastavlja za desni čvor. U zadnjem koraku, prikazanom na slici (c), proces završava stvaranjem dvaju završnih čvorova. Konačan izraz je $f=a+bc$.

Kvaliteta ostvarenog genetskog programa u velikoj će mjeri ovisiti o tome kako ćemo ostvariti one dijelove programa u kojima se neke odluke donose slučajnim odabirom. Jedan od tih dijelova je i generiranje početne populacije. U [Koz98, Jak05] opisane su tri tehnike koje se preporučuju za izgradnju početne populacije:

Grow
Full
Ramped half-and-half

3.3.1. Grow

Jedinka stvorena na ovaj način može biti stablo izgrađeno od elemenata što smo ih definirali. Parametar *m* koji određuje maksimalnu dubinu stabla unaprijed se zadaje.

Jedinke se generiraju prema sljedećem pravilu:

A. Počevši od korijenskog elementa, slučajno se odabire vrsta čvora koji će se dodati u stablo. Pritom je moguće birati operatore ili završne znakove (varijable).
B. Ako je izabran završni znak, sustav odabire jedan znak iz skupa završnih znakova.
C. Ako je odabran operator, odabire se jedan operator iz skupa operatora.

Postupak se dalje rekurzivno ponavlja sve dok svi novododani znakovi nisu završni ili dok se ne dostigne maksimalna dozvoljena dubina stabla *m*.

Jedinke kreirane ovom metodom neće imati stabla jednake dubine, a moguće je da će se izraz sastojati od samo jednog nezavršenog znaka. Općenito ovakvi izrazi nemaju neku veću vrijednost jer u najvećem broju slučajeva neće dati dobro rješenje.

3.3.2. Full

Metoda je vrlo slična prethodnoj, s jedinom razlikom da je unaprijed definirano na kojoj se dubini stabla mogu početi pojavljivati završni znakovi.

3.3.3. Ramped half-and-half

Kako bi se povećala varijacija strukture, obje prethodno navedene metode se mogu koristiti za stvaranje populacije. Ova tehnika, koja je nazvana *ramped-half-and-half*, jedina je metoda koju preporučuje Koza. Prilikom generiranja jedinki unaprijed se definira samo maksimalna dubina *md*. Prednost je ove metode što generira populacije sa dobrom raspodjelom jedinki po veličini i strukturi.

Kreiranje jedinki se obavlja na sljedeći način:

A. Populacija se podijeli na *md-1* dijelova
B. Polovina svakog dijela se generira metodom grow, a druga polovica metodom full. Za prvi dio parametar *m* za metodu grow i *d* za metodu full je 2. Za drugi dio je 3. Niz se nastavlja sve do *md-1*og dijela na kojem se koristi *md* kao vrijednost za *m* i *d*.

3.4. Reprodukcija

Reprodukcija GP jedinki obavlja se na sličan način kao i reprodukcija GA jedinki. Reprodukcija je aseksualan proces u kojem se selektira jedna jedinka iz trenutne populacije i prenosi se u drugu. Najbitniji faktor kod odabira jedinke za reprodukciju je način selekcije. Načini selekcije opisani su u prethodnom poglavlju, a po principu rada se ne razlikuju.

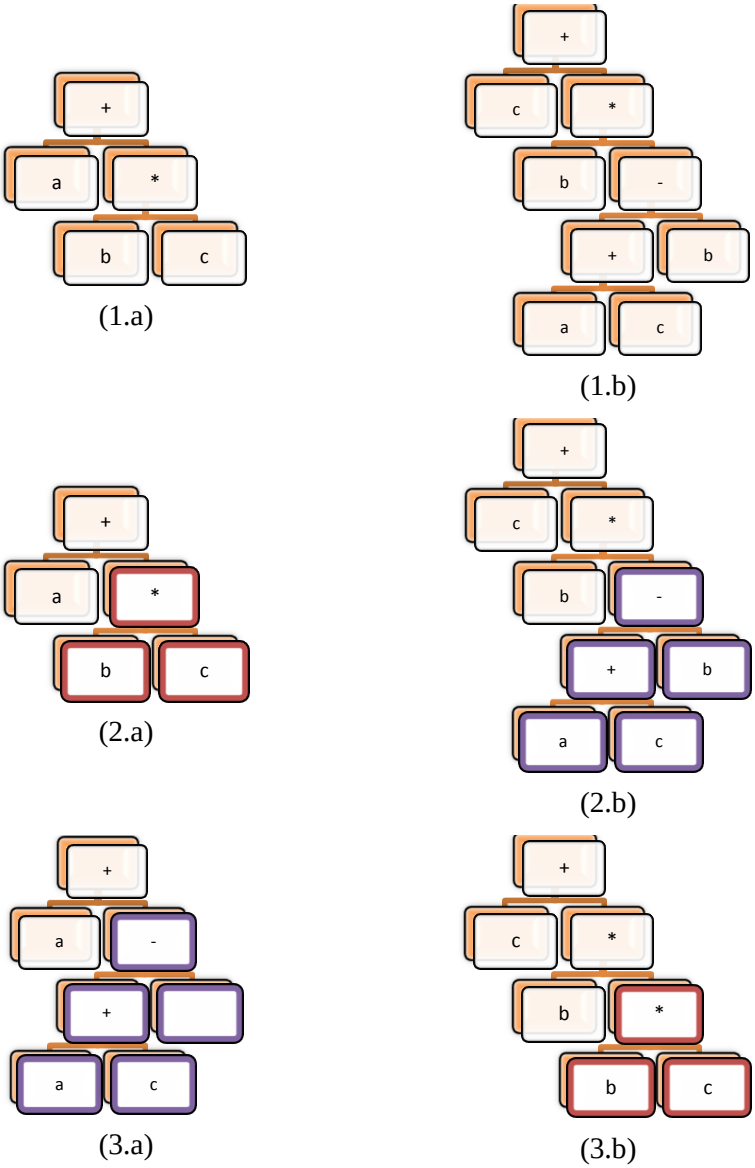
3.5. Križanje

Križanje jedinki je proces u kojem se rekombinira genetski materijal dvaju roditelja. Rezultat križanja su dvije jedinke djece, koje od svakog od roditelja naslijede dio genetskog materijala. Roditelji se odabiru na način kako je opisano u prethodnom poglavlju.

Sam tijek procesa križanja uvelike je sličan procesu koji se odvija u GA-u:

A. Slučajno odaberi točku prekida za obje jedinke (točka prekida je jedan čvor u stablu jedinke).
B. Izdvoji podstabla ispod točke prekida iz obiju jedinki.
C. Zamijeni izdvojena podstabla.

Primjer jednog križanja prikazan je na slici 3.6.



Slika 3.6. Prikaz križanja dviju jedinki genetskog programa

Gornja slika prikazuje tijek križanja dviju jedinki. Na slikama (1.a) i (1.b) prikazane su dvije jedinke koje su odabrane za križanje (roditelji). U sljedećem koraku, na slikama (2.a) i (2.b), roditeljima su slučajnim odabranom odabrana podstabla. Na roditelju *a* podstablo je označeno crvenom bojom, a na roditelju *b* ljubičastom . Nove jedinke (djeca) prikazane su na slikama (3.a) i (3.b). Jedinka (3.a) je nasljedila osnovno stablo od roditelja (1.a), a ružičasto označeno podstablo

od roditelja (1.b). Jedinka (3.b) je nasljedila osnovno stablo od roditelja (1.b) i crveno označeno podstablo od roditelja (1.a)

3.6. Mutacija

Kao i kod GA, mutacija služi da bi se u nove jedinke uveli neki novi geni koji će rezultirati novim jedinkama s boljim rješenjem. Općenito gledajući, mutacijama se može dobiti poboljšanje ali i pogoršanje prosječne dobrote jedinki u populaciji. Jedinke sa „smrtonosnim“ mutacijama će biti vrlo brzo eliminirane u selekcijskom procesu, tako da te jedinke ne bismo jele dugotrajno narušiti prosječnu dobrotu jedinki u populaciji.

Najvažnija razlika između GA-mutacije i GP-mutacije je u načinu provođenja. Ova razlika je uzrokovana različitim načinom prikaza jedinki. U GP-u jedinke se prikazuju kao stabla, a u GA najčešće kao nizovi bitova.

Mutacija za jedinke GP-a najčešće se radi na sljedeći način (proces se ponavlja više puta, ovisno o postavkama GP-a):

A. Iz populacije se odabere jedinka koja će mutirati
B. U stablu jedinke nasumce se odabere jedan čvor. Slučajno je moguće odabrati i čvor i list u stablu
C. Odabrani čvor (list) i sva njegova podstabla se brišu iz stabla
D. Na odabranom mjestu se zatim slučajnim odabirom kreira novo podstablo. Veličina i način stvaranja novog podstabla obično se definiraju kod pokretanja algoritma

Osim ovim načinom jedinke je moguće mutirati i samo dodavanjem završnih znakova na čvorove. Nasumce se odabere čvor kojem se onda doda jedan slučajno odabran završni znak (ako je to dozvoljeno tom operacijom). Osim toga, moguće je dodati i neku novu operaciju, primjerice unarnu operaciju NOT (kada izgrađujemo neki logički izraz).

3.7. Primjena genetskog programiranja: simbolička regresija

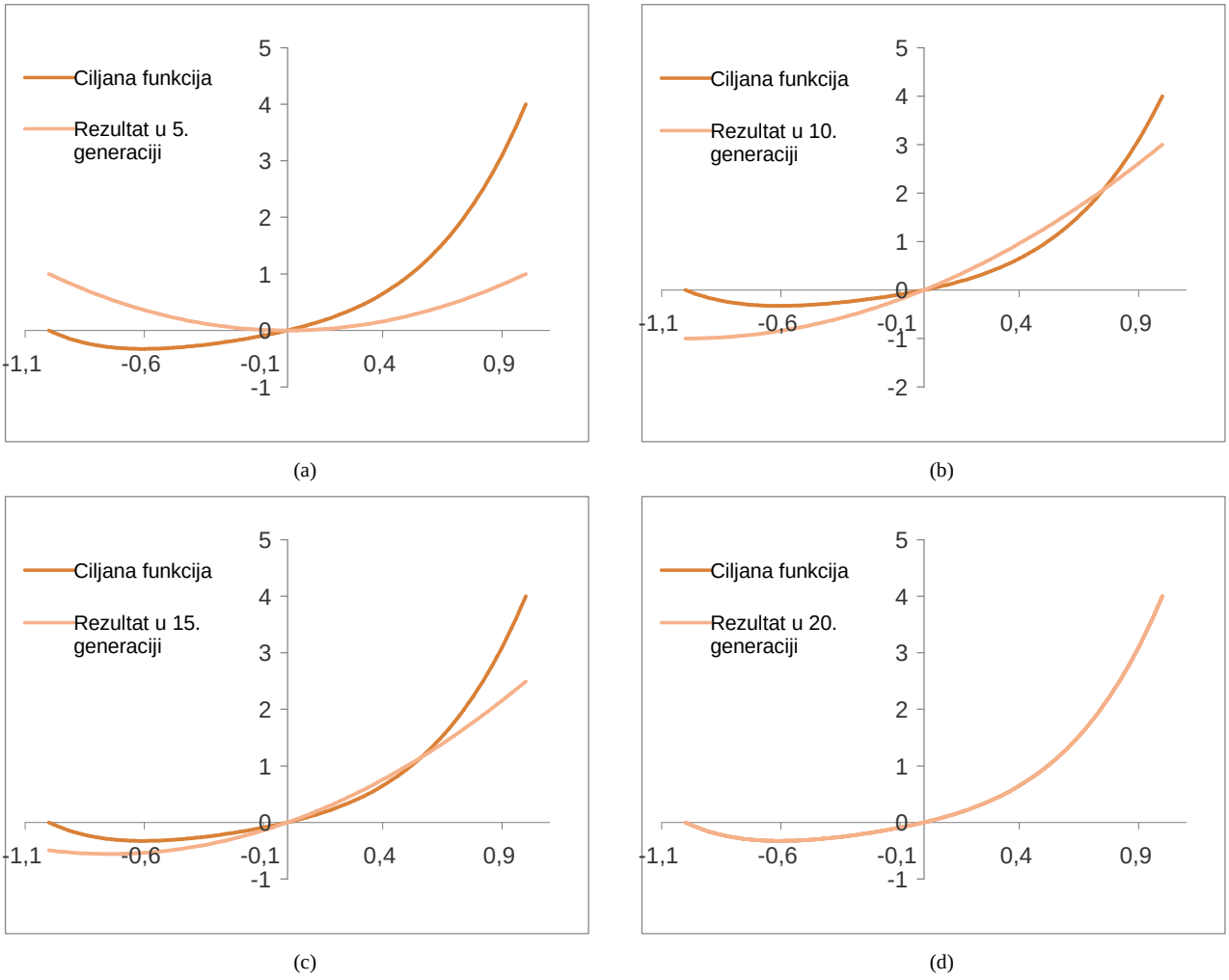
Simbolička regresija je najjednostavniji primjer korištenja genetskog programiranja u praksi. Neka su nam poznati parovi točaka *X* i *f(x)*. Potrebno je pronaći funkciju f koja vrijedi za sve zadane parove. Neka je ta funkcija $f(x) = x^4 + x^3 + x^2 + x$.

Odabir završnih i nezavršnih znakova je preduvjet rješavanja ovog problema genetskim programiranjem. U ovom slučaju za nezavršne znakove odabrat ćemo +, -, * i /, a završni znakovi su *X* i bilo koja druga konstanta.

Primjer jednog izvođenja genetskog programa prikazan je na slici 3.7. Kao što je vidljivo sa slike, genetski program postupno pronalazi sve bolja rješenja koja odgovaraju ciljanoj funkciji. U najvećem broju slučajeva ovaj genetski program uspješno pronalazi funkciju koja je identična ili vrijednosno vrlo slična ciljanoj funkciji.

Korištenje genetskog programiranja za rješavanje problema simboličke regresije pokazalo se kao izvrstan izbor. Genetsko programiranje u mogućnosti je pronaći odgovarajuću jedinku u vrlo

kratkom vremenu i uz malen utrošak računalnih resursa. Optimalne jedinke bilo je moguće pronaći u populacijama veličine 100 jedinki koje su se evoluirale u manje od 50 generacija.



(a) (b)

(c) (d)

Slika 3.7. Pronalazak funkcije $f(x) = x^4 + x^3 + x^2 + x$ genetskim programiranjem.

Na slikama je prikazan tijek jednog pokretanja genetskog programa, od početka (a), pa sve do pred sam kraj optimiranja (d). Slika (d) ujedno prikazuje najbolje rješenje koje je identično ciljanoj funkciji. Funkcije po generacijama su navedene u tablici 3.1.

Tablica 3.1. Dobivene vrijednosti ciljane funkcije po generacijama

Generacija	Izraz
5.	$f(x) = x^2$
10.	$f(x) = x^2 + 2$
15.	$f(x) = 0.49x + x^2 + x$
20.	$f(x) = x^4 + x^3 + x^2 + x$

3.7.1. Ostale primjene

Osim simboličke regresije GP je moguće koristiti u cijelom nizu drugih problema. Prema [Koz06], u rješavanju čak 36 problema korištenjem genetskog programiranja ostvareni su rezultati koji se mogu usporediti s rezultatima što su ih dobili ljudi. Zbirni podaci prikazani su u tablici 3.2.

Tablica 3.2. Rezultati ostvareni genetskim programiranjem, prilagođeno prema [Koz06]

Svojstvo	
Broj problema u kojima su rezultati usporedivi s rezultatima ljudskog rada	36
Broj rezultata koji su jednaki rezultatima poznatih izuma iz 20. stoljeća	15
Broj rezultata koji su jednaki rezultatima poznatih izuma iz 21. stoljeća	6
Broj rezultata koji se mogu kvalificirati za novi patent	2

Većina ovih otkrića su razni algoritmi i protokoli za rješavanje matematičkih i računalnih problema. Od zanimljivijih primjena treba navesti:

Programiranje robota koji igraju nogomet [Koz03]
Programiranje robota sa šest nogu [Jak05]
Otkriće NAND spoja tranzistora [Koz03]
Otkriće spoja s povratnom vezom [Koz03]
Raspoređivanje poslova na strojevima [Jak05]
Predviđanje i raspoznavanje [Jak05]
Potpora računalnim sustavima (otkrivanje nepravilnosti u radu) [Jak05]

3.8. Razlike između genetskih algoritama i genetskog programiranja

U ovom poglavlju će biti uspoređene dvije evolucijske metode (genetsko programiranje i genetski algoritmi) koje su korištene u ovom radu. U tablici 3.3. GA i GP su uspoređeni po najvažnijim svojstvima.

Tablica 3.3. Razlike genetskih algoritama i genetskog programiranja, prilagođeno prema [Koz03]

Svojstvo	Genetski algoritam	Genetsko programiranje
Jedinka	Jedinka je niz znakova određene veličine. Sve jedinke imaju jednaki broj znakova.	Jedinka je stablo koje se sastoji od nezavršnih i završnih znakova (skupovi se određuju na početku).
Inicijalno popunjavanje gena	Slučajno odabrana vrijednost iz zadane abecede dozvoljenih znakova	Slučajno kreirana kompozicija nezavršnih i završnih znakova
Izmjena strukture	Reprodukcija, križanje i mutacije	Reprodukcija, križanje i mutacije
Uvjet zaustavljanja	Broj generacija, broj generacija bez poboljšanja jedinke ili neka vrijednost funkcije dobrote	Broj generacija, broj generacija bez poboljšanja jedinke ili neka vrijednost funkcije dobrote
Rješenje	Najbolja pronađena jedinka	Najbolja pronađena jedinka

Iz prethodne je tablice vidljivo da se po najvažnijim obilježjima ove dvije tehnike ne razlikuju previše. Sitnije su razlike vidljive kod definicije parametara ovih dviju metoda. Za genetsko je programiranje tako potrebno definirati i točan način na koji se kreira inicijalna. GP također ima definirana ograničenja za jedinke koje sudjeluju u populacijama. Jedinke mogu biti različite veličine (više ili manje čvorova u stablu), stoga je potrebno unaprijed ograničiti maksimalni broj čvorova stabla.

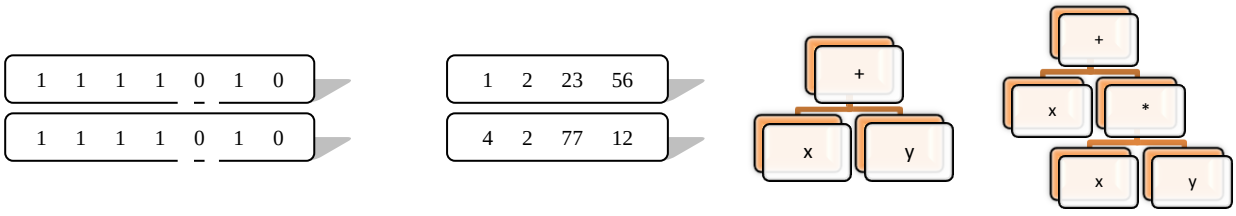
Najvažnije razlike opisane su u sljedećim potpoglavljima.

Prikaz jedinke

Već je prethodno naglašeno da se jedinke GA i GP razlikuju. Na slici 3.8. su radi potpunosti prikazane jedinke GA i GP. Slika prikazuje dva para GA jedinki (jedan sa binarnim genima, a drugi s cjelobrojnim genima) te jedan par GP jedinki. Najvažnije razlike ovih parova su:

GA jedinke iz oba para imaju jednak broj gena. GP jedinke su različite veličine.

GA jedinke su kodirane kao niz gena, a GP jedinke kao stablo.



(a) (b) (c)

Slika 3.8. Jedinke evolucijskih algoritama

Na prethodnoj slici prikazane su jedinke evolucijskih algoritama.

Na slici (a) prikazane su dvije jedinke genetskog algoritma. Obje jedinke su jednake duljine, sastavljene od niza bitova.
Na slici (b) također su prikazane dvije jedinke genetskog algoritma. Jedinke su jednake duljine, svaki gen je prikazan kao cijeli broj.
Na slici (c) prikazane su jedinke za genetski program. Jedinke nisu jednake duljine, prikazane su u obliku stabla, a prikazuju matematički izraz zapisan u poljskoj (postfix) notaciji.

Način optimiranja i područje primjene

Postojanje samo jedne instance problema nekog tipa je dovoljno za primjenu genetskog algoritma. Ako pretpostavimo da je genetski algoritam napisan bez grešaka, da smo odabrali ispravan način prikazivanja jedinki i ispravan način određivanja dobrote pojedine jedinke, možemo očekivati da će naš GA već nakon jednog pokretanja dati dobre rezultate. Kod složenijih problema bit će potrebno provesti podešavanje parametara GA kako bi se pronašla najbolja konfiguracija parametara za zadani problem.

Prije početka optimiranja, potrebno je postaviti i uvjete zaustavljanja. Nakon što se optimiranje dovrši, ispravnost rješenja može se provjeriti uspoređivanjem s poznatim rezultatom neke druge metode.

Općenito gledajući, GP bi se mogao koristiti na isti način kao i GA. Međutim, budući da je rezultat genetskog programiranja struktura (program ili neki izraz), takvo rješenje možemo primijeniti i na druge probleme toga tipa. Upravo zato je potrebno ostvariti neke preduvjete kako bismo mogli pronaći program koji rješava problem u općenitom slučaju.

Za uspješno rješavanje nekog tipa problema u općem slučaju potreban nam je dovoljan broj problema toga tipa s poznatim točnim rješenjem koje je dobiveno nekom drugom metodom. Taj skup problema s poznatim rješenjem dijeli se na dva podskupa: skup za učenje i skup za provjeru. GP se može primijeniti i na probleme bez poznatog rješenja. U tom slučaju je potrebno odabrati odgovarajući način ocjenjivanja dobrote na skupu za učenje.

GP algoritam mora pronaći takav izraz koji za sve elemente iz skupa za učenje daje što bolje rješenje. Proces se odvija na sljedeći način (slika 3.9.):

```
Za svaku jedinku iz populacije
    Za svaki problem Pi iz skupa za učenje
            Izračunaj dobrotu Di za problem Pi
            Pribroji dobrotu Di ukupnoj dobroti D
    Vrati ukupnu dobrotu D
```

Slika 3.9. Izračun dobrote za sve probleme iz skupa za učenje

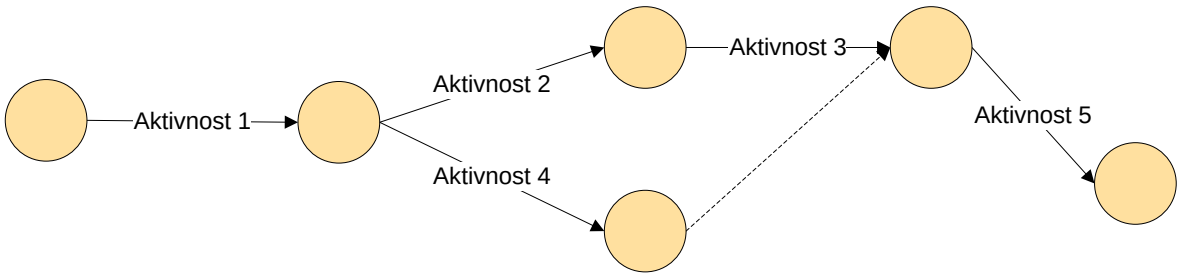
Dobrota jedne jedinke iz populacije utvrđuje se tako da se program pokrene za sve probleme koji su elementi skupa za učenje.

Nakon što se genetski program zaustavi (zadovoljeni su postavljeni uvjeti zaustavljanja), najbolje rješenje se testira na skupu za provjeru. Cijeli proces se ponavlja dok nismo zadovoljni s rezultatima koje dobivamo na skupu za provjeru. Program koji se dobije kao rješenje je dobar ako daje dobre rezultate i na skupu za učenje i na skupu za provjeru.

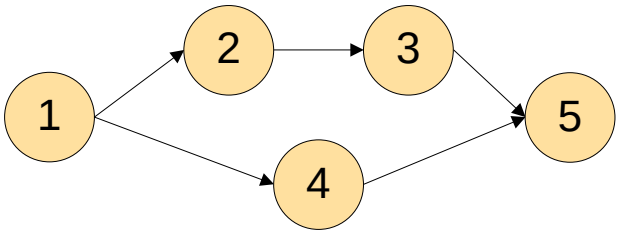
4. Mrežno projektno planiranje

Općenito se projekti mogu prikazivati usmjerenim dijagramom koji sadržava i podatke o dodatnim svojstvima što opisuju jednu aktivnost. Takav se dijagram naziva mrežnim planom. Postoje dva načina kojima je moguće prikazati projektne aktivnosti na dijagramu (slika 4.1.):

Aktivnosti na grani
Aktivnosti na čvoru



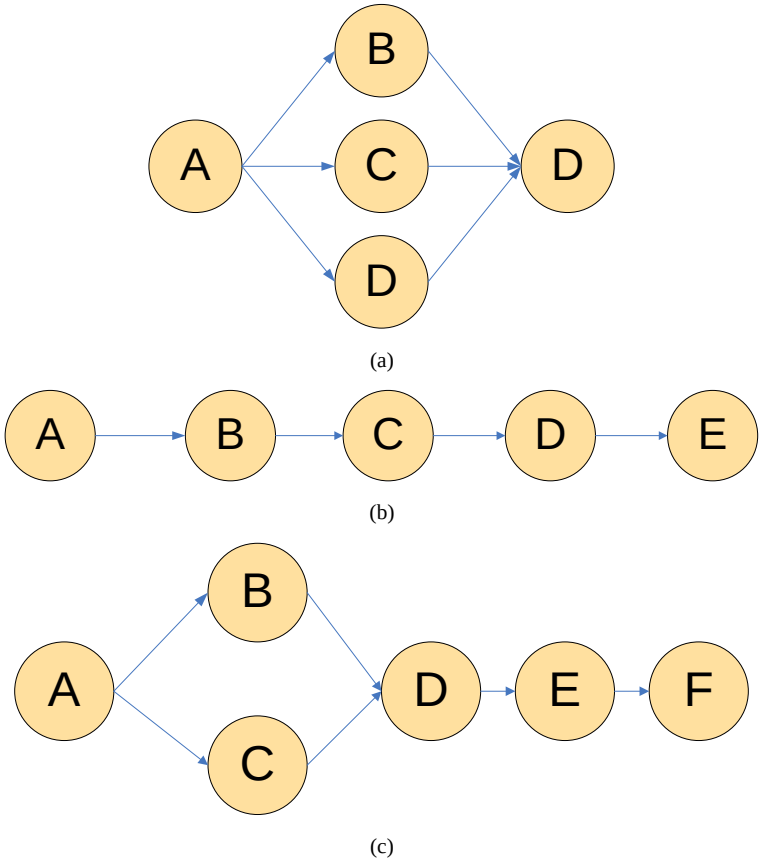
(a)



(b)

Slika 4.1. Prikaz istog mrežnog plana kao: (a) – aktivnosti na grani i (b) aktivnosti na čvoru

U mreži se, usmjerenim granama označava tijek aktivnosti.



Slika 4.2. Primjeri nekoliko mrežnih planova sa različitim mogućim načinima izvoĎenja aktivnosti.

Na slici 4.2. prikazani su različiti mogući načini izvoĎenja aktivnosti:

Slika (a) - Aktivnosti B, C i D mogu se izvoditi paralelno
Slika (b) - Aktivnosti B, C i D moraju se izvoditi serijski
Slika (c) - Aktivnosti B i C mogu se izvoditi paralelno, a aktivnosti D i E serijski

Za svaki je mrežni plan definirano nekoliko pravila kojih se treba pridržavati:

A. Mreža treba imati jedinstven početni čvor
B. Za mrežu mora postojati jedinstven završni događaj
C. Svaka je aktivnost opisana samo jednim lukom
D. Ne postoje dvije aktivnosti koje imaju isti početni i završni događaj

4.1. Tradicionalne metode za analizu mrežnih planova

U ovom poglavlju bit će opisane dvije najpoznatije metode za analizu mrežnih projektnih planova:

Metoda kritičnog puta – (engl. *Critical path method* – CPM)
PERT

Obje su metode nastale sredinom prošlog stoljeća za potrebe poboljšanja u industriji i razvoju naoružanja. Budući da se radilo o vrlo skupim projektima koji nisu smjeli kasniti, bile su potrebne metode za njihovo optimiranje.

U ovom poglavlju ukratko će biti opisane ove metode i njihove mogućnosti kako bi se kasnije mogla napraviti usporedba s mogućnostima algoritama ostvarenih u ovom radu.

4.1.1. Optimiranje trajanja projekta – CPM metoda

Pretpostavimo generalizaciju jednostavnog projektnog modela i definirajmo da su trajanja pojedinih aktivnosti njegove varijable. Vremena trajanja u tom bi se modelu mogla mijenjati dodavanjem rada ili kapitala pojedinoj aktivnosti. Ulaganjem određenih sredstava u tom bi se slučaju, moglo skratiti vrijeme trajanja projekta.

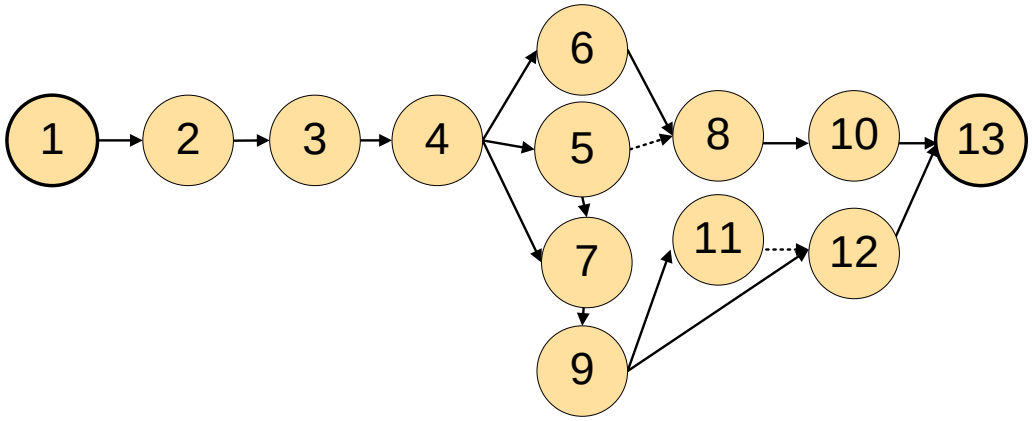
Neka je definiran projekt *P* za koji vrijedi sljedeće:

A. Trajanje svake aktivnosti je linearna funkcija utrošenih sredstava
B. Dozvoljena trajanja pojedinih aktivnosti definirana su gornjim i donjim granicama

Za projekt *P* tada je moguće definirati razmjenu vrijeme/troškovi (engl. *time/cost trade-off*) kako je to prikazano na slici 4.3. Za svaku od aktivnosti definirane su sljedeće vrijednosti (tablica 4.1.):

m – minimalno moguće trajanje
M – maksimalno moguće trajanje
t – trajanje aktivnosti
C_0 – ukupna cijena aktivnosti u slučaju minimalnog trajanja
c – cijena trajanja aktivnosti za svaku vremensku jedinicu koja je dulja od minimalnog trajanja
K – ukupna cijena trajanja aktivnosti za odabrano trajanje
D – dodatni troškovi za jednu vremensku jedinicu produljenja projekta

Optimiranje jednostavnih mrežnih problema, ako pretpostavimo postojanje linearne zavisnosti, može se jednostavno obaviti odgovarajućim linearnim modelom [Kal96]. Jednostavno optimiranje može kao cilj imati ukupno skraćenje trajanja projekta angažiranjem dodatnih resursa (npr. uvođenje druge smjene ili više strojeva). Najčešće će se „ubrzavati“ one aktivnosti koje se nalaze na kritičnom putu koji usporava odvijanje svih ostalih aktivnosti. Uvođenje dodatnih resursa automatski povlači i dodatne troškove. Zbog toga treba naći optimalno rješenje, kod kojeg će dobrobiti (prihodi) od skraćenja ukupnog trajanja biti veće nego cijena dodatnih resursa. Skraćenje vremena trajanja projekta vrlo je kompleksno jer se ne smije gledati samo skraćenje vremena na jednoj grani, nego se mora gledati projektni plan u cjelini. S obzirom na međusobnu isprepletenost elemenata projektnog plana, samo se optimiranjem cijelog plana može pronaći najbolje rješenje. Na slici 4.3. je primjer projekta čije su aktivnosti prikazane grafom .



Slika 4.3. Mrežni problem prikazan grafom.

Bridovi grafa koji su označeni punim linijama predstavljaju stvarne aktivnosti. Bridovi grafa označeni isprekidanom linijom predstavljaju fiktivne aktivnosti.

U tablici 4.1. dani su podaci za ovaj mrežni plan. Podatci uključuju: trajanje pojedine aktivnosti, mogućnosti skraćenja i trošak smanjenja po jedinici vremena. Trošak projekta je 200 novčanih jedinica po jedinici vremena.

Cilj optimiranja je pronaći ono rješenje koje za zadani mrežni plan daje najmanje troškove.

Tablica 4.1. Aktivnosti jednostavnog mrežnog plana

Aktivnost (i-j)	Normalno trajanje	Skraćeno trajanje	Jedinični trošak skraćenja
1-2	2	1	300
2-3	4	3	200
3-4	10	7	300
4-5	4	2	350
4-6	6	4	150
4-7	7	5	100
5-7	5	3	150
5-8	-	-	-

Aktivnost (i-j)	Normalno trajanje	Skraćeno trajanje	Jedinični trošak skraćenja
6-8	7	4	200
7-9	8	6	100
8-10	9	5	100
9-11	4	3	100
9-12	5	3	150
10-13	2	1	100
11-12	-	-	-
12-13	6	3	133.3

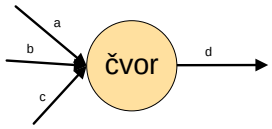
Cilj optimiranja je završiti projekt u što kraćem roku sa što manjim troškovima. U tom je slučaju funkcija cilja:

$$\min Troskovi \quad f\ t_n \qquad \underset{i,j}{} c_{ij}(n_{ij} \quad t_{ij}) \tag{2.1}$$

Za opisani problem ograničenja su sljedeća:

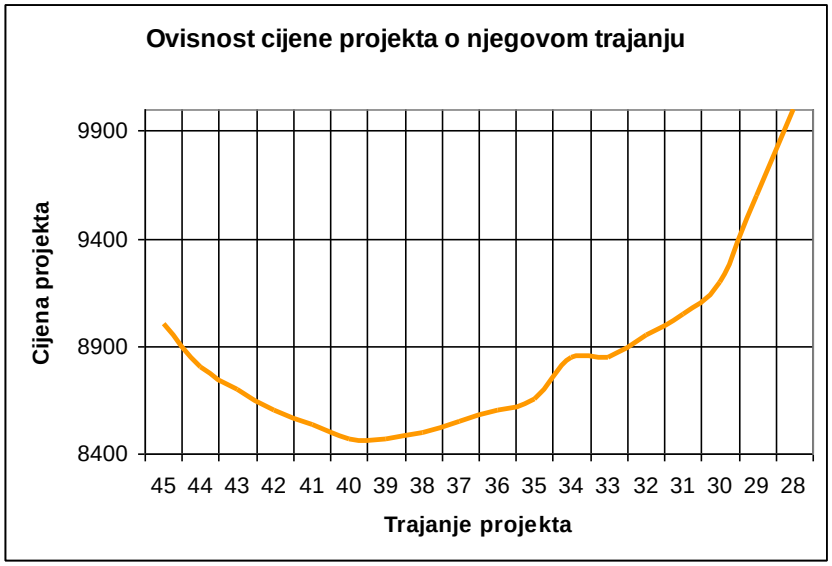
A. Vrijeme trajanja pojedine aktivnost je u intervalu [Skraćeno trajanje, Normalno trajanje]

B. Aktivnost koja slijedi nakon čvora ne može početi prije nego završe sve aktivnosti koje ulaze u taj čvor. (Slika 4.4.)



Slika 4.4. Aktivnost d ne može početi prije nego što završe aktivnosti a, b i c

Ako se pretpostavi linearna zavisnost između skraćenja trajanja i povećanja troškova, problem je moguće riješiti linearnim programiranjem. Pritom se dobivaju rezultati prikazani na slici 4.5.



Slika 4.5. Prikaz zavisnosti troškova projekta o trajanju projekta

Troškovi bi bili najmanji da projekt traje između 39 i 40 vremenskih jedinica.

4.1.2. PERT

Metoda PERT je druga najpoznatija klasična metoda analize projektnih trajanja. Za razliku od CPM-a, u ovoj metodi se pretpostavlja da trajanje pojedinih aktivnosti projekta nije unaprijed poznato. Zato se definiranim izrazima za svaku od aktivnosti izračunavaju tri vrijednosti:

1. Optimistično trajanje - *a*
2. Normalno trajanje - *m*
3. Pesimistično trajanje - *b*

Budući da se radi o procjeni, vrijeme trajanja može biti bilo gdje između *a* i *b*.

Za analizu ovom metodom potrebno je imati definiran mrežni plan aktivnosti. Mrežni plan zatim crtamo kao mrežu s aktivnostima u čvorovima. Primjer jednog čvora prikazan je na slici 4.6.

ESx	Tx	EFx
	x	FFLx
LSx	TFLx	LFx

Slika 4.6. Primjer čvora za metodu PERT

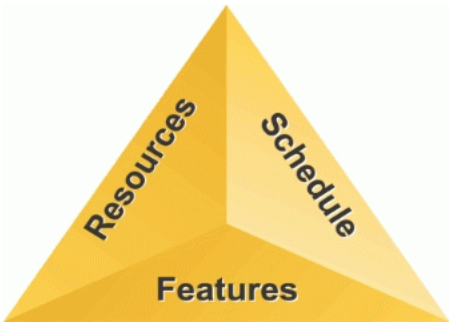
Oznake jednog čvora su sljedeće [Kal96]:

X – identifikacija aktivnosti
Tx – trajanje aktivnosti
ESx – najraniji početak aktivnosti
EFx – najraniji završetak aktivnosti
LSx – najkasniji početak aktivnosti
LFx – najkasniji završetak aktivnosti
TLFx – ukupna rezerva koliko se aktivnost može produljiti da projekt ne zakasni
FFLx – slobodna vremenska rezerva između najranijeg početka sljedbenika aktivnosti i najranijeg završetka aktivnosti

Prema informacijama u mrežnom planu kreira se PERT dijagram, u kojem se za svaki čvor upisuju informacije o trajanjima prema prethodnoj slici.

5. Projekti s ograničenim sredstvima

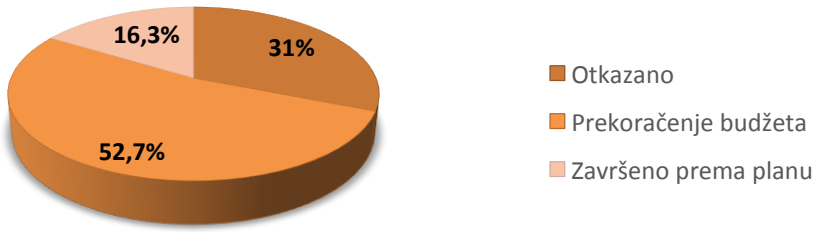
Projekt je privremena aktivnost koja se provodi kako bi se napravio neki proizvod, usluga ili neki drugi rezultat [PMB04]. To je dakle svaka aktivnost koja ima točno određen početak i kraj, te definiran rezultat. Disciplina upravljanja projektima je primjena znanja, vještina i alata kako bi projektne aktivnosti dale očekivani rezultat. Voditelj projekta mora uspješno balansirati trokutom koji se sastoji od sredstava (engl. *resources*), rezultata projekta (na slici prikazano kao *features*) i vremenskog plana (engl. *schedule*). Trokut je prikazan na slici 5.1. (slika je preuzeta iz [MSF99].) Sredstva mogu biti: ljudi, strojevi, novac itd. Ako je cilj projekta izraditi računalni program, rezultati su svojstva proizvoda koji isporučujemo (engl. *features*).



Slika 5.1. Trokut sredstva – rezultati - vrijeme

Najnovije istraživanje Standish Group, koje je provedeno tijekom 10 godina na 40.000 IT projekata, pokazalo je sljedeće poražavajuće rezultate (Slika 5.2.) [Car05]:

31% projekata je otkazano prije predviđenog završetka
U 52,7% slučajeva prekoračenje troškova je iznosilo oko 200%
16,3% završeno je u roku i okviru budžeta



Slika 5.2. Pregled uspješnosti IT projekata

Ovi poražavajući rezultati trebali bi nam biti poticaj da u svakodnevno upravljanje projektima uvedemo nove metode i tehnike. U ovom će se radu stoga pokušati pronaći optimalni rasporedi zadanih projekata.

Metode koje su opisane u prethodnom poglavlju (CPM i PERT) predstavljaju jako dobru teorijsku podlogu za pronalaženje optimalnih rasporeda. Međutim, niti jedna od tih metoda ne uzima u obzir ograničenja koja mogu postojati za sredstva u jednom projektu, kao ni očekivani (najčešće unaprijed zadan) kraj projekta [Sri03]. Najveći problem koji susrećemo u svakodnevnom radu sa projektima je balansiranje trokutom sa slike 5.1. Voditelj projekta mora dakle sa zadanim rezultatima i danim sredstvima dovršiti projekt u dogovorenom vremenu. Upravo nemogućnost uspješnog rješenja toga problema s tri varijable uzrok je propasti većine projekata.

Cilj optimiranja je pronaći optimalno rješenje za jednu stranu trokuta. Uz pretpostavku da su rezultati i vremenski rokovi unaprijed zadani i nepromjenljivi, za zadane projektne rasporede pokušat će se pronaći optimalan raspored pomoću sredstava koja su nam dostupna.

Najpoznatiji rezultati koji su dostupni u ovom području su opisani u radovima Roberta Kolischa [PSP06]. On i njegovi suradnici su izradili bazu podataka koja sadržava 2040 različitih projektnih planova sa po 30, 60, 90 i 120 aktivnosti. Heuristički algoritmi prezentirani u njegovim radovima i radovima drugih autora su izrazito brzi. Međutim, oni ne mogu pronaći optimalne rasporede za projekte sa više od 60 aktivnosti. Istraživanja su također pokazala da komercijalni računalni proizvodi koji sadržavaju neke algoritme za optimiranje, pronalaze optimalne projektne rasporede s odstupanjem od 4.3% do čak 9.8%. Projekti koji su optimirani u tom istraživanju imali su najviše 30 aktivnosti. Ovi izrazito loši rezultati za vrlo malene projekte pokazuju kako postoji potreba da se nekom drugom tehnikom pronađe način brzog, točnog i pouzdanog optimiranja projekata. [Har98]

Pronalaženje optimalnog rasporeda za projekt s ograničenim sredstvima je NP težak problem. Ako pretpostavimo da je N broj aktivnosti i r broj sredstava koja se koriste na projektu prostor mogućih rješenja je jednak N^r. Dakle za projekt koji ima 30 aktivnosti i samo 2 sredstva koja se koriste, broj mogućih rješenja je $2^{30} = 1\,073\,741\,824 \approx 10^9$. Ako uzmemo u obzir da je projekt s 30 aktivnosti vrlo malen, a da već za njegovo optimiranje postoji tako velik broj kombinacija, za veće i realnije projekte broj kombinacija bio bi još veći. Broj kombinacija za testne projekte prikazan je u tablici 5.1.

Tablica 5.1. Veličina prostora rješenja za projekte, čija su svojstva jednaka svojstvima projekata koji su optimirani u ovom radu

Broj aktivnosti	Broj sredstava	Broj kombinacija
30	2	1073741824
30	4	1,15292E+18
30	10	1E+30
60	2	1,15292E+18
60	4	1,32923E+36
60	10	1E+60
90	2	1,23794E+27
90	4	1,5325E+54
90	10	1E+90
120	2	1,32923E+36
120	4	1,76685E+72
120	10	1E+120

5.1. Konceptni model

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 $J=\{0,1,\ldots,n,n+1\}$ i skup sredstava $K=\{1,2,\ldots,k\}$. 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 izvoĎenje 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.
>
> IzvoĎenje aktivnosti može započeti samo ako je dostupno dovoljno sredstava koja su potrebna da bi se aktivnost izvršavala.

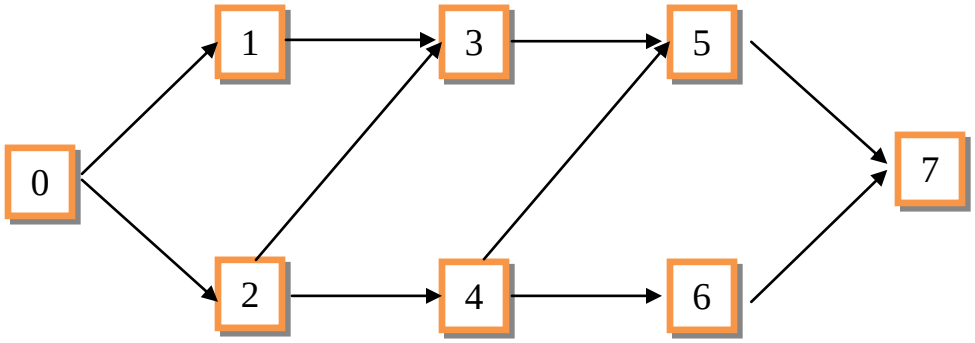
Tijekom izvoĎenja, jednoj je aktivnosti *j* potrebno $r_{j,k}$ jedinica sredstva $k \in K$ tijekom svakog menskog trenutka u kojem se izvodi. Trajanje aktivnosti je d_j, a čim ona započne ne smije se prekidati do završetka.

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.

Na slici 5.3. prikazan je maleni projekt koji odgovara gore navedenoj definiciji. Duljine trajanja pojedinih aktivnosti i zauzeće resursa prikazani su u tablici 5.2.



Slika 5.3. Mrežni plan jednostavnog projektnog plana

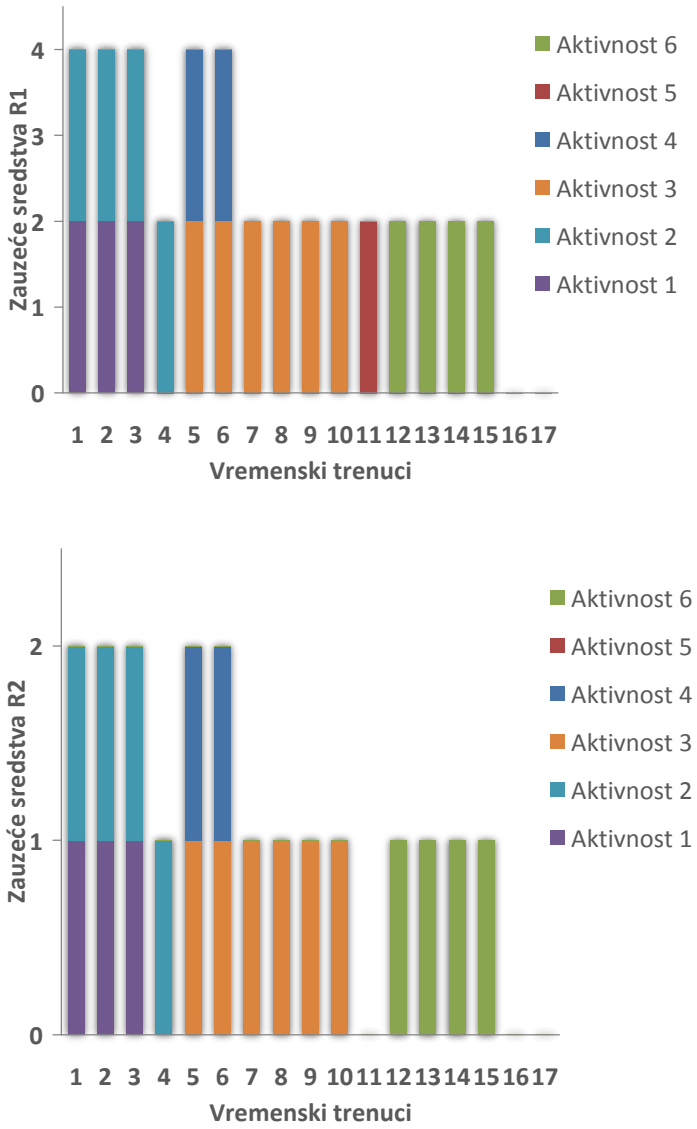
Tablica 5.2. Aktivnosti projekta, njihova trajanja i zauzeće sredstava projekata

Aktivnost	Trajanje	Zauzeće sredstva 1	Zauzeće sredstva 2
0	0	0	0
1	3	2	1
2	4	2	1
3	6	2	1
4	2	2	1
5	1	3	0
6	4	3	1
7	0	0	0

Kao što je vidljivo sa slike i iz tablice, projekt se sastoji od 6 stvarnih aktivnosti te početne i završne aktivnosti. Projekt se koristi dvjema vrstama sredstava koje su dostupne samo u ograničenim količinama. Sredstvo *R1* dostupno je sa četiri jedinice u jednom vremenskom trenutku, a sredstvo tipa *R2* dostupno je s dvije jedinice po vremenskom trenutku.

Optimiranjem bismo trebali dobiti optimalno trajanje projekta od 15 vremenskih jedinica.

Na slici 5.4. prikazano je jedno moguće rješenje optimalnog raspoređa korištenja resursa na projektu:



Slika 5.4. Prikaz optimalnog rješenja raspoređa aktivnosti i sredstava

Na gornjoj je slici vidljivo da u predstavljenom optimalnom rješenju postoji nekoliko trenutaka s „praznim hodom“. Primjerice u četvrtom trenutku, nakon završetka prve aktivnosti, aktivnost 4 ne može započeti jer nema dovoljno dostupnih sredstava *R1* i *R2*. Zbog definiranog ograničenja da se aktivnost ne smije prekidati, aktivnost 2 ima prednost pred aktivnošću broj 4.

Slična se situacija može zamijetiti u 11. trenutku. Aktivnost broj 5 ima veći prioritet od aktivnosti broj 6 i ona je zauzela sredstvo broj 1 onemogućivši time da aktivnost broj 6 započne.

6. Primjena genetskih algoritama u raspoređivanju aktivnosti

6.1. Pregled radova

Više je autora [Bar96, Har98, Kem03, Kim05, Men05] korištenjem evolucijskih algoritama rješavalo problem optimiranja projekata s ograničenim sredstvima. U više radova objavljeni su mogući načini optimiranja projekata s ograničenim sredstvima genetskim algoritmom. U ovom poglavlju ukratko je opisano kako su drugi autori riješili problem. Najvažnija razlika između opisanih metoda je u načinu prikaza jedne jedinke tj. njezina kromosoma.

6.1.1. Kodiranje po prioritetu

U [Men05] autori su opisali ostvareni genetski algoritam za rješavanje ovoga problema. Autori su jedinku genetskog algoritma kodirali kao niz binarnih brojeva (slika 6.1.).

$$kromosom = \{gen_1, gen_2, \ldots, gen_n, gen_{n+1}, \ldots, gen_{2n}\}$$

Slika 6.1. Prikaz kromosoma za projekt od n aktivnosti. Kodiranje po prioritetu.

Prioritet aktivnosti *j* pri tome je jednak (slika 6.2.):

$$prioritet_j = gen_j$$

Slika 6.2. Dekodiranje prioriteta jedne aktivnosti.

Izračunavanje prioriteta dalje se provodi kroz niz formula koje su autori dobili testiranjem svojeg algoritma. Također su uveli dodatne gene od *n+1* do *2n* koji im služe za izračunavanje kašnjenja svake od aktivnosti. Kodiranje prioritetima također je korišteno i u [Har98].

6.1.2. Kodiranje po rasporedu

U [Har98] je opisan i drugi način kodiranja jedinki. Kromosom jedne jedinke zapisuje se kao niz od *n* gena (za projekt od *n* aktivnosti). U svakom genu zapisan je broj aktivnosti koja će se izvoditi. Primjer jednog kromosoma prikazan je na slici slika 6.3.

$$kromosom_i = \{2, 3, 7, 5, 4, 1, 6\}$$

Slika 6.3. Kromosom za jedinku kada u kromosom zapisujemo slijed obavljanja

Kod kromosoma prikazanog na gornjoj slici geni predstavljaju aktivnosti toga projekta. U prikazanom slučaju aktivnosti projekta će se izvoditi redoslijedom koji je zapisan u kromosomu. Prva će se izvršiti aktivnost s brojem 2, a posljednja aktivnost s brojem 6.

6.1.3. Kodiranje po pravilima

Osim gore navedenih načina kodiranja u [Har98] je opisan i napredniji način kodiranja kromosoma. Kao i kod prethodnih primjera kromosom se sastoji od *n* gena za projekt od *n* aktivnosti. Svaki gen je jedna vrijednost iz skupa u tablici 6.1.

Tablica 6.1. Opis pravila za preslagivanje prioriteta

Pravilo	Opis
LFT	Zadnje moguće vrijeme završetka aktivnosti
LST	Zadnje moguće vrijeme početka
MTS	Ukupan broj aktivnosti sljedbenica
MSLK	Maksimalni hod
WRUP	Težinsko korištenje resursa i broj prethodnika
GRPW	Najveća težina

Formule za izračunavanje ovih vrijednosti opisane su u [Har98]. U kromosom se za svaku aktivnost upisuje jedno nasumce odabrano pravilo iz skupa pravila. Prilikom dekodiranja kromosoma za svaku se aktivnost izračunava prioritet. Zatim se aktivnosti raspoređuju poštujući mrežni plan i prioritete.

6.2. Prikaz jedne jedinke

Raspoređivanje aktivnosti u projektima s ograničenim sredstvima zapravo je „borba“ aktivnosti za resurse kojih nema dovoljno. Da bi se ovakav projekt mogao izvesti do kraja, u trenutku kada se resursi dodjeljuju aktivnostima mora postojati točno utvrđeno pravilo koja od dviju aktivnosti ima prednost pri dodjeli resursa. Prema tome, očito je potrebno da za svaku aktivnost postoji točno određen prioritet kako bi se u svakom trenutku moglo utvrditi koja od dviju aktivnosti ima prednost kod dodjele resursa.

U ovom je radu kromosom jedne jedinke prikazan kao slijed od *n* gena, gdje je gen_i prioritet i-te aktivnosti. Radi pojednostavljenja rada s programom i dekodiranja kromosoma, u radu su geni kodirani kao cijeli brojevi. Primjer jednog kromosoma prikazan je na slici 6.4.

```
1, 3, 6, 4, 2, 5
```

Slika 6.4. Prikaz jedne jedinke

Na gornjoj slici prikazani kromosom se koristi za projekt sa 6 aktivnosti. Prva aktivnost ima najveći prioritet, a treća aktivnost najmanji. Ako se u nekom trenutku projekta za neki resurs budu natjecale prva i treća aktivnost, prva aktivnost će imati prednost pri dodjeli toga resursa.

Cjelobrojni prikaz kromosoma olakšava dekodiranje kromosoma, ali također sprečava dupliciranje vrijednosti prioriteta, tako da svaka aktivnost ima jedinstveni prioritet. Ako imamo projekt s 8 aktivnosti, znači da nam treba 6 razina prioriteta. Prioritete za početnu i završnu aktivnost nije potrebno izračunavati. U slučaju projekta sa 6 razina prioriteta moguće vrijednosti prioriteta su u intervalu [1, 6]. Pritom ne dolazi do ponavljanja vrijednosti iz toga skupa.

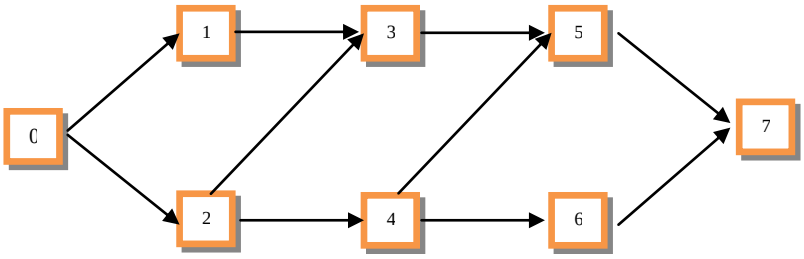
Ovakav način zapisivanja kromosoma također skraćuje trajanje računalne usporedbe prioriteta jer nema potrebe da se binarne vrijednosti pretvaraju u cjelobrojne, podešavaju gornje i donje granica i slično.

6.2.1. Ocjena dobrote jedinke

Nakon što genetski algoritam kreira novu jedinku potrebno je ocijeniti dobrotu te jedinke. Dobrota je trajanje projekta. Obzirom na to da je cilj ovog optimiranja pronaći onaj projektni raspored za koji je trajanje projekta najkraće, najbolja jedinka biti će ona za koju je trajanje projekta najkraće.

Izračunavanje dobrote započinje dekodiranjem kromosoma. Tijekom ovog procesa svakoj se aktivnosti dodjeljuje vrijednost njezina prioriteta. Nakon toga započinje virtualno izvršavanje projekta kako bi se ocijenila njegova dobrota. Potrebno je izvršiti sve aktivnosti kako bi se izračunalo trajanje cijelog projekta.

Izvršavanje počinje prvom aktivnošću, a budući da se radi o fiktivnoj aktivnosti, ona se odmah i izvrši. Na slici 6.5., koja prikazuje jednostavni mrežni plan, aktivnost 0 je početna aktivnost i ona će se odmah izvršiti.



Slika 6.5. Primjer jednostavnog projektnog plana.

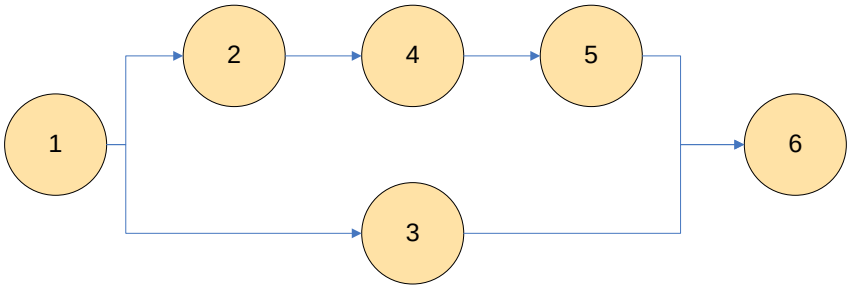
Nakon početne aktivnosti na red dolaze aktivnosti koje joj slijede. U našem slučaju radi se o aktivnostima 1 i 2. Ako pretpostavimo da je kromosom jedinke koju dekodiramo: 1, 3, 6, 4, 2, 5. Aktivnost čiji je iznos prioriteta manji ima prednost kod izvođenja. Aktivnost s iznosom prioriteta 1 ima najveći prioritet, aktivnost s prioritetom 6 najmanji prioritet. U primjeru aktivnost 2 ima veći prioritet od aktivnosti 1 (prioritet 3 > 1). Zbog toga aktivnost 2 ima prioritet pred aktivnosti 1 pri dodjeli resursa. Ona zauzima sve resurse koji su joj potrebni, a aktivnost 1 može započeti jedino ako je ima dovoljan broj resursa da bi se aktivnost mogla izvesti. Aktivnosti koje su započele izvode se sve dok se ne potroši vrijeme previđeno za izvođenje.

U ovisnosti o resursima koji su im potrebni za izvođenje i ukupnom broju dostupnih resursa moguće je da se aktivnost 2 i aktivnost 1 izvode paralelno ili da se izvode serijski.

Nakon što se izvođenje jedne aktivnosti dovrši, aktivnosti sljedbenice mogu započeti s izvođenjem. Na slici 6.5. vidi se da se nakon aktivnosti 2 može izvršavati aktivnost 4, dok se aktivnost 3 može izvršavati nakon aktivnosti 2 i aktivnosti 1. Kada aktivnost postane dostupna, ona se uspoređuje s drugim dostupnim aktivnostima, provjeravaju se prioriteti i ona s najvećim prioritetom od svih dostupnih aktivnosti dobije pravo pristupa resursima. Resursi se dodjeljuju dok ima dovoljno dostupnih resursa, a kada ponestane resursa čeka se sljedeći vremenski trenutak i tako sve do završetka projekta.

6.2.2. Optimiranje jednostavnog projekta genetskim algoritmom

Pretpostavimo da je neki jednostavni projekt zadan mrežnim planom na slici 6.6. Projekt se sastoji od 6 aktivnosti, a koristi 4 sredstva.



Slika 6.6. Primjer jednostavnog projekta prikazan mrežnim planom

Informacije o trajanjima aktivnosti i potrebnim resursima prikazane su u tablici 6.2.

Tablica 6.2. Informacije o zauzeću aktivnosti za projekt

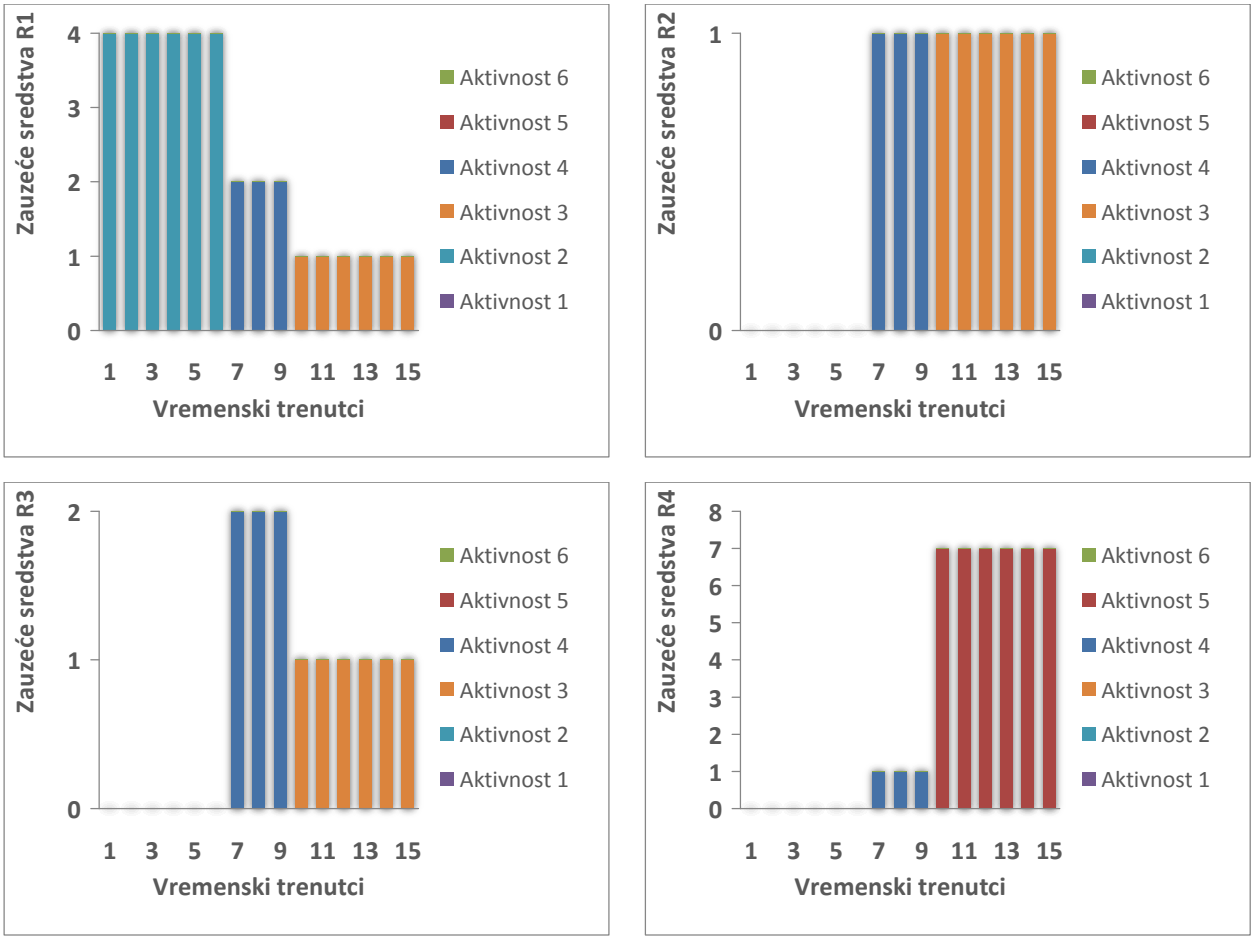
Aktivnost	Trajanje	Zauzeće sredstva 1	Zauzeće sredstva 2	Zauzeće sredstva 3	Zauzeće sredstva 4
1	0	0	0	0	0
2	6	4	0	0	0
3	6	1	1	1	0
4	3	2	1	2	1
5	5	0	0	0	7
6	0	0	0	0	0

Resursi *R1-R4* su dostupni u količinama: 4, 1, 2 i 7.

Nakon što se projekt optimira genetskim algoritmom dobiju se sljedeći prioriteti za aktivnosti (Slika 6.7.). Aktivnost čiji je iznos prioriteta manji ima prednost kod izvoĎenja.

```
2, 3, 5, 1, 0, 4
```

Slika 6.7. Prioriteti aktivnosti



Slika 6.8. Zauzeće resursa *R1-R4* za zadani projekt

Kao što je vidljivo sa slike 6.8. i s liste prioriteta, da bi se ubrzalo izvoĐenje projekta, aktivnost 3 je dobila manji prioritet od aktivnosti 4. Time je projekt skraćen jer se aktivnosti 3 i 5 na ovaj način mogu izvoditi paralelno.

6.3. Podešavanje parametara genetskog algoritma

Za ostvarivanje kvalitetnijih rezultata s ovim genetskim algoritmom, bilo je potrebno podesiti GA parametre. Već u prvim istraživanjima je utvrĐeno da je veličina populacije od 100 jedinki i 100 generacija dovoljan broj jedinki koje GA mora obraditi da bi pronašao odgovarajuće rješenje.

Najbolja konfiguracija parametara tražila se finim podešavanjem parametara mutacije. Općenito su se varirala dva parametra vjerojatnost mutacije jedinke p_j i vjerojatnost mutacije pojedinog gena p_g. Ostvareni rezultati prikazani su u tablicama 6.3. – 6.5. Testiranje se provodilo tako da se za sve kombinacije p_j i p_g genetski algoritam pokrenuo 15 puta, za jedan tipičan problem.

Tablica 6.3. Prosječna vrijednost trajanja projekta za različite kombinacije vjerojatnosti mutacije

p_i \ p_a	**0,002**	**0,004**	**0,01**	**0,02**	**0,04**	**0,06**
0,5	135,13	135,47	133,93	134,80	135,07	135,80
0,4	135,47	135,53	135,40	135,27	135,13	135,13
0,6	134,73	135,33	135,00	135,20	135,00	135,00

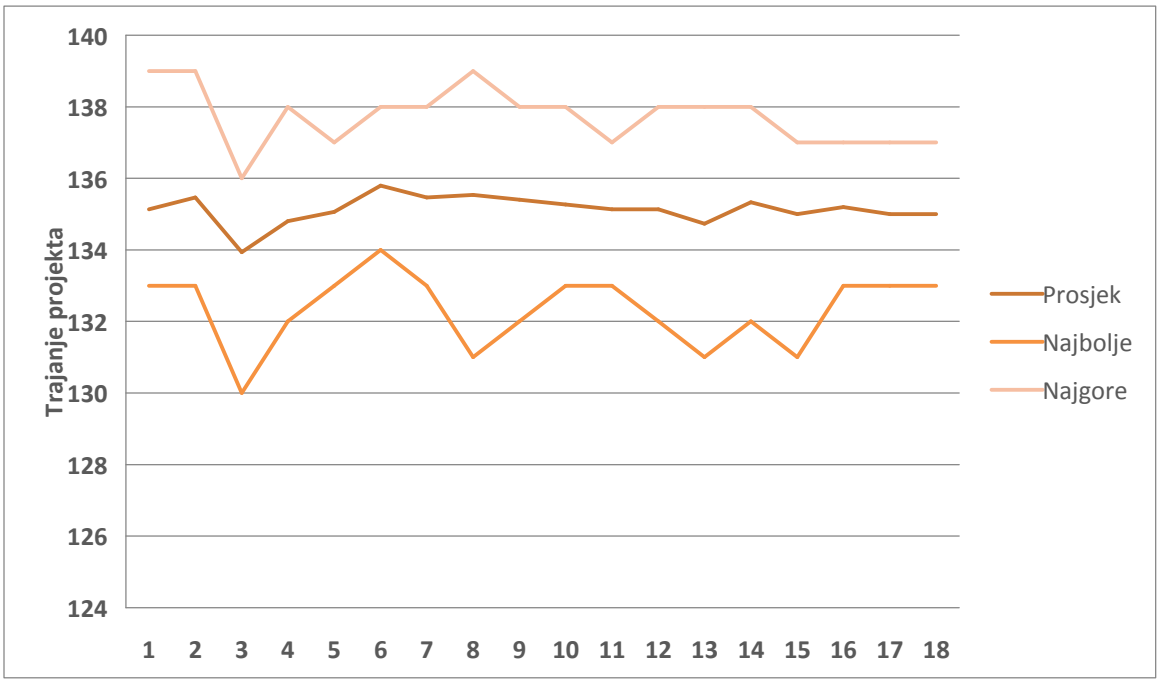
Tablica 6.4. Najmanja vrijednost trajanja projekta za različite kombinacije vjerojatnosti mutacije

p_i \ p_a	**0,002**	**0,004**	**0,01**	**0,02**	**0,04**	**0,06**
0,5	133	133	130	132	133	134
0,4	133	131	132	133	133	132
0,6	131	132	131	133	133	133

Tablica 6.5. Najbolja vrijednost projekta dobivena nakon provedena optimiranja za različite kombinacije vjerojatnosti mutacije

p_i \ p_a	**0,002**	**0,004**	**0,01**	**0,02**	**0,04**	**0,06**
0,5	139	139	136	138	137	138
0,4	138	139	138	138	137	138
0,6	138	138	137	137	137	137

Zbirni rezultati za ovog podešavanja parametara prikazani su na slici 6.9.



Slika 6.9. Trajanja projekata u ovisnosti o parametrima mutacije

Iz prethodnih tablica i sa slike 6.9. je vidljivo da se najbolji rezultati postižu kombinacijom parametara $p_m = p_j * p_g = 0{,}5 * 0{,}01 = 0{,}005$ (vrijednosti prikazane pod rednim brojem 3 na slici 6.9.). Također se dobri rezultati dobivaju u okolici te vrijednosti odnosno za $p_m = 0{,}004$ i $p_m = 0{,}006$.

6.3.1. Optimalni parametri genetskog algoritma

U tablici 6.6. dani su optimalni parametri izvođenja genetskog algoritma za problem optimiranja projekata opisan u ovom radu.

Tablica 6.6. Parametri genetskog algoritma.

Parametar	Vrijednost
Veličina populacije	100
Uvjeti zaustavljanja:	
Maksimalni broj generacija	1000
Maksimalni broj generacija bez poboljšanja najbolje jedinke	300
Vrsta GA	generacijska, M=50%
Način selekcije	3-turnirska
Vjerojatnost mutacije jedinke	0,5
Vjerojatnost mutacije jednog gena	0,01

Algoritam je ostvaren korištenjem biblioteke za evolucijsko računarstvo *Open Beagle* [OB06]. Broj generacija i veličina populacije su nešto manji nego što je uobičajeno. Testiranja su pokazala da se rezultati tek neznatno poboljšavaju s većim brojem generacija / jedinki.

6.4. Rezultati izvođenja

Ostvareni genetski algoritam primijenjen je na projekte iz baze podataka [PSP06]. Baza se sastoji od 2040 projekata sa po 30, 60, 90 i 120 aktivnosti. Autori baze podataka uspjeli su linearnim programiranjem pronaći optimalna rješenja samo za sve projekte od 30 aktivnosti. Za projekte s više aktivnosti poznata su samo neka rješenja.

Za potrebe testiranja genetskog algoritma nasumce je odabrano po 10 projekata za svaku grupu projekata klasificiranih po trajanju. Dobiveni rezultati prikazani su u tablici 6.7.

Tablica 6.7. Rezultati koji su dobiveni optimiranjem genetskim algoritmom za različite tipove projekata

Tip projekata	Prosječno kašnjenje projekata u odnosu na poznata optimalna rješenja
30 aktivnosti	1,36%
60 aktivnosti	3,73%
90 aktivnosti	1,92%
120 aktivnosti	8,20%

Rezultati ostvareni s GA ne odstupaju od optimalnih rješenja. Oni pokazuju da se za projekte do 90 aktivnosti može vrlo velikom vjerojatnošću tvrditi da će genetski algoritam pronaći optimalno rješenje u razumnom vremenu. Rezultati za projekte sa 120 aktivnosti su nešto lošiji, međutim

GA je najlošije rezultate postizao upravo s onim projektim a za koje je više autora dobilo različita optim alna rješenja m etodam a linearnog program iranja opisanim u [PSP06].

7. Primjena genetskog programiranja u raspoređivanju aktivnosti

Već su početni testovi koji su provedeni na najjednostavnijim projektima s manjim brojem aktivnosti (30 aktivnosti i 4 resursa), pokazali da genetski algoritmi u najvećem broju slučajeva daju optimalna rješenja. Ipak, optimiranje koje provode nije posve zadovoljavajuće. Trajanje optimiranja od početka do kraja za opisani slučaj može potrajati više od sata. U stvarnoj situaciji s većim brojem generacija i jedinki u populaciji optimiranje bi trajalo i znatno dulje. Sama duljina trajanja nije toliko problematična jer je prihvatljivo čekati nekoliko sati da bismo skratili projekt za nekoliko tjedana. Međutim, čak i nakon takvog optimiranja ne postroji apsolutna sigurnost da je GA pronašao optimalno rješenje jer ostvareni algoritam nije pokazao pouzdanost od 100%. U većini slučajeva algoritam je uspijevao pronaći optimalna rješenja koja su jednaka onima opisanima na [PSP06]. U nekim slučajevima GA čak i za najmanje projektne instance (30 aktivnosti) niti nakon višestrukih pokretanja nije za pojedine instance pronašao optimum jednak onome koji je dobiven istraživanjima. U tim ekstremnim slučajevima rješenja su ipak odstupala najviše 20% od optimalnih.

U ovom poglavlju je opisana primjena genetskog programiranja u optimiranju projekta s ograničenim sredstvima. Također je dana usporedba ostvarenog GA i GP, s posebnim naglaskom na uočene nedostatke koje je pokazao GA.

7.1. Funkcija cilja

Cilj ovog genetskog programa je pronaći onaj izraz koji može izračunati ispravan prioritet za sve aktivnosti projekta. Ispravan prioritet je onaj prioritet za koji će projekt imati minimalno trajanje.

Pseudokod željenog programa prikazan je na slici 7.1.

```
Za svaku aktivnost_i
    ako su završile sve aktivnosti koje joj prethode
        dodaj aktivnost aktivnost_i u skup aktivnih aktivnosti A

poredaj aktivnosti u skupu A po prioritetu aktivnosti P

dok ima aktivnost_i iz skupa P i dok ima slobodnih resursa
    ako su slobodni resursi koje aktivnost treba za izvođenje
        dodijeli aktivnost_i odgovarajuće resurse
```

Slika 7.1. Pseudokod algoritma za raspoređivanje aktivnosti.

Ciljano rješenje ovog genetskog programa neće biti cijeli računalni program već samo jedan manji dio, matematička funkcija koja izračunava vrijednost prioriteta P za svaku aktivnost na projektu.

7.2. Prikaz jedne jedinke

Jedinka ovog genetskog programa je izraz koji izračunava prioritet. Budući da se radi o jednostavnom problemu, odabrane su standardne aritmetičke operacije za rad s realnim brojevima (Tablica 7.1.).

Tablica 7.1. Funkcijski čvorovi koje se koriste za stvaranje izraza genetskim programom

Oznaka funkcijskog čvora	Opis
ADD	Zbrajanje
SUB	Oduzimanje
MUL	Množenje
DIV	Zaštićeno dijeljenje; u slučaju kada je nazivnik manji od 0,000001, rezultat dijeljenja je jednak 1.

Osim ovih operacija odabrani su i sljedeći završni znakovi (Tablica 7.2.)

Tablica 7.2. Podatkovni čvorovi koji se koriste za stvaranje izraza genetskim programom

Oznaka podatkovnog čvora	Opis
D	Trajanje aktivnosti.
RR	Broj različitih resursa koji su potrebni za izvođenje aktivnosti.
ARU	Prosječno zauzeće jednog resursa
SC	Broj aktivnosti koje slijede nakon trenutne aktivnosti (sve do kraja projekta)
PC	Broj aktivnosti koje prethodne trenutnoj aktivnosti (sve od početka projekta)

Kao što je vidljivo iz tablice 7.2. definirani podatkovni čvorovi ovisni su o broju aktivnosti i resursa koji su povezani s pojedinom aktivnošću. *D* (engl. *duration*) – označava broj vremenskih trenutaka u kojima se neka aktivnost odvija, ova vrijednost ne ovisi o broju i resursima koji se koriste. *RR* (engl. *requested resources*) – je broj različitih resursa koji su potrebni za izvođenje aktivnosti. Ako neka aktivnost *A* treba resurse *R1* i *R7* da bi se izvodila vrijedi *RR*=2. *ARU* (engl. *average resource usage*) je ukupan kapacitet potrebnih resursa podijeljen s brojem potrebnih resursa. Primjerice ako neka aktivnost *A* treba 7 jedinica sredstva *R2* i 2 jedinice sredstva *R3*, vrijedi *AVG*= (7+2)/2. Podatkovni čvor SC (engl. *successors count*) označava ukupan broj aktivnosti koje slijede nakon aktivnosti koja se trenutno analizira. PC (engl. *predecessor count*) je ukupan broj aktivnosti koje moraju završiti prije nego što trenutna aktivnost može početi.

Proces ocjene dobrote jedinke sličan je procesu koji je opisan za genetski algoritam. Nakon što GP kreira jedinku koja sadržava izraz, izračunava se prioritet za svaku aktivnost. To se radi tako

da se izračunaju atributi svake aktivnosti (tablica 7.2.). Izračunati atributi se zatim predaju kao argumenti u izraz trenutne jedinke.

Nakon što se izračunaju prioriteti za sve aktivnosti, pokrene se algoritam za raspoređivanje aktivnosti prikazan pseudokodom na slici 7.1. Rezultat izvršavanja toga algoritma je duljina trajanja projekta.

7.3. Ostvareni rezultati

U ovom su poglavlju opisani rezultati koji su dobiveni pokretanjem ostvarenog GP-a za projekte iz baze projekata. Rezultati su podijeljeni u dvije skupine, a dobiveni su optimiranjem samo jednog projekta i skupine projekata. Izraz dobiven samo za jedna projekt vrijedi samo za taj projekt. Izraz koji je dobiven za skupinu projekata vrijedi za sve projekte iz skupine.

Općenito su zanimljiviji oni rezultati koji su dobiveni za skupinu projekata. Rezultati su pokazali da izraz daje izvrsne rezultate kada se primijeni na bilo koji projekt iz te skupine, ali također da daje vrlo dobre rezultate kada se primijeni na bilo kakav nepoznati projekt koji nikad prije nije bio evaluiran tim izrazom.

7.3.1. Rezultati ostvareni za pojedinačne projekte

Za usporedbu rezultata koji dobiju sa genetskim algoritmom i genetskim programiranjem provedeno je testiranje identično onome opisanom u poglavlju 6.4. Optimirali su se isti projekti s istim postavkama genetskog programa (uvjeti zaustavljanja, veličina populacije, mutacije).

Tijekom ovog testiranja nisu postojala dva odvojena skupa podataka, već se tražio optimalan izraz za svaki od 40 projekata pojedinačno. Izrazi koji su pronađeni ovim načinom ne vrijede u općem slučaju već samo za pojedinačni projekt za koji je izraz izračunat.

Tablica 7.3. Rezultati koji su dobiveni optimiranjem genetskim programom za različite tipove projekata

Tip projekata	Prosječno kašnjenje projekata u odnosu na poznata optimalna rješenja	
	GP	GA
30 aktivnosti	0,89%	1,36%
60 aktivnosti	4,50%	3,73%
90 aktivnosti	0,89%	1,92%
120 aktivnosti	11,40%	8,20%

Kao što je vidljivo iz prethodne tablice (Tablica 7.3.), rezultati se kvalitetom ne razlikuju previše od onih dobivenih pomoću GA.

7.3.2. Rezultati ostvareni za skupinu projekata

Ostvareni GA koji raspoređuje aktivnosti za projekte s ograničenim sredstvima kao jednu od ključnih mana imao je predugo vrijeme izvođenja za složenije projekte. To je posebno bilo očito u slučajevima kada su se radi veće vjerojatnosti postizanja boljeg rješenja koristile populacije s većim brojem jedinki i većim brojem generacija. Optimiranje još složenijih projekata trajalo bi predugo da bi se poslovne odluke o načinu izvođenja tih projekata mogle donijeti na vrijeme. Upravo zbog toga postoji potreba da se pronađe onaj izraz koji vrijedi u svim slučajevima. Pomoću takvog izraza analiza projekata bi se mogla vršiti trenutno što bi ubrzalo i olakšalo donošenje poslovnih odluka.

Za pronalazak jedinstvenog izraza koji bi mogao općenito vrijediti za sve projekte koristila se baza opisana u [PSP06]. Baza sadržava 2040 instanci projekata koji su kreirani s različitim početnim parametrima. Projekti se dijele u 4 skupine s po 30, 60, 90 i 120 aktivnosti. Za pronalaženje izraza koji će vrijediti „univerzalno“ baza je podijeljena na dva dijela - skup za učenje od 240 projekata i skup za provjeru od 1800 projekata. Skup za učenje i skup za provjeru sastoje se od jednakih udjela projekata s 30, 60, 90 i 120 aktivnosti. GP provodi optimiranje samo nad skupom za učenje. Nakon što se dostigne neki od uvjeta zaustavljanja GP se zaustavlja i ispisuje izraz najbolje jedinke. Izraz se zatim primjenjuje nad skupom za provjeru. Bitni rezultati su oni koji su ostvareni nad skupom za provjeru jer su to projekti koje GP nikad prije nije optimirao. Dobri rezultati optimiranja skupa za provjeru pokazali bi da je pronađeni izraz valjan u općem slučaju. Takav izraz mogao bi se ugraditi u neki komercijalni alat za optimiranje projekata.

Izračun dobrote jedinke na skupu za učenje

Dobrota jedne jedinke je trajanje projekta izraženo brojem vremenskih trenutaka. Dodatno se takvo trajanje uspoređuje s poznatim optimalnim trajanjem kako bi se izračunala dobrota jedne jedinke na skupu za učenje. Algoritam za ocjenu dobrote jedne jedinke na skupu za učenje prikazan je pseudokodom na slici 7.2.

```
Ocjena dobrote D:
   Za svaki projekt P iz skupa za učenje
      Za svaku aktivnost A projekta P
         Izračunaj prioritet π aktivnosti A
         Pohrani prioritet

      Pokreni projekt P
      Rasporedi aktivnosti poštujući dodijeljen prioritet π
      Izračunaj trajanje projekta T

      Usporedi trajanje T s očekivanim trajanjem OT

      D += (T-OT)/OT
```

Slika 7.2. - Ocjena dobrote za projekte iz skupa za učenje

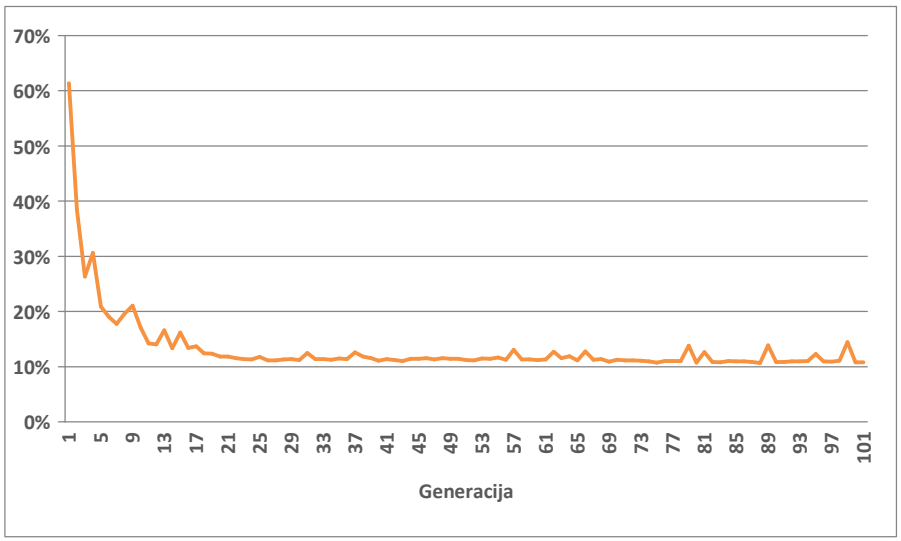
Očito je da nam za izračun ove vrijednosti mora biti poznata vrijednost OT, očekivano trajanje. Zato za pokretanje GP-a u skupu za testiranje moramo imati dovoljan broj projekata za koje smo nekom drugom nezavisnom metodom utvrdili optimalno trajanje.

Korištenjem prethodno prikazanog pseudokoda pokrenut je GP sa sljedećim parametrima (tablica 7.4.):

Tablica 7.4. Parametri genetskog programa

Parametar	Vrijednost
Veličina populacije	100
Uvjeti zaustavljanja:	
Maksimalni broj generacija	100
Maksimalni broj generacija bez poboljšanja najbolje jedinke	30
Vrsta GP	generacijska, M=50%
Način selekcije	3-turnirska
Vjerojatnost mutacije jedinke	0,5
Vjerojatnost mutacije jednog gena	0,01
Najveća dubina stabla za GP jedinke	5
Najmanja dubina stabla za GP jedinke	2

Optimiranje GP-a trajalo je neprekidno više od tjedan dana, na računalu Pentium III, 1,2 GHz, i s 256 MB RAM-a. PronaĐeni izraz je projekte iz skupa za učenje raspoređivao tako da je prosječno trajanje svih projekata bilo prosječno 10% dulje od optimalnog. Obzirom da se radi o većem broju projekata ovakvo odstupanje je i više nego dobro. Rezultati optimiranja prikazani su na slici 7.3.



Slika 7.3. Postotno kašnjenje jednog projekta tijekom generacija

Kao što je vidljivo iz grafa, kašnjenje se naglo smanjuje u prvih 30 generacija, a nakon toga nizom malih promjena teži prema kašnjenju od 10%.

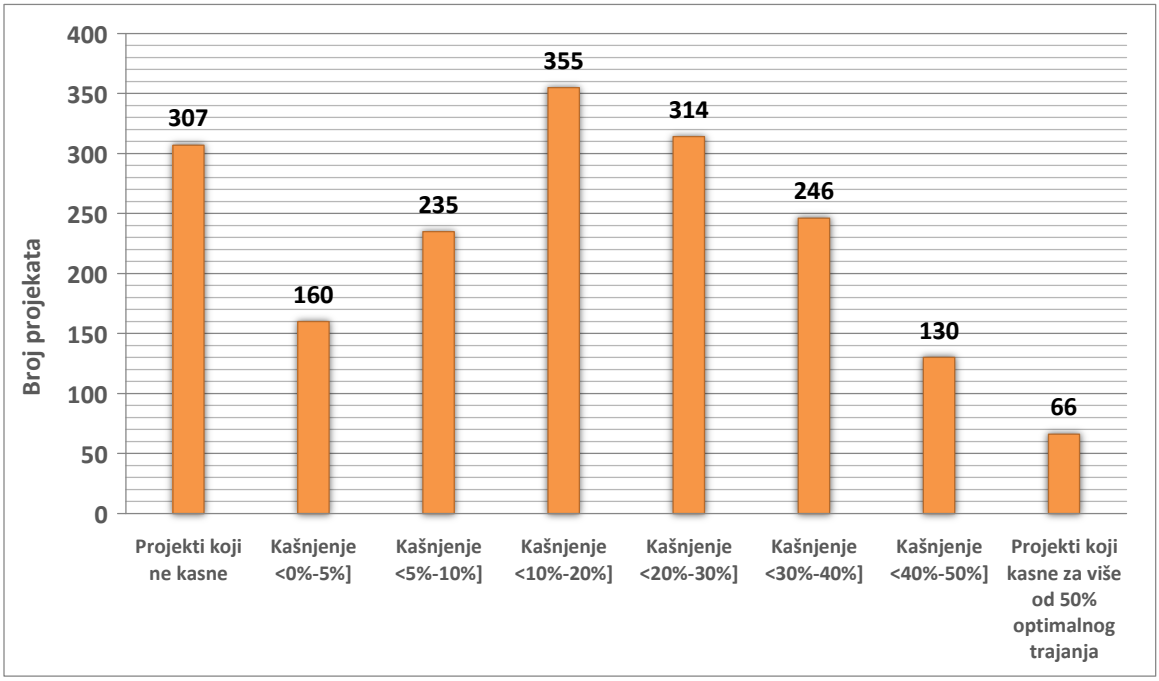
Primjena ostvarenih rezultata na skup za provjeru

Izraz za izračunavanje prioriteta jedne aktivnosti koji je ostvaren optimiranjem na skupu za učenje prikazan je na slici 7.5.

```
f = (((Divide(( D  *  D ),Divide( D ,(( SC  -  RR ) *  RR ))) * ( D  -  SC )) + ( D  *  RR
)) + ((Divide(( RRT  +  D ),( D  -  RRT )) + ((( SC  +  ARU ) - Divide((( RR  +  RRT ) +
(((Divide(( D  *  D ),((( PC  -  RRT ) + ( SC  * ( D  +  SC ))) + (( PC  -  PC ) * ( D  -
ARU )))) * ( D  -  RR )) + (( D  -  ARU ) + (( RRT  + (( D  +  RR ) * ( D  +  SC ))) + (( D
*  D ) * Divide( D ,( D  - ( RRT  *  ARU )))))) + (( PC  + (((Divide( ARU ,( PC  -  RRT ))
+ ( D  -  RRT )) -  RRT ) - Divide( PC ,( D  -  RRT )))) + ( RR  *  PC )))),( PC  -  RR )))
- Divide((( RR  +  RRT ) + (((Divide(( D  *  D ),(( RRT  + ( D  * ( RRT  +  D ))) + (( PC
-  PC ) * ( D  -  ARU )))) * ( D  -  RR )) + (( RRT  + (( D  +  SC ) * ( D  +  SC ))) + (
RR  * Divide( D ,( D  - Divide(( SC  -  RRT ),( D  *  D )))))) + ( RR  + ( RR  *  PC
)))),( PC  -  RR )))) + ( D  *  RR )))
```

Slika 7.5. Optimalni izraz za izračunavanje prioriteta aktivnosti

Kada se ovaj izraz primijeni na sve projekte iz skupa za provjeru dobiju se rezultati prikazani na slici 7.6.



Slika 7.6. Kašnjenje projekata po skupinama kašnjenja

Iz prethodne je slike vidljivo da vrlo veliki broj projekata uopće ne kasni (16,93% - 307 projekata), odnosno pronađena rješenja su jednaka optimalnim rješenjima. To su dakle projekti koji se izvršavaju točno onoliko dugo koliko je i njihovo optimalno trajanje koje je dokazano nekom

drugom nezavisnom metodom. Usporedimo to s rezultatima Standish Groupa koji su opisani u [Car05] i prema kojima se samo 16,3% projekata dovrši u roku. Prema tome, dobiveni broj projekata koji su dovršeni u roku i s predviĎenom budžetom je jednako dobar kao i u stvarnom svijetu. Ako bi se ostvareni genetski algoritam primijenio na većem skupu za učenje s duljim vremenom učenja vjerojatno bi ostvareni rezultati bili daleko bolji od stvarne situacije s IT projektima koji se izvode diljem svijeta.

Rezultati su za testnu skupinu još i bolji. Projekti iz testne skupine u prosjeku kasne oko 10%, što znači da su svi projekti iz testne skupine bolji od rezultata iz [Car05].

Ostali rezultati skupine za učenje takoĎer su vrlo zanimljivi. Čak 58,30% projekata će se završiti s produljenjem optimalnog trajanja od najviše 20%. Prosječno je kašnjenje svih jedinki 18,30%.

Ostvareni rezultati sigurno bi bili i bolji da je optimiranje obuhvatilo veći broj testnih projekata, veći broj jedinki po generaciji i ukupno veći broj generacija. Kad bi se pronašao izraz koji na testnom skupu ne uzrokuje kašnjenje niti jednog projekta, rezultati na testnom skupu bi sigurno bili i bolji. Pronalaženje izraza koji ne uzrokuje kašnjenje na testnom skupu sigurno može biti tema nekog od budućih radova.

8. Zaključak

U ovom su radu ostvareni evolucijski algoritmi namijenjeni optimiranju projektnih rasporeda za projekte s ograničenim sredstvima. Od evolucijskih algoritama korišteni su genetski algoritam i genetsko programiranje. Cilj rada bila je provjera mogu li se korištenjem evolucijskih algoritmima pronaći rješenja koja su jednaka dobra kao i ona dobivena brojnim drugim heurističkim metodama.

Evolucijski algoritmi ostvareni u ovom radu dali su izvrsne rezultate. Kada su se i genetski algoritam i genetski program primjenjivali na samo jednu izoliranu instancu projekta, pronalazili su optimalne rasporede koji su u prosjeku odudarali od optimalnih rješenja za tek nekoliko postotaka. Za razliku od ostalih heurističkih metoda ove evolucijske algoritme nije bilo potrebno posebno prilagoĎavati za svaku instancu problema. Algoritmi vrijede za sve projekte s ograničenim sredstvima neovisno o izgledu projektnog plana, složenosti, broju aktivnosti i resursa.

Posebno dobri rezultati ostvareni su kada je GP generirao izraz na skupu za učenje i kada je taj izraz primijenjen na skupu za testiranje. Rezultati su u postotku nešto više odudarali od rezultata za jedan projekt, ali su i dalje usporedivi sa poznatim rezultatima i istraživanjima stvarnih projekata. Za ispravno i detaljno testiranje koje bi dalo još bolje rezultate bili bi potrebni resursi koji nisu bili dostupni zbog opsega ovog rada.

Rezultati koji su ostvareni genetskim programiranjem nad skupinom projekata pokazali su da postoji izraz odnosno izrazi koji mogu izračunati prioritet aktivnosti na projektu s ograničenim sredstvima. Postojanje takvog izraza može olakšati izgradnju sustava za optimiranje projekata s ograničenim sredstvima. Ugradnja izračunatog izraza u neki takav sustav omogućila bi pronalazak optimalnih rasporeda aktivnosti projekata u realnom vremenu.

Istraživači koje su pronalazili rasporede koristeći druge tehnike optimiranja morali su posebno prilagoĎavati svoje algoritme za svaki problem ili inačicu problema. Izraz koji je pronaĎen genetskim programiranjem vrijedi u općem slučaju, tako da njegova primjena može biti trenutna za bilo koji problem koji zadovoljava osnovne pretpostavke toga modela.

Ovdje opisana rješenja u svakom slučaju ostavljaju prostor za daljnja istraživanja. Ovaj bi se rad mogao proširiti na nekoliko različitih načina. Za pronalazak boljeg i preciznijeg izraza za izračunavanje prioriteta bilo bi potrebno provesti detaljnija testiranja s većim skupovima za učenje, većim brojem generacija i većom populacijom jedinki.

Osim jednostavnih projekta, ostvarene bi algoritme trebalo testirati i na više-modalnim projektima sa ograničenim sredstvima. To su projekti čije je aktivnosti moguće izvršavati na više različitih načina. Svaki od tih načina ima različito trajanje aktivnosti i drugačije zauzeće resursa. Ovi tipovi projekata su posebno interesantni jer mogu razriješiti dvojbu kojim načinom je najbolje izvoditi projekt.

9. Literatura

[Gol04] Golub M. *Genetski algoritmi – Prvi dio*, 2004. Fakultet elektrotehnike i računarstva. Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave. Dostupno na Internet adresi: http://www.zemris.fer.hr/~golub/ga/ga.html

[Gol02] Golub M. *Genetski algoritmi – Drugi dio*, 2002. Fakultet elektrotehnike i računarstva. Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave. Dostupno na Internet adresi: http://www.zemris.fer.hr/~golub/ga/ga.html

[Kal96] Kalpić D., Momar V. *Operacijska istraživanja*. 1996. Biblioteka informacijsko društvo.

[Jak05] Jakobović D. *Raspoređivanje zasnovano na prilagodljivim pravilima – doktorska disertacija*, 2005. Fakultet elektrotehnike i računarstva. Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave.

[Koz98] Koza J.R. *Genetic Programming - On the Programming of Computers by Means of Natural Selection*, 1998. The MIT Press.

[Koz03] Koza J.R. *Lecture notes: Comparison of Search and learning methods.* Stanford University, 2003. Dostupno na internet adresi: http://www.genetic-programming.com/coursemainpage.html

[PMB04] *A guide to the: Project Management Body of Knowledge*, Third Edition, PMBOK Guide, 2004. An American National Standard. ANSI.

[MSF99] *Microsoft Solution Framework – Overview white paper*, 1999. Microsoft Corporation. Dostupno na Internet adresi: http://www.microsoft.com/msf

[Men05] Mendes, Gonçalves, Resende. *A Random Key Based Genetic Algorithm for the Resource Constrained Project Scheduling Problem*, 2005. AT&T Labs Research Technical Report TD-6DUK2C

[Kam03] Kamaraien O., Ek V., Nieminen K., Ruuth S. *Large scale generalized resource constrained scheduling problems: A genetic algorithm approach*, 2003. IC-Parc, Imperial College, London.

[Sri03] Sriprasert E., Dawood N. *Genetic algorithms for multi-constraint scheduling: An application for the construction industry*, 2003. Centre for Construction Innovation Research, Universty of Teeside.

[PSP06] Kolisch R. i drugi autori: *PSPLIB – Library for project scheduling problems.* Dostupno na internet adresi: http://129.187.106.231/psplib/

[Har98] Hartmann S. *A competitive genetic algorithm for resource-constrained project scheduling*, 1998. Naval Research Logistics.

[Kim05] Kim J. *A framework for integration model of resource-constrained scheduling using genetic* algorithms, 2005. Proceedings of the 2005 Winter Simulation Conference.

[OB06] Open Bealge, a versatile EC Framework. Dostupno na Internet adresi: http://beagle.gel.ulaval.ca/

[Sev00] Sevaux M., Dauzere-Peres S. *Building a Genetic Algorithm for a Single Machine Scheduling Problem*, 2000.

[Bar96] Bartschi Wall M. A Genetic Algorithm for Resource-Constrained Scheduling, 1996. Massachusetts Institute of Technology.