UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Dejan Štumberger

UPORABA GENETSKEGA ALGORITMA PRI UPRAVLJANJU PROGRAMSKIH PROJEKTOV

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Arjana Žitnik

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01759/2011
Datum: 04.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DEJAN ŠTUMBERGER**

Naslov: **UPORABA GENETSKEGA ALGORITMA PRI UPRAVLJANJU PROGRAMSKIH PROJEKTOV**

SOFTWARE PROJECT MANAGEMENT WITH GENETIC ALGORITHMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V diplomskem delu obravnavajte problem upravljanja programskih projektov in genetski algoritem za njegovo reševanje. Algoritem tudi implementirajte in preizkusite na različnih testnih primerih.

Mentor: Žitnik

doc. dr. Arjana Žitnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani: Dejan Štumberger,
z vpisno številko: 63020330,
sem avtor diplomskega dela z naslovom:
Uporaba genetskega algoritma pri upravljanju programskih projektov.

S svojim podpisom zagotavljam, da:
- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Arjane Žitnik,

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela Fri«.

V Ljubljani, dne Podpis:

Zahvala

Zahvaljujem se mentorici doc. dr. Arjani Žitnik za mentorstvo in pomoč pri izdelavi diplomske naloge. Zahvaljujem se tudi staršema z moralno in finančno podporo.

Kazalo

Povzetek 1
Abstract 2
1 Uvod 3
2 Genetski algoritmi 5
2.1 Uvod v genetske algoritme 5
2.2 Osnovni gradniki genetskega algoritma 7
2.2.1 Geni in kromosomi 7
2.2.2 Ocenjevalna funkcija 8
2.2.3 Populacija kromosomov 9
2.2.4 Koraki genetskega algoritma 10
2.3 Selekcija 12
2.4 Križanje 14
2.5 Mutacija 16
2.6 Posodabljanje populacije kromosomov 17
2.7 Zaključek genetskega algoritma 18
3 Predstavitev problema upravljanja programskih projektov 21
3.1 Opis projekta in virov 22
3.2 Množica dopustnih rešitev 23
3.3 Kriterijska funkcija 23
3.4 Primer rešitve programskega projekta 25
3.5 Problem prekrivanja aktivnosti 27
4 Realizacija genetskega algoritma za upravljanje programskih projektov 29
4.1 Opis algoritma 30
4.2 Sestavljanje kromosomov 31
4.3 Ustvarjanje začetne populacije kromosomov 33
4.4 Selekcijske metode 34
4.5 Križanje in mutacija 36
4.6 Posodabljanje populacije in zaključek programa 37
4.7 Ocena časovne zahtevnosti algoritma 38
5 Meritve 39
5.1 Podrobnejši prikaz meritev za prvi testni primer 40
5.1.1 Grafi meritev testnega primera 42
5.1.2 Podrobni izpis najboljše rešitve testnega primera 46
5.2 Tabela meritev testne množice in ugotovitve 47
6 Sklepne ugotovitve 49
Dodatek: kazalo slik 51
Literatura 53

Kazalo simbolov

Poglavje 2

$P(i)$ - populacija v generaciji i

i - generacija (iteracija) genetskega algoritma

$O(i)$ - razširjena populacija staršev in potomcev v generaciji i

l - dolžina gena

d - dolžina kromosoma

c_g - število genov v kromosomu

f_k - ocenjevalna funkcija

p_m - verjetnost mutacije

O - vsota vseh fitnes ocen

ps_i - verjetnost selekcije za kromosom i

N - število kromosomov v populaciji

p_{el} - odstotek elitne populacije

p_{cs} - verjetnost križanja

k - število pozicij križanja

c_{poz} - pozicija križanja

p_u - verjetnost, da prvi starš prenese niz genov na prvega otroka

s – velikost populacije

s_o - velikost začetne populacije

s_i - velikost populacije generacijo i

n – število izbranih kromosomov iz populacije

m – število kromosomov dodanih v populacijo

pc_i - povprečna fitnes ocena generacije *i*

Δv_i - rast povprečne fitnes ocene v generaciji *i*

pr - sprejemljiv odstotek odstopanja rasti povprečne fitnes ocene

Poglavje 3

Pv – množica aktivnosti projekta

maxef - maksimalna predanost projektu

$Zp_i^{\max ef}$ - maksimalna predanost zaposlenega *i* projektu

Zp_i^{placa} - mesečno plačilo zaposlenega *i*

Zaposleni – množica zaposlenih projekta

Sz - velikosti množice zaposlenih projekta

Znanje – množica znanja projekta

Szn - velikosti množice znanja projekta

Sp - velikost projekta

Ak_j - aktivnost *j*

t_j - čas izvajanja aktivnosti *j*

Ak_j^t - trajanje aktivnosti *j*

Ak_j^{pred} - predniki aktivnosti *j*

Ak_j^{zz} - zahtevana znanja za izvajanje aktivnosti *j*

$G(V, A)$ - usmerjeni aciklični graf projekta

$M = (m_{i\,j})$ - matrika kromosoma projekta

$m_{i\,j}$ - element matrike *M*

j - število vrstic kromosoma

i - število stolpcev kromosoma

$|Ak|$ - velikost množice *Ak*

$|Zp|$ - velikost množice *Zp*

u_{kp} - utež kritične poti

t_{kp} - čas izvajanja kritične poti

u_{Str} - utež stroškov

S_{str} - stroški projekta

u_{pr} - utež prekrivanja aktivnosti

t_{pr} - čas izvajanja vseh prekrivajočih aktivnosti

KPA - množica aktivnosti ki nastopajo v kritični poti projekta

Oc_M - fitnes ocena kromosoma *M*

tc_n - normalni čas trajanja aktivnosti

c_n - normalna cena aktivnosti

tc_i - izjemni čas trajanja aktivnosti

c_i - izjemna cena aktivnosti

Poglavje 5

Oc_{im} - fitnes ocena najboljše rešitve za testni primer Pr_i in meritev *m*

Pr - množica testnih primerov

Pr_i - testni primer *i*

Seznam kratic

EA – evolucijski algoritmi

GA – genetski algoritem

PUPP – problem upravljanja programskega projekta

GAUPP – genetski algoritem za upravljanje programskih projektov

UV – uporabniški vmesnik

MKP - metoda kritične poti

Povzetek

V diplomskem delu obravnavamo genetski algoritem za reševanje *problema upravljanja programskih projektov*. Programski projekt sestavljajo aktivnosti, ki jih je treba izvesti v čim krajšem času in/ali s čim manjšimi stroški in z minimalnim prekrivanjem aktivnosti. Pri tem so na voljo viri: zaposleni z znanji.

Naprej podrobno predstavimo delovanje genetskih algoritmov. Nato formalno opišemo problem upravljanja programskih projektov. Sledi opis realizacije genetskega algoritma za problem upravljanja programskih projektov. Delo zaključimo s testiranjem genetskega algoritma za upravljanje programskih projektov. S pomočjo testiranja poskušamo podati napotke za izbiro parametrov genetskega algoritma, da bo le-ta organiziral aktivnosti programskega projekta in vire tako, da se bo, glede na zastavljene kriterije, projekt izvedel optimalno.

Ključne besede:
- dopustna rešitev
- genetski algoritem
- optimizacijska naloga
- metoda kritične poti
- upravljanje programskih projektov

Abstract

In this graduation thesis, a genetic algorithm for solving the *software project management problem* is discussed. A software project consists of activities which need to be implemented as quickly as possible and/or with minimal cost and with minimal overlap of activities, with use of available resources. These are employees with skills.

In the first part of the thesis, genetic algorithms are presented. Then the software project management problem is formally described. Description of the realization of a genetic algorithm for the software project management problem follows. The thesis is completed by testing the algorithm in order to provide guidance for selecting parameters of the genetic algorithm, such that it will organize the activities and the resources of the software project optimally with respect to given criteria.

Keywords:
- feasible solution
- genetic algorithm
- optimization problem
- critical path method
- software project management

1 Uvod

Genetski algoritmi so se kot posebna inačica evolucijskih algoritmov začeli razvijati v 70-tih letih prejšnjega stoletja in sodijo v širše področje umetne inteligence. Genetski algoritmi (GA) so hevristični algoritmi, ki se pri delovanju zgledujejo po biološki evoluciji. GA pri iskanju optimalne rešitve v danem iskalnem prostoru – množici dopustnih rešitev, uporablja genetske operacije, kot so dedovanje, selekcija, mutacija, križanje, ter tako posnema mehanizme naravne evolucije. Vsak GA operira nad populacijo dopustnih rešitev problema ali drugače imenovano populacijo *kromosomov*. GA se izvaja nad populacijo več generacij, pri tem se vsako generacijo populacija posodobi. Posodabljanje populacije določa metoda posodabljanja in kriterijska funkcija GA, ki ločuje slabše kromosome od boljših. S primerno izbrano metodo posodabljanja dosežemo, da se populacija vsako generacijo izboljšuje, kar nas pripelje do uporabnih rešitev problemov. GA uporabljamo za iskanje uporabnih rešitev problemov, pri katerih je čas reševanja z deterministični algoritmi prevelik, na primer problem potujočega trgovca. Slabost GA je, da pri iskanju optimalnih rešitev problema pogosteje vrača lokalne kot globalne optimume.

Namen diplomskega dela je predstavitev delovanja genetskih algoritmov in opis realizacije GA, namenjenega reševanju problema upravljanja programskih projektov. *Problem upravljanja programskih projektov* (PUPP) je podan s projektom, ki ga sestavljajo medsebojno povezane aktivnosti z danim časom trajanja in zahtevanimi znanji in viri: zaposleni z različnimi znanji. Aktivnosti projekta je potrebno organizirati tako, da bo projekt izveden v čim krajšem času in/ali s čim manjšimi stroški in z minimalnim prekrivanjem aktivnosti. Genetski algoritem reševanje tega problema bomo imenovali *genetski algoritem za upravljanje programskih projektov* (GAUPP). Namen GAUPP je projektni vodji služiti kot orodje pri načrtovanju in organizaciji aktivnostih in zaposlenih programskega projekta.

Problem upravljanja programskih projektov je inačica *problema organizacije z omejenimi viri* (*resource-constrained project scheduling problem*), podrobneje opisan v [4]. Problem organizacije z omejenimi viri je sestavljen iz aktivnosti, ki jih moramo opraviti, omejeno količino virov, s katerimi bomo opravljali aktivnosti, in kriteriji, ki jih moramo izpolniti pri opravljanju aktivnosti. Kriteriji so lahko časovna omejitev za dokončanje aktivnosti, postopkovna omejitev, kjer je vrsti red aktivnosti pomemben, in omejitev sredstev, kjer določimo količino denarja in zaposlenih na voljo za izvedbo aktivnosti. Za problem organizacije z omejenimi viri je dokazano, da je NP-težek, glej [3]. Ker je reševanje slednjega problema s klasičnimi deterministični tehnikami časovno prezahtevno, uporabimo hevristične tehnike. Izkaže se, da se pri reševanju problema organizacije z omejenimi viri genetski algoritmi zelo dobro obnesejo. Različni GA namenjeni reševanju tega problema so predlagani v [1], [8] in [10]. Kot temeljni vir pri definiciji PUPP in realizaciji GAUPP uporabimo [1].

Na kratko opišimo zgradbo diplomskega dela. V 2. poglavju najprej podrobneje predstavimo genetski algoritem. Opišemo njegove osnovne gradnike: gene in kromosome, kriterijsko – ocenjevalno funkcijo, populacijo kromosomov, ter s psevdokodo in diagramom poteka prikažemo njegov potek izvajanja. Podrobneje opišemo logiko delovanja vseh genetskih operacij poimenovanih koraki GA: selekcije, križanja, mutacije. Pri vsaki operacij predstavimo več različic, ter za vsako navedemo prednosti in slabosti. Opišemo tudi tri načine rasti populacije, štiri metode posodabljanja populacije in predstavimo različne načine zaključka izvajanja GA.

V 3. poglavju podrobneje predstavimo problem upravljanja programskih projektov. PUPP formalno definiramo in podrobno opišemo njegove vhodne podatke: projekt in vire. Določimo množico dopustnih rešitev. Dopustno rešitev poimenujemo *kromosom*. Nato v obliki enačb zapišemo kriterijsko funkcijo PUPP. Podamo tudi primer izračuna fitnes ocene poljubnega kromosoma na demonstracijskem primeru programskega projekta in se dotaknemo problema prekrivanja aktivnosti, pri katerem ponudimo več možnih rešitev.

V 4. poglavju nato opišemo realizacijo GUAPP. Delovanje GAUPP opišemo s psevdokodo. Predstavimo pravila sestavljanja kromosomov in opišemo matrični zapis kromosomoma. Pri matričnem zapisu vrednost posameznega gena kromosoma prikažemo kot kombinacijo treh bitov, ki jo poimenujemo binarni zapis dolžine tri. Pravila sestavljanja kromosomov implementiramo pri korakih ustvarjanja začetne populacije, križanja in mutacije. S psevdokodo prikažemo delovanje vseh treh podprtih selekcijskih metod in na praktičnem primeru demonstriramo delovanje križanja in mutacije. Na koncu poglavja razdelamo vse tri načine rasti populacije, ter definiramo pogoje za zaključek GAUPP. Na podlagi opisane realizacije je GAUPP tudi implementiran.

V poglavju meritev ustvarimo naključno množico desetih testnih primerov in za GAUPP poiščemo kombinacijo izbire selekcijske metode in načina rasti populacije, ki bo našla najboljše rešitve. Za lažjo predstavitev za prvi testni primer prikažemo grafe vseh devet meritev - kombinacij in podrobneje izpišemo najboljšo rešitev. Pri iskanju optimalne kombinacije izbire selekcijske metode in načina rasti populacije za vsak testni primer iz testne množice opravimo devet meritev in po opravljenih meritvah prikažemo tabelo meritev testne množice, ter opišemo ugotovitve meritev. Izkaže se, da kombinacija selekcije elitizma in počasne rasti populacije daje najboljše rezultate.

2 Genetski algoritmi

Genetski algoritmi so vrsta evolucijskih algoritmov (EA), ki s pomočjo posnemanja naravnih evolucijskih tehnik iščejo optimalne rešitve optimizacijskih problemov. V tem poglavju bomo predstavili temeljne zakonitosti in naravo delovanja genetskega algoritma. Celotno drugo poglavje o teoriji genetskih algoritmov temelji na virih [2], [5], [6], [11], [12], [13] in [14].

2.1 Uvod v genetske algoritme

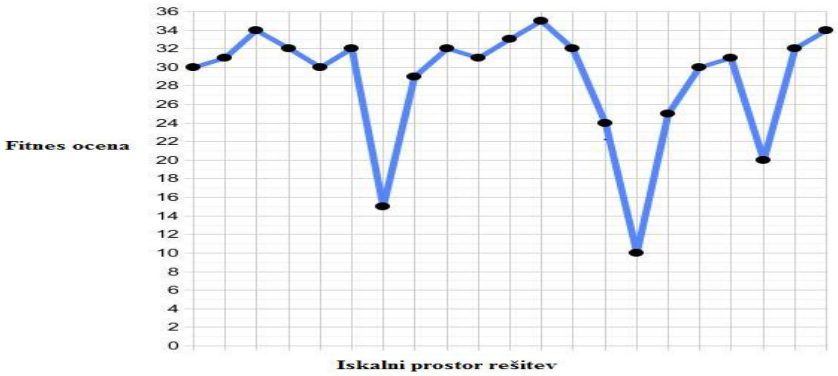
Evolucijski algoritmi

Evolucijski algoritmi (EA) so kot podmnožica evolucijskega računanja del področja umetne inteligence. Klasificiramo jih kot hevristične optimizacijske algoritme, ki temeljijo na populacijah. Hevristične optimizacijske algoritme uporabljamo pri reševanju problemov, za katere ne znamo poiskati hitrih eksaktnih algoritmov. Običajno proizvedejo dobro rešitve, ki pa niso nujno optimalne. Posamezni kandidati rešitve za optimizacijski problem so predstavljeni kot člani populacije. EA se zgledujejo po biološki evoluciji živih bitij. Uporabljajo mehanizme dedovanja, selekcije, mutacije in križanje, da posnemajo delovanje naravne evolucije. Kriterijska ali ocenjevalna funkcija v populaciji loči slabše člane od dobrih. Ker se populacija skozi generacije (iteracije algoritma) spreminja, je naloga ocenjevalne funkcije, da jo regulira v populacijo, kjer uspevajo najboljši posamezniki. Zaradi te lastnosti EA najdejo dobre približne rešitve za mnogo vrst problemov. Uporabljajo se na različnih področjih kot so biologija, računalništvo, ekonomija, operacije raziskave, robotika, fizika, kemija, itd.

Poznamo več vrst evolucijskih algoritmov: genetski algoritmi, genetsko programiranje, evolucijsko programiranje in evolucijske strategije. Te tehnike imajo podobne osnovne gradnike in logiko izvajanja. Glavna razlika med posameznimi vrstami EA je v predstavitvi rešitve in v naravi problemov, ki jih rešujejo. *Genetski algoritmi* (GA) rešitev predstavijo v obliki niza številk ali znakov, najpogosteje se uporablja binarna predstavitev. GA pogosto uporabljamo za reševanje optimizacijskih problemov. Pri *genetskem programiranju* so rešitve podane v obliki računalniških programov. Njihova ustreznost je določena z njihovo sposobnost reševanja v naprej zadanega problema. *Evolucijsko programiranje* je podobno genetskemu programiranju, s tem da je struktura računalniškega programa fiksna in se numerični parametri programa lahko spreminjajo med izvajanjem. Glavni in najpogostejši operator evolucijskega programiranja je mutacija. *Evolucijske strategije* za predstavitev rešitve uporabljajo vektorje realnih števil in med izvajanjem algoritma samostojno spreminjajo pogostost mutacije za iskanje najboljših rešitev.

Genetski algoritem

Genetski algoritmi so vrsta evolucijskih algoritmov, ki se primarno uporabljajo kot optimizacijski algoritmi za iskanje lokalnih in globalnih maksimumov ali minimumov v podanem iskalnem prostoru možnih rešitev optimizacijske naloge (slika 2.1). V celotnem drugem poglavju bom zaradi lažje obravnave snovi predpostavil, da v iskalnem prostoru iščemo globalni minimum podane optimizacijske naloge.



Slika 2. 1: Graf lokalnih in globalnih minimumov v iskalnem prostoru

Optimizacijska naloga je trojica (D, f, opt), kjer je D množica *dopustnih rešitev*, $f: D \to \Re$ *kriterijska funkcija* in $opt \in \{\min/\max\}$. Iščemo *optimalno rešitev* problema $x \in D$, kjer je $f(x)$ najmanjša/največja vrednost na množici D. Z genetskim algoritmom iščemo čim boljšo rešitev optimizacijske naloge, ki pa ni nujno optimalna. Pri tem elemente množice dopustne rešitve imenujemo *kromosomi*. Genetski algoritem poteka v več iteracijah. Na začetku izberemo podmnožico rešitev, ki jo imenujemo *začetna populacija* in jo označimo s $P(0)$. V vsaki iteraciji nad populacijo *P(i)* izvedemo genetske operacije: selekcijo, križanje in mutacijo. Selekcija izbira kromosome populacije, ki jih bomo uporabili za križanje, glede na njihovo oceno. Križanje sestavi iz dveh kromosomov nov kromosom. Mutacija naključno spremeni novo ustvarjen kromosom. Novo ustvarjeno populacijo kromosomov - potomcev $O(i)$ nato ocenimo. Na koncu vsake iteracije iz elementov *P(i)* in *O(i)* sestavimo novo populacijo $P(i+1)$. Po določenem številu generacij algoritem skonvergira v globalno (optimalno) ali lokalno (sub-optimalno) rešitev problema. Ko se algoritem zaključi, poiščemo najboljše ocenjen kromosom iz populacije. Ta predstavlja rešitev problema, ki jo vrne GA.

Psevdokoda **GA** za reševanje optimizacijske naloge (D, f, opt)

Vhod: vhodni podatki optimizacijske naloge

---

i = 0
ustvarimo začetno naključno populacijo $P(i)$ sestavljeno iz naključnih kromosomov
ocenimo začetno populacijo $P(i)$
ponavljaj dokler (ni izpolnjen pogoj za zaključek GA)
selekcija $P(i)$
križanje in **mutacija** $P(i)$, dobimo $O(i)$
sestavi populacijo $P(i+1)$ iz elementov $P(i)$ in $O(i)$
i = i + 1
ponavljaj

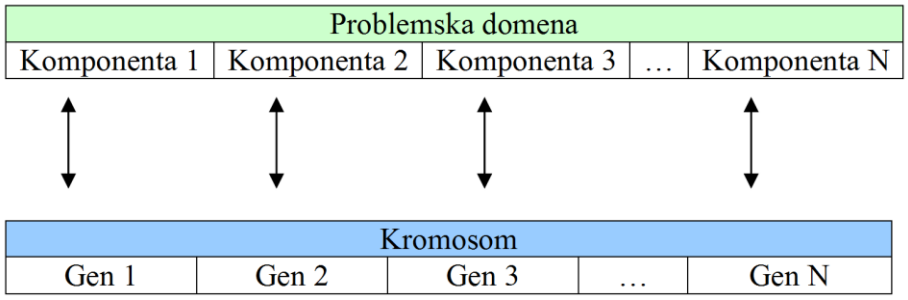
Izhod: optimalna rešitev $x \in D$

2.2 Osnovni gradniki genetskega algoritma

V tem razdelku prestavimo osnovne gradnike genetskega algoritma in podrobneje razdelamo vsakega od njih v nadaljnjih razdelkih. V celotnem razdelku predpostavimo da rešujemo optimizacijsko nalogo (D, f, opt). Dopustne rešitve, to je elemente množice D, bomo imenovali *osebke*.

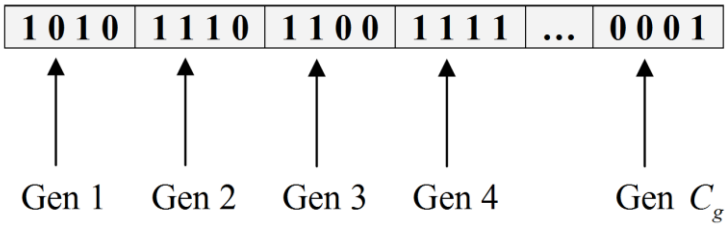
2.2.1 Geni in kromosomi

Vsak osebek prestavimo s *kromosomom*. Vsak posamezni kromosom je sestavljen iz več *genov*. Za zapis kromosoma dani optimizacijski problem prevedemo iz matematičnega zapisa v genetski zapis (slika 2.2). To storimo tako, da vsako komponento rešitve problema prevedemo v posamezni gen. Polje vseh prevedenih komponent rešitve - genov tvori posamezni kromosom. Vsak kromosom v populaciji predstavlja eno izmed možnih rešitev danega problema. Vsak gen v kromosomu prikazuje posamezno komponento rešitve.



Slika 2. 2: Preslikava problemske domene iz matematičnega zapisa v genetski zapis

Osnovna informacijska celica GA je gen, ki je hkrati tudi njegov najmanjši gradnik. Zapišemo ga lahko v različnih oblikah, ki so podrobneje predstavljene na naslednji strani. Za najpogostejši zapis genov se uporablja binarni zapis, kjer je posamezni gen predstavljen kot polje bitov dolžine l. Vsi geni, ki nastopajo v GA morajo biti enake dolžine. Kromosom predstavimo kot polje genov dolžine d (slika 2.3). Dolžina kromosoma je $d = c_g \cdot l$, pri čemer je c_g število genov v kromosomu. Vsi kromosomi, ki nastopajo v danem GA, imajo torej enako dolžino d. Poudariti je tudi potrebno, da posamezni gen v kromosomu ni v medsebojni odvisnosti od drugih genov znotraj istega kromosoma. Zato lahko gene poljubno prenašamo med kromosomi brez kakršnihkoli omejitev.



Slika 2. 3: Kromosom

Oblike zapisa kromosoma

Kromosome zapišemo v različne oblike glede na potrebe reševanja optimizacijskega problema. Za zapis kromosoma lahko med drugim uporabimo: binarni, desetiški, znakovni ali zapis v obliki permutacijskega koda. Navedeni zapisi so predstavljeni v slikah 2.4, 2.5, 2.6 in 2.7.

Kromosom A	1	1	0	0	0	1	0	1
Kromosom B	0	0	1	1	0	1	0	0

Slika 2. 4: Primer binarnega zapisa kromosoma

Kromosom A	2	5	6	7	2	9	4	1
Kromosom B	7	2	5	1	8	4	5	3

Slika 2. 5: Primer desetiškega zapisa kromosoma

Kromosom A	A	L	C	E	G	Z	B	H
Kromosom B	H	J	K	E	T	U	L	B

Slika 2. 6: Primer znakovnega zapisa kromosoma

Kromosom A	1	3	5	2	6	4	7	8
Kromosom B	8	6	5	7	2	3	1	4

Slika 2. 7: Primer kromosoma zapisanega v obliki permutacijskega koga

2.2.2 Ocenjevalna funkcija

Kriterijsko funkcijo optimizacijske naloge GA poimenujemo tudi *ocenjevalna funkcija* f_k. Naloga ocenjevalne funkcije je implementacija naravne selekcije v GA. Med izvajanjem algoritma potrebujemo mehanizem, ki loči slabše rešitve od dobrih. Ocenjevalna funkcija je lahko objektivna in subjektivna. Objektivna funkcija temelji na matematičnem modelu ali računalniški simulaciji, medtem ko pri subjektivni uporabnik algoritma loči dobre in slabe rešitve. Ocenjevalna funkcija vsako generacijo vsak novo ustvarjen kromosomom oceni - izračuna mu *fitnes oceno* ali t.i. *oceno ustreznosti*. Fitnes ocena kromosoma je realna ali celoštevilska vrednost ki predstavlja moč posameznega kromosoma znotraj populacije. Če iščemo globalni minimum, potem manjša kot je ocena kromosoma, močnejši je ta v primerjavi z ostalimi kromosomi v populaciji. Močnejši, ko je kromosom, večja je verjetnost, da bo izbran v selekciji. Poveča se tudi verjetnost, da kromosom ostane v populaciji več generacij.

2.2.3 Populacija kromosomov

Pri vsaki iteraciji GA delujemo na končni množici možnih rešitev – kromosomov. To množico poimenujemo *populacija kromosomov*, na kratko *populacija* (slika 2.8). Kromosomi ki sestavljajo populacijo so medsebojno neodvisni. Dodajanje in odstranjevanje kromosomov so edine dovoljene operacije, ki lahko spreminjajo populacijo. Te operacije nastopajo v GA vsako generacijo ob izvršitvi koraka posodabljanje populacije. Velikost populacije med izvajanjem GA je lahko konstantna ali rastoča. Nikoli pa ne sme biti padajoča, ker bi raznolikost populacije drastično upadala z vsako generacijo, kar bi pomenilo vedno slabše rešitve. Če bi populacija izumrla, bi se GA predčasno zaključil brez rešitve.

Populacija	Kromosom 1	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
	Kromosom 2	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**
	Kromosom 3	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
	Kromosom 4	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
	……	..	..	..	..	..	..	..	..
	Kromosom N	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**

Slika 2. 8: Populacija kromosomov

Začetno populacijo v večini primerov sestavljajo naključno ustvarjeni kromosomi. Velikost začetne populacije ter njena prihodnja rast so pogojeni z zahtevnostjo problema, ki ga rešujemo. V idealnem primeru bi bila velikost populacije skoraj neomejena. Vendar se pri reševanju realnih problemov srečamo z časovnimi okviri za izvedbo algoritma. Velikost populacije je največje časovno breme algoritma, zato moramo poiskati najbolj optimalno razmerje med velikostjo populacije in pričakovanim časom izvajanja algoritma. Slednje še posebej velja za rastoče populacije. Rast populacije vsako z generacijo močno poveča nabor potencialnih rešitev, a se tudi znatno poveča časovna zahtevnost algoritma.

Pomembno je omeniti tudi pojem genetske raznolikosti populacije. Poznamo dva tipa genetske raznolikosti, močno in šibko. Za populacijo močne genetske raznolikosti velja, da jo sestavljajo kromosomi, katerih fitnes ocene so dovolj raznolike, da GA ne zaide hitro v lokalne minimume, kar omogoča iskanje boljših optimalnih rešitev. Za šibko genetsko raznolikost (zelo majhna nihanja med fitnes ocenami kromosomov) populacije je značilen zgodnji pristanek GA v lokalnem minimumu, in zato rešitve v večini primerov niso zadovoljive. Torej velikost populacije vpliva na raznolikost nabora potencialnih rešitev. Zato so rastoče populacije poleg mutacije eden izmed najpomembnejših načinov preprečevanja šibke genetske raznolikosti.

2.2.4 Koraki genetskega algoritma

GA je iz sestavljen iz več iteracij, pri katerih se zaporedno izvede nekaj korakov (slika 2.9). Prvi in drugi korak GA, ustvarjanje in ocenitev začetne populacije, se izvedeta samo ob pričetku algoritma. Nato se zaporedno izvajajo koraki od selekcije do posodabljanja populacije, dokler ni izpolnjen noben od pogojev za zaključek algoritma. Na koncu v populaciji kromosomov poiščemo kromosom z najboljšo oceno, slednji predstavlja najboljšo najdeno rešitev danega problema. Koraki izvajanja GA so naslednji:

- **Ustvarjanje začetne populacije**:
  Naprej ustvarimo začetno populacijo kromosomov. Vrednosti genov ki tvorijo kromosome so določene naključno.

- **Ocenitev začetne populacije**
  Vsak kromosom v populaciji ocenimo in mu določimo fitnes oceno.

- **Selekcija**
  Selekcija je prvi korak vsako za generacijo GA, pri katerem s primernim načinom (naključna ruleta, elitizem, itd.) izberemo pare kromosomov – starše, ki jih uporabimo v koraku križanje.

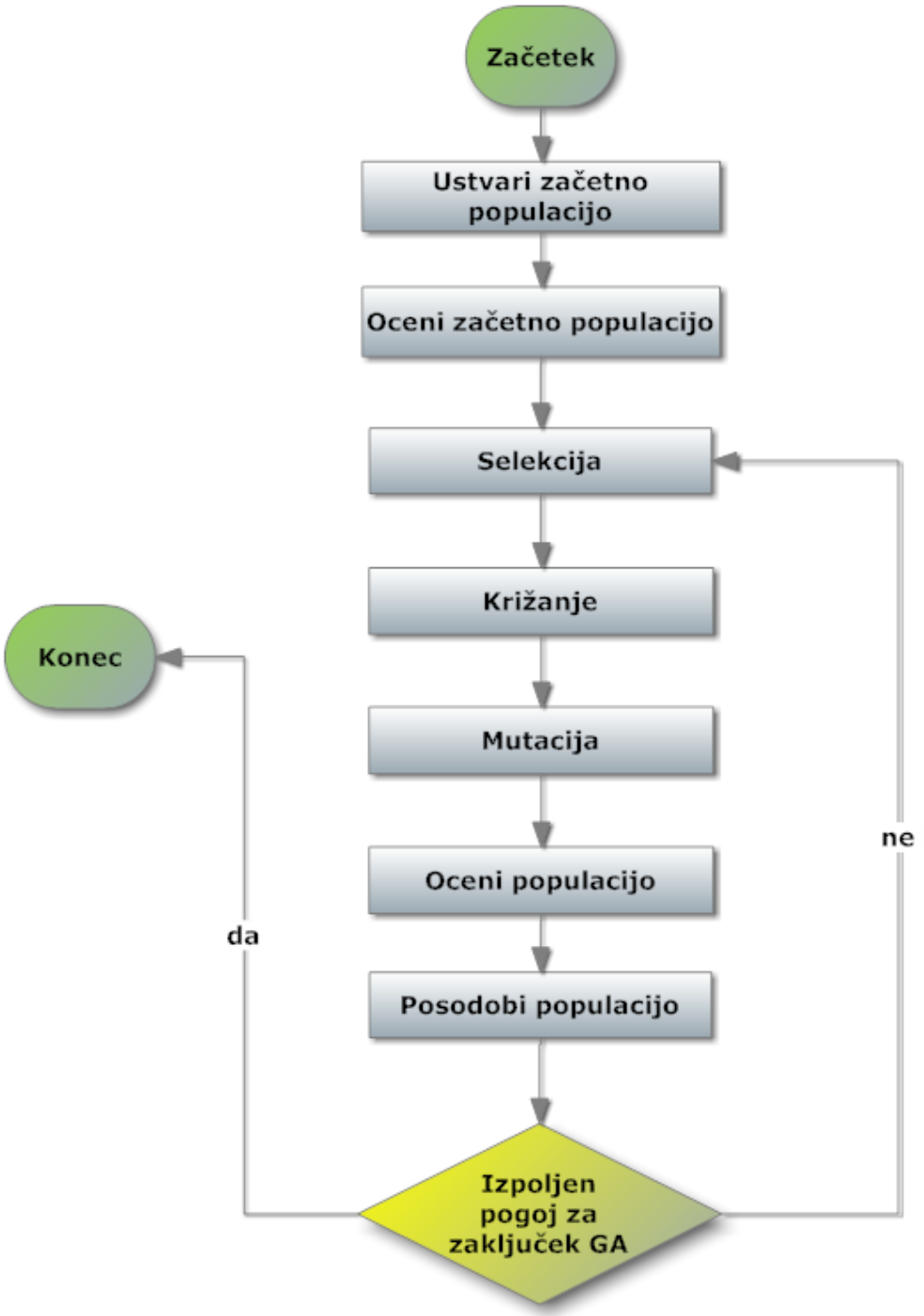
- **Križanje**
  Ko s selekcijo izberemo pare staršev, sledi križanje genov staršev. V tem koraku z različnimi metodami prenašanja genov med staršema sestavljamo nove kromosome – otroke. Korak križanje bi lahko poimenovali tudi podkorak selekcije. Selekcija namreč ob stvaritvi vsakega novega para staršev pokliče podkorak križanje in mu ju poda kot vhodna argumenta.

- **Mutacija**
  Vsakemu otroku, ki je nastal v križanju z verjetnostjo mutacije p_m spremenimo enega ali več genov.

- **Ocenjevanje populacije**
  Nove kromosome v populaciji – otroke ocenimo in jim določimo fitnes oceno.

- **Posodabljanje populacije kromosomov**
  Glede na izbrano metodo populacijo posodobimo. Lahko ohranjamo konstantno velikost populacije ali ji dovolimo da raste.

- **Preverjanje pogojev za zaključek algoritma**
  Ko se izpolni vsaj eden od pogojev, algoritem zaključimo in poiščemo najboljšo rešitev v populaciji.



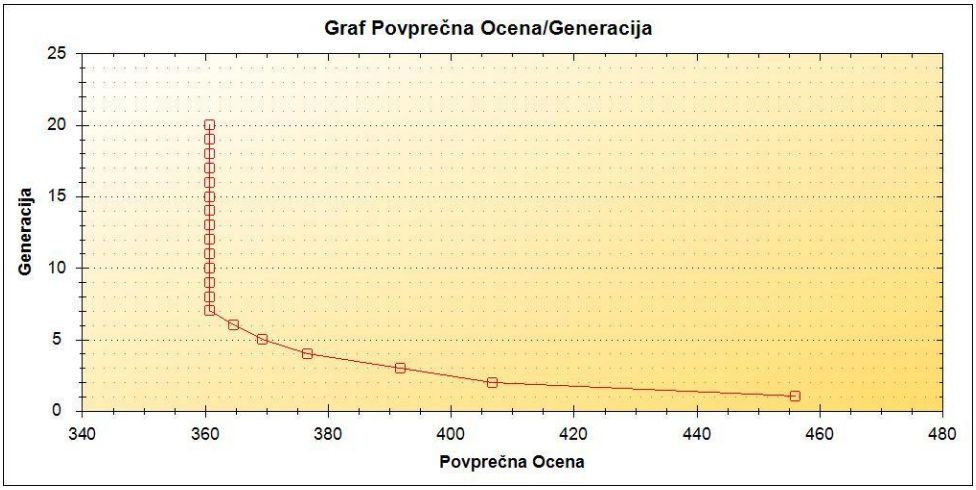
Slika 2. 9: Koraki izvajanja genetskega algoritma

2.3 Selekcija

Selekcija je korak, pri katerem izberemo za križanje dva ali več kromosomov iz populacije, ter ustvarimo otroke za naslednjo generacijo. Selekcija je mehanizem, s katerim poskušamo z dobro izbiro staršev prenašati boljše gene iz generacije v generacijo in se tako bolj približati optimalni rešitvi. Vprašanje je, katere kromosome izbrati iz populacije, da se bodo prenašali najboljši geni. Odgovor je izbira kromosomov glede na selekcijski kriterij. Pri oblikovanju kriterija se v glavnem uporabljata dva pristopa. Prvi pristop pogojuje izbiro kromosoma iz populacije z njegovo fitnes oceno. Boljšo oceno kot ima kromosom, večja je verjetnost, da bo izbran v selekciji. Drugi pristop je izbira kromosoma na podlagi njegove prioritete znotraj populacije. Prvi pristop je najpogostejši in tudi najbolj zanesljiv, vendar trčimo ob problem, ko hitrost konvergence začne upadati. Takrat so razlike med ocenami kromosomov minimalne, torej izbira na podlagi fitnes ocene ni dobra. V tem primeru je bolj učinkovito, da izbiramo na podlagi prioritete.

Omeniti moramo tudi pomembnost selekcijskega kriterija na obnašanje GA. Izbira strožjega kriterija pomeni, da bo selekcija izbirala samo najboljše kromosome, medtem ko šibkejši kriterij poveča količino kromosomov primernih za križanje. V primeru strožjega kriterija se GA hitreje konča pri lokalnem minimumu, ki ne daje optimalne rešitve (slika 2.10). Zgodi se namreč, da se raznolikost kromosomov populacije po nekaj generacijah zgosti v populacijo podobnih rešitev, kjer napredek iz generacije v generacijo postaja vedno manjši ali ničen. Zato moramo pri izbiri selekcijskega kriterija imeti v mislih tudi mutacijo. Dovolj pogosta mutacija lahko znatno ublaži negativne posledice strožjega selekcijskega kriterija.

V primeru prešibkega kriterija je konvergenca GA algoritma bolj počasna in algoritem porabi znatno več generacij za iskanje optimalne rešitve. Ker imamo pri reševanju problemov z GA večinoma opravka z zelo velikimi populacijami, moramo paziti, da kriterij ni prešibek, saj lahko časovna zahtevnost GA postavne prevelika.

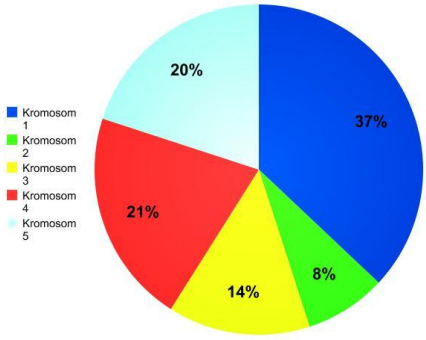


Slika 2. 10: Primer zgodnje konvergence GA ob strogem selekcijskem kriteriju in $p_m = 0$.

V nadaljevanju bom podrobneje predstavil najbolj pogosto uporabljane selekcijske metode.

Naključna ruleta

Naj bo N število kromosomov v populaciji in naj bodo $f_i \ldots f_N$ fitnes ocene posameznih kromosomov populacije. V primeru iskanja globalnega minimuma pri naključni ruleti najprej izračunamo $O = \sum_{i=1}^{N} \frac{1}{f_i}$, nato določimo verjetnost selekcije ps_i za vsak kromosom *i* posebej: $ps_i = \frac{1}{f_i \cdot O}$. Ruleto zavrtimo *N* krat (slika 2.11). Starše izbiramo tako da vsakič pogledamo kje se je ruleta ustavila. Pri naključni ruleti je dovoljeno, da je posamezni kromosom lahko izbran za starša večkrat.



Slika 2. 11: Primer naključne rulete

Naključna selekcija

Iz populacije naključno vzamemo dva kromosoma in ju križamo. To ponovimo $\left\lfloor \frac{N}{2} \right\rfloor$ -krat. Selekcija daje slabše otroke kot naključna ruleta, a ohranja bolj raznoliko populacijo.

Selekcija po prioriteti

Problem pri selekciji z naključno ruleto je, da imajo v primeru posameznih zelo močnih kromosomov ostali kromosomi zelo majhno verjetnost, da bodo izbrani. To povzroči slabšo raznolikost populacije, kar povzroča hitro konvergenco. V takšnih primerih se raje poslužujemo selekcije po prioriteti. Selekcijo realiziramo tako, da vsakemu kromosomu na podlagi njegove fitnes ocene določimo prioriteto; najslabši dobi 1, najboljši pa *N*. Nato *N*-krat izberemo dva naključna kromosoma iz populacije in za prvega starša vzamemo tistega z višjo oceno, zmagovalec naslednje primerjave je drugi starš in tako naprej.

Elitizem

Metoda kot parameter dobi odstotek elitne populacije p_{el}. Starše izbiramo samo iz elitne populacije najboljših kromosomov velikosti p_{el} odstotkov celotne populacije. Sledi $\left\lfloor \frac{N \cdot p_{el}}{2} \right\rfloor$ -krat naključna izbira obeh staršev iz elitnega dela in njuno križanje.

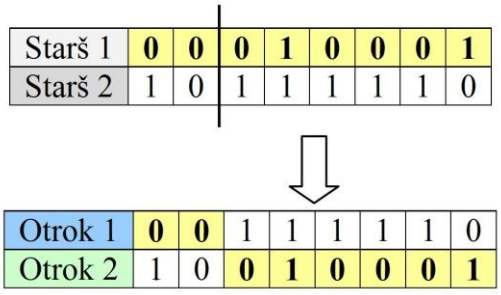
2.4 Križanje

Križanje je proces ki ga izvaja selekcija nad populacijo izbranih kromosomov – potencialnih staršev. Kot vhod dobi dva ali več kromosomov, ki ju križa in ustvari dva ali več otrok. Osnovna ideja križanja je prenesti gene staršev na otroke v upanju, da bodo slednji boljši in bodo obogatili populacijo kromosomov. Otroci so novo ustvarjeni kromosomi, sestavljeni iz kopiranih genov, ki pripadajo dvema ali več staršev. V nadaljevanju bom predstavil nekaj osnovnih načinov križanja. Treba je poudariti, da so ti načini križanja namenjeni za lažje razumevanje konceptov in so pogosto podlaga za ustvarjanje kompleksnejših načinov, namenjenih reševanju težavnejših problemov.

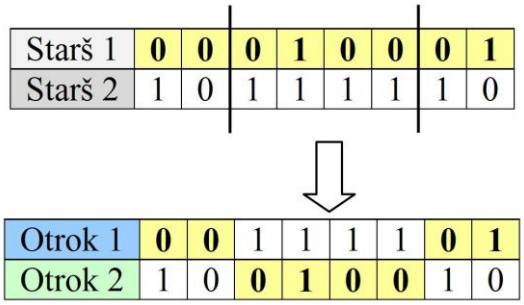
Za vse načine križanja najprej definiramo verjetnost križanja p_{cs}. Če je $p_{cs} = 0$, potem so otroci čiste kopije staršev, če je $p_{cs} = 1$, nad vsakim parom vedno staršev izvedemo križanje. Za lažje razumevanje snovi bomo privzeli da pri predstavljenih načinih križanja starša ustvarita dva otroka.

K-pozicijsko križanje

Naj bo *d* fiksna dolžina kromosomov staršev in $k \in [1, d)$ število pozicij križanja. Ustvarimo polje elementov *c* velikosti *k*, pri katerem je $c_0 = 0, c_{poz} \in \{1,...,d\}$ in $c_{poz} < c_{poz+1}$ za $1 \le poz < k-1$. Križanje deluje tako, da zajamemo niz genov prvega starša od prejšnje pozicije križanja c_{poz-1} do trenutne c_{poz} in jih kopiramo na enako mesto v prvega otroka. Nato kopiramo gene drugega starša od pozicije c_{poz} do c_{poz+1} (ali do *d*, če $c_{poz}+1 > d$) na isto mesto v prvega otroka. Isti postopek ponovimo še za drugega otroka, le da tokrat zamenjamo vrstni red staršev. Najpogosteje v praksi uporabljena števila pozicij križanja sta 1 in 2. Primera križanj za *k=1* in *k=2* sta na slikah 2.12 in 2.13.



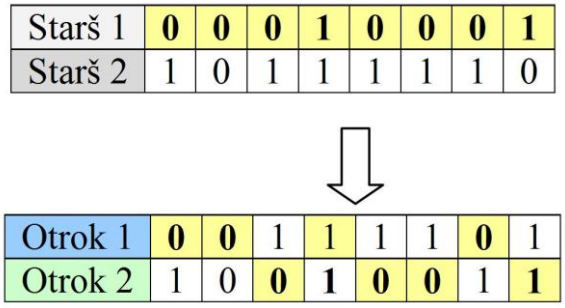
Slika 2. 12: Primer pozicijskega križanja za *k*=1



Slika 2. 13: Primer pozicijskega križanja za *k*=2

Uniformno križanje

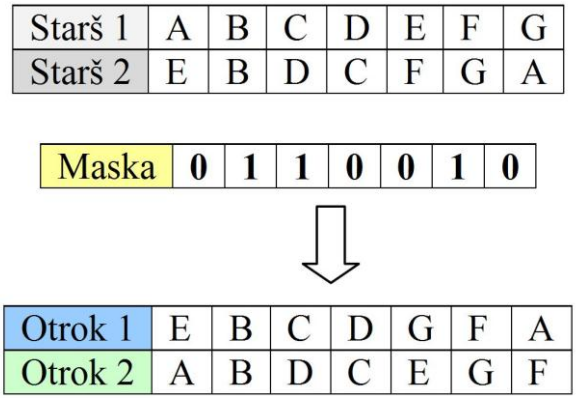
Uniformno križanje izpeljanka *k-pozicijskega* križanja, kjer ima vsak niz genov prvega starša vnaprej dano verjetnost p_u da se prenese na prvega otroka. Če niza genov ne prenesemo od prvega starša, je prenesen niz genov drugega starša. Postopek ponovimo za drugega otroka, le da zamenjamo vrstni red staršev. V večini primerov je $p_u = 0.5$ (slika 2.14).



Slika 2. 14: Primer uniformnega križanja

Prioritetno uniformno križanje

K-pozicijsko in uniformno križanje nista uporabna za probleme, pri katerih kromosomi tvorijo permutacijski kod (npr. problem potujočega trgovca). Pri uporabi binarnih kromosomov za takšne probleme imamo pogosto opravka z velikim deležem populacije ki predstavlja neveljavne rešitve. Z uporabo prioritetnega uniformnega križanja ta problem odpravimo. Po izbiri staršev za križanje ustvarimo polje maska, naključno sestavljeno iz ničel in enk. Število elementov v maski je enako število genov starša. Prvemu otroku prenesemo gene prvega starša, kadar je v maski na isti poziciji enka. Ostala prazna mesta kromosoma otroka zapolnimo z geni drugega starša, razporejenih po vrstnem redu. Pri tem na prazna mestna ne prenašamo genov drugega starša, ki so že prisotni v otroku. Tako odpravimo podvajanje genov v otroku. Isti postopek ponovimo še za drugega otroka, pri čemer obrnemo vrstni red staršev (slika 2.15).



Slika 2. 15: Primer prioritetnega uniformnega križanja

2.5 Mutacija

Glavni namen mutacije je preprečitev hitre konvergence algoritma v lokalne minimume. Mutacija sledi križanju in se pogojno izvede nad vsakim novo nastalim kromosomom - otrokom. Mutacija se izvrši z dano verjetnostjo mutacije $p_m \in [0,100]$, pri tem se spremeni eden ali več genov v kromosomu. Preko mehanizma mutacije lahko poleg novih genov pridobimo nazaj izgubljen genetski material iz predhodnih generacij. Zato je mutacija odlično orodje proti nepopravljivi izgubi genskega materiala, ki nastane kot posledica selekcije. Ohranjanje večje genetske raznolikosti v populaciji kot posledice mutacije nam omogoča izogibanje pasti hitre konvergence, ter iskanje bolj optimalnih rešitev. Kljub temu, da je mutacija močno orodje, moramo upoštevati, da po določenem številu generacij vsak še tako dobro zasnovan GA začne delati zelo majhne korake v smeri optimalne rešitve. V tem primeru večanje faktorja mutacije bistveno ne vpliva na delovanje GA, ker se slednji že močno približuje optimalni rešitvi.

Obračanje bita

Obstaja veliko različnih oblik mutacij za različne vrste zapisa kromosomov. Zaradi lažje predstave sem bom omejil na bitni zapis kromosoma. Najpogostejša oblika mutacije pri bitnemu zapisu je *obračanje bita* mutacija. Metoda obrne vrednost *j*-tega bita v kromosomu otroka iz 0 v 1 in obratno (slika 2.16), kadar je *j*-ti bit mutacijskega kromosoma postavljen na 1. Za verjetnost mutacije p_m se običajno vzame približno $1/d$, kjer je d dolžina kromosoma.

Kromosom starša	**1**	0	1	1	**0**	1	0	**1**
Mutacijski Kromosom	**1**	0	0	0	**1**	0	0	**1**
Kromosom otroka	**0**	0	1	1	**1**	1	0	**0**

Slika 2. 16: Primer mutacije obračanje bita

Izmenjava bitov

Pri izmenjavi bitov naključno izberemo dva bita v kromosomu otroka in izmenjamo njune vrednosti.

Naključno obračanje bitov

Pri naključnem obračanju bitov v kromosomu obrnemo vrednosti enega ali več naključno izbranih bitov

2.6 Posodabljanje populacije kromosomov

Zadnji korak vsake iteracije GA je posodobitev populacije kromosomov. Po križanju in mutaciji dobimo otroke - nove potencialne rešitve problema, ki jih dodajamo v populacijo. V tem koraku imamo opravka z metodami, ki nam pomagajo izbrati množico kromosomov, ki jih bomo odstranili iz populacije. Tako bomo ustvarili novo populacijo, ki bo nastopila v naslednji generaciji.

Pred prvo generacijo GA imamo najprej začetno populacijo kromosomov velikosti *s*. Pri koraku posodabljanja je velikost populacije, ki jo posodobimo, vsako generacijo lahko konstantna ali naraščajoča. Pri konstantnemu posodabljanju populacije *n* kromosomov pred križanjem zamenjamo z *n* kromosomi po križanju, pri tem pa mora veljati: $n \le s \text{ in } n > 0$. Po križanju novo ustvarjeni kromosomi vstopijo v populacijo, ki je sedaj velikosti $n + s$. Da posodobimo populacijo, izberemo poljubno metodo, s katero odstranimo *n* kromosomov iz populacije, in tako po zaključku *i*-te generacije ohranimo prvotno velikost populacije. Če velja $n = s$, zamenjujemo celotno populacijo kromosomov vsako generacijo. V tem primeru imamo opravka z popolnim posodabljanjem populacije. Če pa velja $n < s$ imamo opravka z delnim posodabljanjem populacije, kjer se poveča verjetnost, da poljubni kromosom ostane v populaciji več generacij neodvisno od izbire metode za posodabljanje populacije. Slednja lastnost povečuje raznolikost populacije. Prednost pristopa $n < s$ je redkejša hitra konvergenca GA v lokalne minimume. Potrebno je poudariti, da je število generacij, ki jih potrebujemo za zaključek GA z delnim posodabljanjem populacije vedno večje od GA z popolnim posodabljanjem populacije, a so hkrati rešitve prvega v večini primerov tudi boljše.

Označimo s s_i velikost populacije za generacijo *i*, in s s_0 velikost začetne populacije. Pri naraščajočem posodabljanju populacije *n* kromosomov pred križanjem zamenjamo z *m* kromosomi po križanju, pri tem mora veljati: $n \le s_i$, $m > n$ in $n > 0$. Velikost nove populacije je tako $s_i = s_{i-1} + (m - n)$. Običajno predpostavimo, da se z vsako generacijo sorazmerno z velikostjo populacije s_i povečujeta tudi vrednosti spremenljivk *m* in *n*. Prednosti naraščajočega posodabljanja so podobne delnemu konstantnemu posodabljanju, kjer tudi velja, da se poljubni kromosom pogosteje obdrži več generacij v populaciji, neodvisno od metode posodabljanja. Ker vsako generacijo velikost populacije narašča, se verjetnost, da se poljubni kromosom obdrži v populaciji še dodatno povečuje, kar doprinese še k večji raznolikosti kromosomov kot pri delnemu konstantnemu posodabljanju. Kot smo že poudarili, nas večja raznolikost vedno privede do boljših rešitev. Slabost tega načina posodabljanja je znatno povečanje števila generacij, potrebnih za iskanje optimalne rešitve v primerjavi z popolnim in delnim konstantnim načinom posodabljanja populacije. Hkrati pa se tudi znatno poveča čas izvajanja GA, saj izračun vsake generacije na račun povečevanja populacije traja dlje kot prejšnji. Pri tem pristopu moramo razumno omejiti velikost začetne populacije, da GA ne postane neobvladljivo časovno zahteven.

Metode za posodabljanje populacije

Skozi leta raziskav na področju genetskih algoritmov se je razvilo precejšnje število metod za posodabljanje populacije kromosomov. V tem razdelku se omejimo samo na štiri najpogostejše in najbolj praktične metode.

Počisti vse prednike

Počisti vse prednike je metoda, s katero iz populacije odstranimo vse prednike – starše, ki so nastopali v trenutni generaciji in jih zamenjamo z njihovimi otroki. Pri tej metodi moramo biti pozorni, da staro populacijo zamenjamo vsaj z enako veliko populacijo otrok. Upadanje populacije povzroči predčasni konec GA. Ta metoda je ena izmed najbolj pogosto uporabljenih pri posodabljanju populacije, saj sta njeno razumevanje in implementacija zelo enostavna.

Naključno odstranjevanje

Pri metodi *naključnega odstranjevanja* po križanju vseh staršev iz populacije odstranimo *n* kromosomov. Naključni generator je ključ, po katerem določimo, katere kromosome bomo odstranili. To metodo je tako kot metodo *počisti vse prednike* enostavno razumeti in realizirati, a hkrati zaradi naključnosti dopušča večjo verjetnost ohranjanja poljubnega kromosoma več generacij, kar doprinese k večji raznolikost populacije.

Odstranjevanje slabih staršev

Pri metodi *odstranjevanja slabih staršev* po križanju (in morebitni mutaciji) prednostno odstranimo iz populacije slabe starše. Slabe starše določimo po njihovi fitnes oceni ali pa po prioriteti. Ker pa vseh otrok ne moremo prenesti v novo populacijo, prenesemo samo najboljše otroke. Slabih otok, tako kot slabih staršev ne prenesemo v naslednjo generacijo. Tako imamo v novi populaciji najboljše starše in otroke, kar zelo izboljša povprečno oceno populacije.

Odstranjevanje najslabših iz populacije

Po zaključeni selekciji in križanju v populacijo dodamo nove kromosome - otroke. Med celotno novo populacijo (otrok in staršev) se izbere *n* kromosomov. Tako kot pri metodi *odstranjevanja slabih staršev* kromosome za odstranitev določimo na podlagi fitnes ocene ali prioritete. Ker je metoda po delovanju podobna metodi *slabih staršev* tudi izboljšuje povprečno oceno populacije iz generacijo v generacijo. Za razliko od prej omenjene metode pa število odstranjenih staršev in otrok ni v enakomernem razmerju. Zato metoda ni tako toga in v večini primerov daje nekoliko boljše rezultate.

2.7 Zaključek genetskega algoritma

Za zaključek GA mora bit izpolnjen najmanj eden izmed v naprej definiranih pogojev s strani načrtovalca algoritma glede na naravo problema, ki ga rešuje. Spodaj so navedeni najbolj pogosti pogoji za zaključek GA.

- **Čas izvajanja algoritma je potekel.** Algoritem se ustavi, ko doseže vnaprej določen čas izvajanja. Če pred iztekom časa GA doseže maksimalno število generacij, se algoritem ustavi.

• **Dosežen maksimum števila generacij.** Algoritem se ustavi, ko je doseženo vnaprej določeno maksimalno število generacij.

• **Fitnes ocena najboljšega kromosoma je konstantna *z* generacij.** Če algoritem ugotovi, da je fitnes ocena najboljšega kromosoma konstantna zadnjih *z* generacij, potem se algoritem zaključi. V tem primeru imamo dve možni rešitvi: ali je algoritem padel v past lokalnega minimuma ali pa smo našli optimalno rešitev glede na podane vhodne parametre. Oba tipa rešitev se da v večini primerov ločiti po naslednjih indikatorjih: celotno število izvedenih generacij algoritma in *z*. Večje ko je število izvedenih generacij in manjši kot je *z*, večja je verjetnost optimalne rešitve – globalni minimum, v nasprotnem primeru imamo opravka z lokalnim minimumom.

• **Zanemarljiva rast povprečne fitnes ocene generacije.** Označimo s pc_i povprečno fitnes oceno generacije *i*. Predpostavimo da iščemo globalni minimum, ter določimo rast Δv_i povprečne fitnes ocene v generaciji *i*: $\Delta v_i = pc_{i-1} - pc_i$. Definirajmo še sprejemljiv odstotek odstopanja $pr \in (0,0.01]$. Če velja: $\Delta v_i < pr \cdot pc_i$, potem je rast povprečne fitnes ocene zanemarljiva in algoritem ustavimo.

• **Našli smo najboljšo posamezno rešitev.** Algoritem se zaustavi, ko se izpolni kriterij konvergence za najboljšo posamezno rešitev v populaciji.

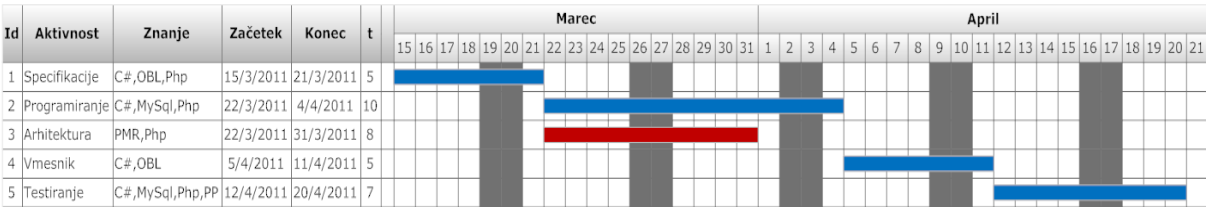
3 Predstavitev problema upravljanja programskih projektov

Programski projekt sestavljajo medsebojno povezane aktivnosti z danimi časi trajanja in potrebnimi znanji. Aktivnosti je treba izvesti v čim krajšem času in/ali s čim manj stroški in z minimalnim prekrivanjem aktivnosti, pri čemer so na voljo viri: zaposleni z znanji. Aktivnosti projekta in vire je potrebno organizirati tako, da bomo glede na zastavljene cilje – kriterije, izvedli projekt optimalno.

Aktivnosti projekta so v našem primeru koraki pri izdelavi programske rešitve. Primeri korakov pri izdelavi programske rešitve so: izdelava specifikacij iz naročnikovih zahtev, postavljanje arhitekture sistema, programiranje posameznih funkcionalnosti, itd. Znanje kot vir ni opredeljeno količinsko, ampak zgolj kot množica, ki povezuje aktivnosti z zaposlenimi. Vsak zaposleni ki sodeluje na projektu, ima določen nabor znanj, ki mu omogočajo sodelovanje na aktivnosti projekta. Stroške projekta v našem primeru tvorijo samo plače zaposlenih, ki sodelujejo na aktivnostih projekta. Ostalih stroškov, kot so licence programske opreme in najemnina prostorov, ne štejemo med stroške projekta. Potek projekta pogosto opišemo z Ganttovim diagramom (slika 3.1), kjer je razvidno, kdaj potekajo posamezne aktivnosti.

Programski projekt je podan z vhodnimi podatki: množico aktivnosti – projektom in njegovi viri, zaposlenimi z znanji. *Problem upravljanja programskega projekta* (PUPP) je optimizacijska naloga (D, f, opt), kjer je D množica dopustnih rešitev, $f : D \to \Re$ je kriterijska (ocenjevalna) funkcija in $opt = \{\min\}$. Dopustne rešitve PUPP bomo imenovali *kromosomi*, ker bodo v naslednjem poglavju pri reševanju PUPP z genetskim algoritmom predstavljale kromosome GA. Množico D in funkcijo f predstavimo v posebnem razdelku.

V tem poglavju podrobno predstavimo vhodne podatke pri PUPP, podrobno opišemo množico dopustnih rešitev, matematično zapišemo kriterijsko funkcijo in prikažemo ocenjevanje poljubnega kromosoma pri podanem zgledu programskega projekta. Opišemo tudi problem prekrivanja aktivnosti in podamo možne rešitve. Realizacija tretjega poglavja temelji na virih [1], [8] in [10]. Metoda kritične poti temelji na virih [6] in [15].



Slika 3. 1: Ganttov diagram programskega projekta *Pv* iz razdelka 3.4

3.1 Opis projekta in virov

Projekt Pv je sestavljen iz aktivnosti $Ak_1, Ak_2, ..., Ak_{Sp}$, vsaka aktivnost Ak_j ima podano:

- seznam prednikov aktivnosti: Ak_j^{pred}
- trajanje aktivnosti: Ak_j^t
- seznam zahtevanih znanj: Ak_j^{zz}

Projekt Pv zapišemo kot množico aktivnosti $Pv = \{Ak_1, Ak_2, ..., Ak_{Sp}\}$ velikosti Sp, kjer slednja predstavlja število aktivnosti, ki sestavljajo projekt. Projekt Pv predstavimo kot usmerjeni aciklični graf $G(V, A)$, $V = \{Ak_1, Ak_2, ..., Ak_{Sp}\}$, kjer $(Ak_i, Ak_j) \in A$, če aktivnost Ak_i zaključimo brez prekinitev in nato pričnemo z Ak_j. Trajanje aktivnosti Ak_j označimo Ak_j^t, množico prednikov aktivnosti z $Ak_j^{pred} \subset Pv$ in zahtevana znanja za izvajanje aktivnosti $Ak_j^{zz} \subseteq Znanje$. Trajanje aktivnost je celoštevilska vrednost podana v enoti dni/zaposleni.

Viri projekta Pv so zaposleni: $Zp_1, Zp_2, ..., Zp_{Sz}$ z znanji: $Zn_1, Zn_2, ..., Zn_{Szn}$, ki so potrebna za izvajanje projekta. Vsak zaposleni Zp_i ima podano:

- množica znanj zaposlenega: Zp_i^{znanja}
- mesečno plačilo: Zp_i^{placa}
- maksimalno predanost projektu: $Zp_i^{\max ef}$

Zaposlene zapišemo kot množico $Zaposleni = \{Zp_1, Zp_2, ..., Zp_{Sz}\}$ velikosti Sz. Znanje, ki nastopa v projektu, zapišemo kot množico $Znanje = \{Zn_1, Zn_2, ..., Zn_{Szn}\}$ velikosti Szn. Znanje posameznega zaposlenega Zp_i je množica $Zp_i^{znanja} \subseteq Znanje$. Mesečno plačilo zaposlenega je realna vrednost zapisana v valuti €. Mesečno plačilo predstavlja plačilo zaposlenega, če je ta stoodstotno zaposlen z izvajanjem aktivnosti projekta Pv ves mesec, za katerega prejme plačilo. Če je zaposleni zaposlen manj, dobi temu sorazmerno manjše plačilo. Maksimalna predanost projektu $maxef$ lahko zavzame eno izmed treh vrednosti: 0, 50, 100. Vrednost 0 pomeni, da zaposleni ne sodeluje na projektu in ob tem ne prejme mesečnega plačila za čas izvajanja projekta. Vrednost 50 pomeni, da ima zaposleni čas sodelovati na projektu samo polovico delovnika, njegovo mesečno plačilo je zato sorazmerno manjše. Vrednost 100 pomeni da zaposleni lahko izkoristi celotni čas delovnika za sodelovanje na projektu. Maksimalna predanost projektu nam omogoči, da ključne zaposlene lahko razporedimo na več projektov ($maxef$ = 50) ali pa jih premestimo na druge projekte ($maxef$ = 0). Za tri vrednosti 0, 50 in 100 smo se odločili zaradi lažje posplošitve problema dodeljevanja zaposlenih pri izvajanju več projektov hkrati.

3.2 Množica dopustnih rešitev

Množico rešitev za PUPP tvorijo poljubne rešitve - kromosomi *M*, ki jih predstavimo z matriko $M = (m_{ij})$, kjer so vrstice označene z zaposlenimi *Zp*, stolpci pa z aktivnostmi *Ak* projekta *Pv*. Element m_{ij} predstavlja posamezni gen kromosoma *M*. Število stolpcev *i* je velikost množice *Zp*, medtem ko je število vrstic *j* velikost množice *Ak*. Število genov, ki sestavljajo kromosom *M*, je tako $|Ak| \cdot |Zp|$. Vsak gen m_{ij} je predstavljen kot pozitivna realna vrednost na intervalu [0,1], ki predstavlja odstotek časa sodelovanja zaposlenega Zp_i pri izvajanju aktivnosti Ak_j. Zaposleni Zp_i mora sodelovati $Akt_j^t \cdot m_{ij}$ časa pri izvajanju aktivnosti Ak_j. Koliko ur svojega delovnega časa bo posvetil projektu, si zaposleni uravnava sam, pomembno je samo, da izpolni obvezo sodelovati čas $Akt_j^t \cdot m_{ij}$ na projektu Ak_j. Pri PUPP predpostavimo, da nobeden od zaposlenih nima časa za sodelovanje na posamezni aktivnosti več kot maksimalnih 100 odstotkov. Če zaposleni nima vsaj enega od potrebnih znanj za opravljanje aktivnosti Ak_j in tako ne sodeluje pri aktivnosti, potem pri aktivnosti Ak_j sodeluje z odstotkom 0.

Da je poljubna rešitev - kromosom *M*, lahko član množice dopustnih rešitev *D*, mora zadoščati naslednjim pogojem:

- Če je $m_{ij} > 0$, mora zaposleni *i* imeti vsaj eno od zahtevanih znanj za opravljanje aktivnosti *j*.
- $m_{ij} \le \frac{Zp_i^{\max ef}}{100}$.
- Vsako od aktivnosti Ak_j je možno izvesti: $Ak_j^{zz} \subseteq \bigcup_{i:m_{ij}>0} Zp_i^{znanja}$.

3.3 Kriterijska funkcija

Kriteriji za iskanje optimalne rešitve PUPP so:

- čim manjše trajanje kritične poti projekta,
- čim manjši stroški izvedbe projekta,
- čim manjše prekrivanje aktivnosti projekta.

Torej cilj je poiskati kromosom – dopustno rešitev, ki najbolj ustreza vsem trem kriterijem. Ustreznost kromosoma podamo v obliki fitnes ocene kromosoma. Manjša kot je fitnes ocena, boljši je kromosom. Fitnes oceno vsakemu kromosomu izračuna kriterijska (ocenjevalna) funkcija.

Določitev kriterijske funkcije

Kriterijska funkcija opravlja nalogo ločevanja slabih rešitev od dobrih. Pri izračunu fitnes ocene Oc_M kromosoma *M* definiramo naslednje parametre: u_{kp} - utež kritične poti, t_{kp} - čas izvajanja kritične poti, u_{Str} - utež stroškov, S_{str} - stroški projekta, u_{pr} - utež prekrivanja aktivnosti in t_{pr} - čas izvajanja vseh prekrivajočih aktivnosti, kjer velja $u_{kp}, u_{Str}, u_{pr} \in [0,1]$. Fitnes oceno Oc_M za kromosom *M* v množici dopustnih rešitev *D* izračunamo po formuli: $Oc_M = u_{kp} \cdot t_{kp} + u_{str} \cdot S_{str} + u_{pr} \cdot t_{pr}$.

Projekt traja toliko časa, kolikor traja najdaljša pot v grafu projekta. Imenujemo jo *kritična pot*. Množico aktivnosti projekta, ki nastopajo v kritični poti projekta označimo s *KPA*. Množico *KPA* določimo z *metodo kritične poti*. Čas trajanja projekta izračunamo torej tako, da seštejemo trajanja aktivnosti na kritični poti:

$$t_{kp} = \sum_{j \in KPA} \frac{Ak_j^t}{\sum_{i=1}^{Z} m_{ij}} .$$

Stroške S_{str} izračunamo po tako, da seštejemo plačila zaposlenih za opravljanje posameznih aktivnosti:

$$S_{str} = \sum_{k=1}^{Z} \left(\sum_{j=1}^{A} \left(\frac{Akt_j^t}{\sum_{i=1}^{Z} m_{ij}} \cdot m_{kj} \right) \cdot \frac{Zp_i^{placa}}{30} \right) .$$

Ker so stroški vrednostno mnogo večji od t_{kp} in t_{pr}, stroške pri izračunu fitnes ocene normaliziramo na njun nivo: $S_{str} = \frac{S_{str}}{|Ak| \cdot |Zp|}$.

Čas prekrivanja izračunamo tako, da od vsote trajanj vseh aktivnosti odštejemo čas trajanja projekta: $t_{pr} = \left(\sum_{j=1}^{A} \frac{Akt_j^t}{\sum_{i=1}^{Z} m_{ij}} \right) - t_{kp}$.

Metoda kritične poti

Pri izračunu trajanja časa izvajanja kritične poti projekta t_{kp} uporabimo metodo kritične poti (MKP) za določitev množice aktivnosti *KPA*. MKP je deterministična metoda, ki se uporablja za razvrščanje aktivnosti projektov, kadar je možno natančno oceniti čas trajanja

posameznih aktivnosti. Kritično pot opredelimo kot zaporedje aktivnosti projekta, ki ima najdaljši čas trajanja od začetka do konca izvedbe projekta. Aktivnosti, ki sestavljajo kritično, pot ti. *kritične aktivnosti*, želimo izvesti v čim krajšem času, ker pogojujejo trajanje projekta. Metoda kritične poti je natančno opisana v [4]. Kritične aktivnosti lahko poskušamo skrajšati z dodeljevanjem večje količine virov projekta. Ker ima PUPP vir zaposlene, jih poskušamo preusmeriti na kritične aktivnosti. Ostale aktivnosti, ki niso del kritične poti, imajo manjšo prioriteto izvedbe. Časovna zahtevnost algoritma je polinomskega reda [9].

Postopek za skrajšanje kritične poti:

- Za vsako aktivnost ocenimo:
  1. Običajni čas trajanja tc_n in njene stroške oz. običajno ceno c_n.
  2. Minimalni čas za dokončanje aktivnosti oz. izjemni čas tc_i in njene izjemne stroške oz. izjemno ceno c_i.

- Vmesne vrednosti ocenimo s pomočjo premice, ki aproksimira odvisnost med trajanjem in stroški, in ima naklon $\frac{c_i - c_n}{tc_n - tc_i}$.
- Čas aktivnosti krajšamo tako, da povečujemo zmogljivosti ali izberemo dražje rešitve.
- Algoritem

  a. Izračunamo čas trajanja projekta in določimo kritične aktivnosti.

  b. Izračunamo naklone vseh premic aktivnosti.

  c. Izberemo aktivnost na kritični poti, ki ima najmanjši naklon in jo je možno še skrajšati.

  f. V primeru, da moramo skrajšati dve ali več aktivnosti hkrati, seštejemo stroške vseh aktivnosti in upoštevamo najdaljši možni čas.

  d. Izračunamo nov čas trajanja projekta, določimo nove kritične aktivnosti in izračunamo nove stroške aktivnosti projekta.

  e. Postopek ponavljamo, dokler ni več aktivnosti na kritični poti, ki jih lahko še skrajšamo.

3.4 Primer rešitve programskega projekta

Za lažje razumevanje PUPP podajmo zgled projekta izdelave programskih rešitev *Pv*, ki ga tvori pet aktivnosti *Ak*, ki so pogoste v projektih izdelave programskih rešitev, ter viri: pet zaposlenih z znaji. Povezave med posameznimi aktivnostmi so prikazane na sliki 3.3.

V projektu *Pv* nastopa množica znanj *Znanje* = {*C#*, *MySql*, *Php*, *PP*, *OBL*}.

PP je okrajšava za postavljanje mrež in OBL za oblikovanje vmesnika. C#, MySql in Php so programski jeziki, ki se uporabljajo pri izvajanju aktivnosti.

Definirajmo projekt $Pv = \{Ak_1, Ak_2, Ak_3, Ak_4, Ak_5\}$, kjer je

$Ak_1^{naziv} = specifikacije,\ Ak_1^{t} = 5,\ Ak_1^{zz} = \{C\#, OBL, PMR\}$,

$Ak_2^{naziv} = programiranje,\ Ak_2^{prednik} = \{specifikacije\},\ Ak_2^{t} = 10,\ Ak_2^{zz} = \{C\#, MySql, Php\}$,

$Ak_3^{naziv} = arhitektura,\ Ak_3^{prednik} = \{specifikacije\},\ Ak_3^{t} = 8,\ Ak_3^{zz} = \{PMR, Php\}$,

$Ak_4^{naziv} = vmesnik,\ Ak_4^{prednik} = \{programiranje\},\ Ak_4^{t} = 7,\ Ak_4^{zz} = \{C\#, OBL\}$,

$Ak_5^{naziv} = testiranje,\ Ak_5^{prednik} = \{arhitektura, vmesnik\},\ Ak_5^{t} = 5,\ Ak_5^{zz} = \{C\#, Php, MySql, PP\}$.

Podajmo še množico zaposlenih, ki sodelujejo na projektu *Pv* *Zaposleni* = $\{Zp_1, Zp_2, Zp_3, Zp_4, Zp_5\}$, kjer je

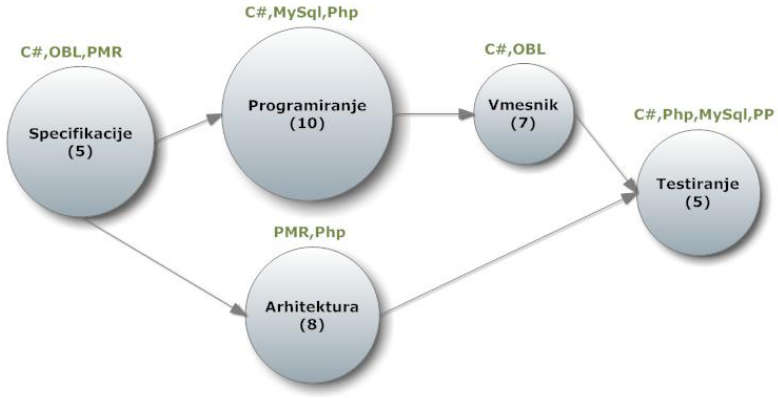
$Zp_1^{naziv} = Dejan,\ Zp_1^{placa} = 1100,\ Zp_1^{znanja} = \{C\#, MySql, Php\},\ Zp_1^{\max ef} = 100$,

$Zp_2^{naziv} = Jani,\ Zp_2^{placa} = 900,\ Zp_2^{znanja} = \{Php, PP\},\ Zp_2^{\max ef} = 100$,

$Zp_3^{naziv} = Pero,\ Zp_3^{placa} = 1100,\ Zp_3^{znanja} = \{Php, PMR, OBL\},\ Zp_3^{\max ef} = 50$,

$Zp_4^{naziv} = Tilen,\ Zp_4^{placa} = 900,\ Zp_4^{znanja} = \{OBL, PMR, PP\},\ Zp_4^{\max ef} = 100$,

$Zp_5^{naziv} = Miha,\ Zp_5^{placa} = 1000,\ Zp_5^{znanja} = \{C\#, MySql\},\ Zp_5^{\max ef} = 100$.



Slika 3. 2: Primer grafa *G(V,A)* projekta *Pv*

Primer izračuna fitnes ocene dopustne rešitve

Na primeru dopustne rešitve za projekt *Pv* (slika 3.2) z viri *Zaposleni* izračunajmo fitnes oceno kromosoma *M* iz slike 3.3. Preverimo lahko, da *M* zadošča vsem pogojem za članstvo v množici dopustnih rešitev *D* (razdelek 3.2).

Čas izvajanja aktivnosti t_j v projektu *Pv* je odvisen od trajanja aktivnosti Ak_j^t in vsote odstotkov časa sodelovanja zaposlenih (ki so pogojeni z zahtevanim znanjem): $t_j = \dfrac{Akt_j^t}{\sum m_{ij}}$. Poglejmo na primer čas izvajanja aktivnosti postavitev arhitekture t_3.

Izračunamo čas trajanja aktivnosti: $t_3 = \frac{8}{0.44+0.84+0+0.56+0} = 4dni$. Nato na enak način izračunamo čase trajanja aktivnosti projekta $t_1 = 3$, $t_2 = 5$, $t_4 = 3$ in $t_5 = 2$. Aktivnosti na kritični poti določimo z metodo kritične poti. V našem primeru kritično pot sestavljajo aktivnosti: specifikacije, programiranje, vmesnik in testiranje. Trajanje kritične poti je $t_{kp} = 3+5+3+2 = 13$ dni. Trajanje časa prekrivanja aktivnosti izračunamo: $t_{pr} = \sum t_i - t_{kp} =$ 17 – 13 = 4 dni. Stroške izračunamo s pomočjo enačbe za izračun stroškov S_{str} (razdelek 3.3) in znašajo 1178 €. Pri, za naš primer, izbranih utežeh $u_{kp} = 0.95$, $u_{Str} = 0.15$ in $u_{pr} = 0.5$ je ocena kromosoma *M*: $Oc_M = u_{kp} \cdot t_{kp} + u_{str} \cdot S_{str} / (|Ak| \cdot |Zp|) + u_{pr} \cdot t_{pr} = 0.95 \cdot 13 +$ $1178/(5 \cdot 5) \cdot 0.15 + 0.5 \cdot 4 = 21.42$.

M	Specifikacije	Programiranje	Arhitektura	Vmesnik	Testiranje
Dejan	**0,72**	**0,84**	**0,44**	**0,84**	**0,84**
Jani	**0**	**0,56**	**0,84**	**0**	**0,44**
Pero	**0**	**0,28**	**0**	**0,44**	**0,16**
Tilen	**0,84**	**0**	**0,56**	**1**	**0,56**
Miha	**0,28**	**0,44**	**0**	**0,16**	**0,28**

Slika 3. 3: Primer kromosoma *M* projekta *Pv*

3.5 Problem prekrivanja aktivnosti

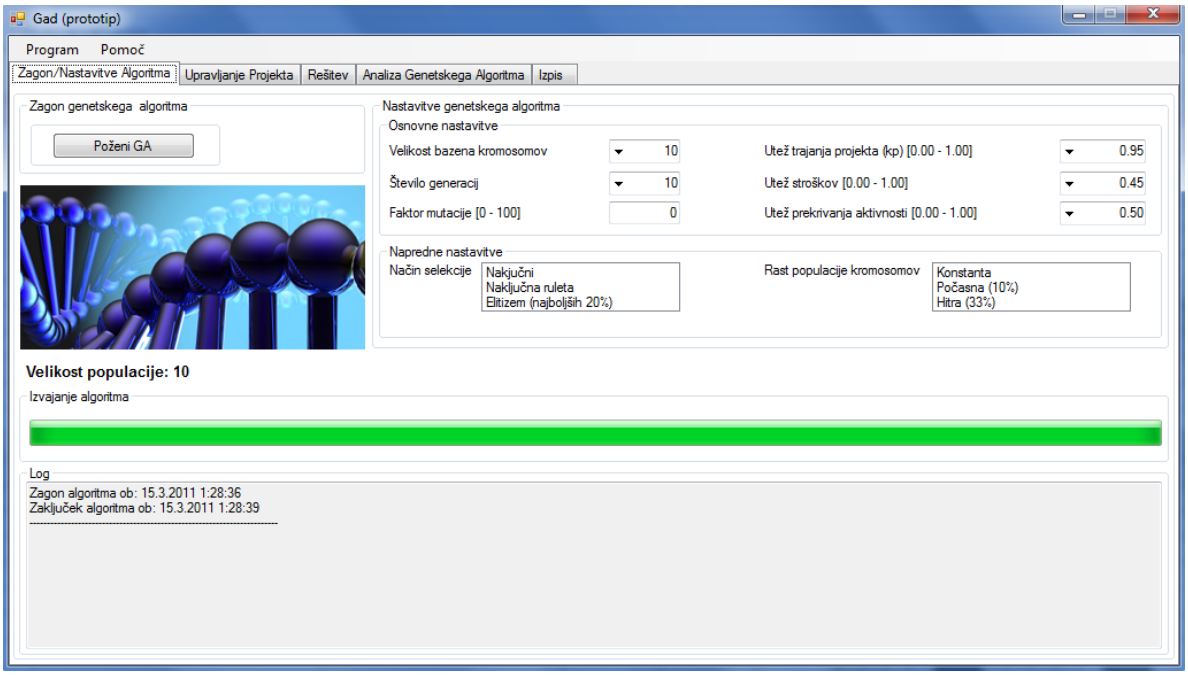
V projektih pogosto pride do časovnih obdobij izvajanja dveh ali več aktivnosti hkrati. To imenujemo *prekrivanje aktivnosti*. Problem nastane, kadar ima zaposleni znanje za sodelovanje v dveh ali več prekrivajočih se aktivnostih. V kromosomu *M* (slika 3.3) pri zaposlenemu Janiju opazimo, da zaradi prekrivajočih aktivnosti programiranje in arhitektura (slika 3.2), njegova vsota odstotkov časa sodelovanja (0,56 in 0,84) na projektu *Pv* za določen čas naraste na več kot 1. To pomeni, da v času istočasno se izvajajočih aktivnosti programiranje in arhitektura Jani v resnici dela nadure. Določeno število nadur je razumljivo za vsak projekt, ki temelji na programskih rešitvah, vendar lahko preveliko število in dolga obdobja nadur povzročijo nizko koncentracijo zaposlenih, to pa pomeni, da se posledično poveča število napak, kar ima za posledico porast trajanja in stroškov projekta. Proti prevelikemu število nadur in prekrivanju projekta se borimo z večanjem uteži prekrivanja aktivnosti u_{pr} in manjšanjem uteži kritične poti u_{kp}. Tako genetski algoritem (GA) usmerjamo k rešitvam z manjšim prekrivanjem (in posledično manj nadurami) in stran od rešitev z krajšimi kritičnimi potmi. Lahko pa problem rešimo z zaposlovanjem novih ljudi z zahtevanim znanjem med izvajanjem projekta.

4 Realizacija genetskega algoritma za upravljanje programskih projektov

V drugem poglavju smo predstavili genetski algoritem, njegove gradnike in njegovo podrobneje delovanje. V tretjem poglavju smo nato podrobneje predstavili problem upravljanja programskih projektov, njegove vhodne podatke, množico dopustnih rešitev in kriterijsko funkcijo. Cilj tega poglavja je realizacija genetskega algoritma za upravljanje programskih projektov (GAUPP). Skozi poglavje bomo predstavili podrobno delovanje in korake tega algoritma. Algoritem spada v družino algoritmov za načrtovanje razvrstitev aktivnosti projekta in je zastavljen kot orodje za načrtovanje programskih projektov. GAUPP načrtovalcu pomaga organizirati aktivnosti in zaposlene tako, da se projekt zaključi v čim krajšem roku in/ali s čim manj stroški in z minimalnim prekrivanjem aktivnosti.

Na podlagi četrtega poglavja sem izdelal aplikacijo v jeziku C# in programskem okolju Visual Studio .NET 2010, ki temelji na GAUPP. Aplikacija ima pregleden uporabniški vmesnik (UV), možnost shranjevanja in odpiranja projektov v obliki tekstovne datoteke, ter grafični prikaz rešitve in konvergence povprečne fitnes ocene. Osrednji namen aplikacije je predstavitev in testiranje GAUPP. Z aplikacijo so opravljene meritve v petem poglavju.

Realizacija GAUPP temelji na virih: [1], [8] in [10]. Grafi v aplikaciji so izdelani s pomočjo odprtokodne rešitve [17]. Implementacija algoritma metode kritične poti v aplikaciji temelji na viru [16].



Slika 4. 1: Prikaz uporabniškega vmesnika aplikacije, ki uporablja GAUPP

4.1 Opis algoritma

Podrobno izvajanje korakov GAUPP predstavimo s psevdokodo, opisano spodaj. Pri izvajanju GAUPP se izvajajo koraki GA, ki so podrobneje opisani v razdelku 2.2.4. Pri GAUPP bomo realizirali tri selekcijske metode: naključna ruleta, naključna selekcija in elitizem (razdelek 4.4). Realiziramo tudi tri načine rasti populacije: konstantno rast, počasno rast in hitro rast (razdelek 4.6). Vhod GAUPP so vhodni podatki: aktivnosti projekta, zaposleni, seznam znanja in parametri algoritma: izbira selekcijske metode, načina rasti, uteži fitnes ocene, velikost začetne populacije, število generacij in faktor mutacije. Izhod je najboljši kromosom zadnje generacije populacije.

Psevdokoda **GAUPP**:

Vhod:
Vhodni podatki: aktivnosti projekta *Pv*, zaposleni *Zaposleni* in seznam znanja *Znanje*
Parametri: *Ut1* - utež kritične poti, *Ut2* - utež stroškov, *Ut3* - utež prekrivanja, *St_gen* - število generacij, *Vel_pop* – velikost začetne populacija, *St_sel* – selekcijska metoda, *St_pop* – način rasti populacije, *Pm* – faktor mutacije

---

začni

velikost_pop = *Vel_pop*, *i* = 0

ustvarimo začetno naključno populacijo *P(i)* na podlagi vhodnih podatkov: *Pv*, *Zaposleni* in *Znanje*
ocenimo začetno populacijo *P(i)*, uporabimo parametre *Ut1*, *Ut2* in *Ut3*

ponavljaj dokler ((i < *St_gen*) ali (uporabnikova prekinitev izvajanja algoritma))
Selekcija(*St_sel*, *P(i)*)
posodobi populacijo *P(i)* z elementi *O(i)*. Način rasti določa parameter *St_pop*
i = *i* + 1
ponavljaj

Rx = poišči najboljšo rešitev v *P(i)*

konec

funkcija Selekcija (vhod: selekcijska metoda *St_sel*, populacija *P(i)*)
j = 0
Parameter *St_sel* določi izračun število križanj *stc*

ponavljaj dokler (*j* < *stc*)
glede na selekcijsko metodo *St_sel* izberi starša *S1* in *S2*
Križanje(*S1*, *S2*). Dobimo otroka *O1* in *O2*
rnd = naključno število v intervalu 0 do 100

pogoj (*rnd* > *Pm*)
Mutacija(O1, O2)
konec pogoj

ocenimo *O1* in *O2*, za oceno podamo argumente *Ut1*, *Ut2* in *Ut3*
Dodamo *O1* in *O2* v *O(i)*
j = *j* + 1
ponavljaj

konec funkcija Selekcija (izhod: nova populacija: *O(i)*)

Izhod: Rešitev *Rx*

4.2 Sestavljanje kromosomov

Na podlagi definicij vhodnih podatkov GAUPP iz razdelka 3.1 bomo kromosom – dopustno rešitev zapisali v matrični obliki. To obliko bomo podrobneje opisali spodaj.

Matrični zapis dopustne rešitve

Pri matričnem zapisu vrednost gena m_{ij} kromosoma *M* predstavimo z eno izmed osmih približno enakomerno porazdeljenih realnih vrednosti na intervalu [0,1]. Torej vsak gen m_{ij} opišemo z poljem treh bitov, kjer pozitivni bit 1 predstavlja vrednost 0.56, pozitivni bit 2 vrednost 0.28, medtem ko pozitivni bit 3 zaokrožimo na vrednost 0.16. Če je katerikoli bit ničen, je potem vrednost, ki jo bit predstavlja enaka 0. Tako binarni zapis gena 111 predstavlja vrednost natanko 1 in zapis gena 000 vrednost natanko 0. Pomembnost bitov gena je torej razvrščena od leve proti desni. Vsak gen lahko zavzame natanko eno izmed naslednjih realnih vrednosti: 0, 0.16, 0.28, 0.44, 0.56, 0.72, 0.84 in 1. Te vrednosti predstavljajo približen odstotek časa sodelovanja zaposlenega Zp_i pri izvajanju aktivnosti Ak_j in projektnemu vodji služijo kot kazalec pri organizaciji zaposlenih. Na sliki 4.2 si lahko ogledamo primer matričnega zapisa kromosoma *M*.

Celotni kromosom lahko tako predstavimo kot polje bitov velikosti $|Ak| \cdot |Zp| \cdot 3$, kar nam omogoča izdelavo genetskih algoritmov (GA), ki temeljijo na bitnem zapisu kromosoma. GAUPP uporablja kromosome v bitnem zapisu dolžine tri. Za binarni zapis dolžine tri, imamo sest vrednosti, ter 0 in 1, dovolj vrednosti za opis približnih odstotkov časa sodelovanja slehernega zaposlenega. Binarni zapis velikosti dva omogoča premalo raznolikih vrednosti za opis vrednosti časa sodelovanja, medtem ko binarni zapis dolžine štiri, predstavlja prevelik razpon vrednosti, ter pomembno vpliva na povečanje časovne zahtevnosti problema.

Pravila sestavljanja kromosomov

Pri korakih GAUPP: križanje, mutacija in ustvarjanje začetne populacije tvorimo nove kromosome - dopustne rešitve problema. Pri vseh teh korakih se moramo držati treh pravil sestavljanja kromosomov, drugače novo ustvarjen kromosom ni veljaven in ga ne dodamo v

populacijo. Pravila so izpeljana iz pravil za članstvo dopustne rešitve v množici dopustnih rešitev. Delovanje pravil je predstavljeno na primeru projekta *Pv* iz razdelka 3.4. Pravila so naslednja:

1. Vrednost 0, kjer zaposlenimi nima znanja za aktivnost

Za vsakega zaposlenega *i*, ki nima vsaj enega od potrebnih znanj za izvajanje aktivnosti *j*, mora veljati $m_{ij} = 0$. Ničlo na tej poziciji moramo ohranjati med celotnim izvajanjem algoritma, zato pri načrtovanju križanja in mutacije izberemo metode ki ohranjajo to pravilo. Pri kromosomu *M* projekta *Pv* (slika 4.2) so rdeče pobarvani geni kromosoma, kjer velja prvo pravilo. Na primer, Tilen ne more sodelovati pri aktivnosti programiranje, saj nima nobenega od potrebnih znanj (C#, MySql, Php), zato je $m_{42} = 0$.

M	Specifikacije	Programiranje	Arhitektura	Vmesnik	Testiranje
Dejan	**0,72**	**0,84**	**0,44**	**0,84**	**0,84**
Jani	**0**	**0,56**	**0,84**	**0**	**0,44**
Pero	**0,56**	**0,84**	**0,56**	**1**	**0,72**
Tilen	**0,84**	**0**	**0,56**	**1**	**0,56**
Miha	**0,28**	**0,44**	**0**	**0,16**	**0,28**

Slika 4. 2: Primer zaposlenih z vsemi zahtevanimi znanji za aktivnosti

2. Normalizacija vrstic zaposlenih

Normalizacija kromosoma poskrbi, da vrednosti genov kromosoma $M = (m_{ij})$ odražajo maksimalno predanost *maxef* vsakega zaposlenega *i* na projektu. Spremenljivka *maxef* lahko zavzame tri vrednost: 0, 50, 100. V primeru *maxef* = 100 normalizacija ne spremeni nobenih vrednosti v vrstici *i*. Če je 0, postavimo vrednosti vseh genov v vrstici *i* na 0. V primeru, da je enaka 50, normalizacija izklopi (postavi na 0) prvi bit vsakega gena v vrstici *i*. Pri kromosomu *M* projekta *Pv* (slika 4.3) je rdeče pobarvana tretja vrstica. Ker je Petrova maksimalna predanost *maxef* = 50, je vrstica normalizirana.

M	Specifikacije	Programiranje	Arhitektura	Vmesnik	Testiranje
Dejan	**0,72**	**0,84**	**0,44**	**0,84**	**0,84**
Jani	**0**	**0,56**	**0,84**	**0**	**0,44**
Pero	**0,56**	**0,84**	**0,56**	**1**	**0,72**
Tilen	**0,84**	**0**	**0,56**	**1**	**0,56**
Miha	**0,28**	**0,44**	**0**	**0,16**	**0,28**

⇩

M	Specifikacije	Programiranje	Arhitektura	Vmesnik	Testiranje
Dejan	**0,72**	**0,84**	**0,44**	**0,84**	**0,84**
Jani	**0**	**0,56**	**0,84**	**0**	**0,44**
Pero	**0**	**0,28**	**0**	**0,44**	**0,16**
Tilen	**0,84**	**0**	**0,56**	**1**	**0,56**
Miha	**0,28**	**0,44**	**0**	**0,16**	**0,28**

Slika 4. 3: Primer normalizacije za tretjega zaposlenega ob $Max_{ef} = 50$

3. Če ena ali več aktivnosti ni obdelanih, povečamo oceno

Kromosomu *M* izračunamo fitnes oceno Oc_k. Nato preverimo, ali je vsaka od aktivnosti v projektu obdelana s strani vsaj enega zaposlenega (slika 4.4). Če najdemo vsaj eno neobdelano aktivnost, kromosomu povečamo oceno: $Oc_k = Oc_k \cdot 100$. To pravilo omogoča zelo preprost mehanizem za izločanje nedopustnih kromosomov iz populacije vsako generacijo.

M	Specifikacije	Programiranje	Arhitektura	Vmesnik	Testiranje
Dejan	**0**	**0,84**	**0,44**	**0,84**	**0,84**
Jani	**0**	**0,56**	**0,84**	**0**	**0,44**
Pero	**0**	**0,84**	**0,56**	**1**	**0,72**
Tilen	**0**	**0**	**0,56**	**1**	**0,56**
Miha	**0**	**0,44**	**0**	**0,16**	**0,28**

Slika 4. 4: Primer, kjer prva aktivnost ni obdelana

4.3 Ustvarjanje začetne populacije kromosomov

Pred začetkom izvajanja prve generacije GAUPP najprej ustvarimo začetno populacijo kromosomov. V GAUPP so kromosomi začetne populacije ustvarjeni naključno. Naključno ustvarjeni kromosomi morajo ustrezati vsem trem pravilom sestavljanja kromosomov.

Novo populacijo, ob upoštevanju pravil sestavljanja kromosomov, ustvarimo v naslednjih korakih. Naprej ustvarimo nov kromosom z naključnimi vrednostmi genov. Pri vstavljanju naključnih vrednosti v gene moramo upoštevati, da ima vsak zaposleni, ki nima vsaj enega zahtevanega znanja za opravljanje določene aktivnosti, vrednost gena enako 0. Sledi normalizacija kromosoma v skladu z drugim pravilom. Nato izračunamo fitnes oceno kromosoma in preverimo, ali je v kromosomu kakšna aktivnost, na kateri ne dela noben zaposleni. Če najdemo eno ali več takšnih aktivnosti, fitnes oceno pomnožimo s 100, drugače oceno pustimo takšno, kot je bila. Ustvarjanje začetne populacije opišemo s psevdokodo, navedeno spodaj.

Psevdokoda **ustvarjanje začetne populacije**:

---

začni

ustvarimo populacijo *P(0)* velikosti 0
velikost_pop = Zp
    ponovi (*Zp- krat*)
        Ustvari nov naključni kromosom, pri čemer vstavimo 0 kjer zaposleni nima znanja za aktivnost
        Normalizacija kromosoma
        Izračunamo Oc_k kromosoma
        pogoj (če ostaja ena ali več aktivnosti ki jih ne opravlja noben zaposleni)
            $Oc_k = Oc_k \cdot 100$
        konec pogoj
        nov kromosom dodamo v *P(0)*
    ponavljaj ponovi

konec

Izhod: populacija *P(0)*

4.4 Selekcijske metode

Kot smo že v razdelku 2.3 omenili, je glavni namen selekcije poiskati primerne starše za križanje v upanju, da ustvarimo otroke, ki bodo dvignili povprečno fitnes oceno populacije. Pri implementaciji imamo na voljo več selekcijskih metod. V GAUPP so podprte (implementirane) tri selekcijske metode: naključna ruleta, naključna selekcija in elitizem. Vse selekcijske metode izberejo dva starša. Katero metodo bo algoritem izvedel, uporabnik izbere s seznama na uporabniškemu vmesniku (UV). Elitizem ima konstantno velikost elitne populacije 20%.

Podprte selekcijske metode smo že teoretično podrobneje predstavili v drugem poglavju, zato bomo njihovo implementacijo v GAUPP le opisali s psevdokodo.

Psevdokoda **naključne rulete**:

Vhod: populacija *P(i)*

---

začni

velikost_pop = velikost *P(i)*

Suma_Ocen = izračunamo vsoto vseh ocen 1/ Oc_k , $k \in P(i)$
Ruleta_list = seznam velikosti *vel_pop* vrednosti *fi*
Za vsak kromosom izračunamo njegov delež ocene $fi_k = 1/(Oc_k \cdot Suma\_Ocen)$

ponovi ($\lfloor velikost\_pop/2 \rfloor$ - *krat*)
Starš1 = **Zavrti_ruleto**(*velikost_pop*)
Starš2 = **Zavrti_ruleto**(*velikost_pop*)
Križanje (*Starš1*, *Starš2)*. Dodamo *Otrok1* in *Otrok2* v *O(i)*
Mutacija (*Otrok1*, *Otrok2)*. Posodobimo *O(i)*
ponavljaj ponovi

konec

funkcija Zavrti_ruleto (vhod: *vel_pop*)
ustvarimo naključno število $r \in [0,1]$, *i* = 1, *inx* = 0

ponavljaj dokler (ni konec seznama *Ruleta_list*)
pogoj (r > *Ruleta_listi[i])*
inx = *i* + 1
konec pogoj
i = *i* + 1
ponavljaj

konec funkcija Zavrti_ruleto (izhod: indeks kromosoma *inx*)

Izhod: nova populacija *O(i)*

Psevdokoda **naključne selekcije**:

Vhod: populacija *P(i)*

---

začni

velikost_pop = velikost *P(i)*

ponovi ($\lfloor velikost\_pop / 2 \rfloor$ - *krat*)
ustvarimo dve različni naključni celi števili $r1, r2 \in [0, \text{velikost\_pop}]$
Starš1 = r1
Starš2= r2
Križanje (*Starš1*, *Starš2)*. Dodamo *Otrok1* in *Otrok2* v *O(i)*
Mutacija (*Otrok1*, *Otrok2)*. Posodobimo *O(i)*
ponavljaj ponovi

konec

Izhod: nova populacija *O(i)*

Psevdokoda **elitizma**:

Vhod: populacija *P(i)*

---

začni

velikost_pop = velikost *P(i)*
ustvari začasno populacijo *B(i)* ki jo sestavlja 20% najboljših kromosomov
velikost_el = je velikost *B(i)*

ponovi ($\lfloor velikost\_pop \rfloor / 2$ - *krat*)
ustvarimo dve različni naključni celi števili $r1, r2 \in [0, \text{velikost\_el}]$
Starš1 = r1
Starš2= r2
Križanje (*Starš1*, *Starš2)*. Dodamo *Otrok1* in *Otrok2* v *O(i)*
Mutacija (*Otrok1*, *Otrok2)*. Posodobimo *O(i)*
ponavljaj ponovi

konec

Izhod: nova populacija *O(i)*

4.5 Križanje in mutacija

Pri križanju in mutaciji imamo na voljo več različnih metod križanja, ki smo jih podrobneje obdelali v razdelku 2.4. Pri realizaciji GAUPP smo uporabili (dva starša in dva otroka) eno-pozicijsko križanje ($k=1$) z vnaprej definirano pozicijo križanja $c_{poz}=\left\lfloor \frac{d}{2} \right\rfloor$, kjer je d dolžina kromosoma. Verjetnost križanja je: $p_{cs}=1$, se pravi nad vsakim parom staršev vedno izvedemo križanje (slika 4.5). Za eno-pozicijsko križanje smo uporabili, ker ohranja prvi dve pravili sestavljanja kromosomov (razdelek 4.3), kjer konstantni c_{poz} učinkovito odvrača GAUPP od hitrih konvergenc. Ali oba otroka ustrezata tretjemu pravilu, preverimo po izračunu fitnes ocen novonastalih kromosomov po morebitni mutaciji.

Starš 1

0,72	0,84	0,44	0,84	0,84
0	0,56	0,84	0	0,44
0	0,28	0	0,44	0,16
0,84	0	0,56	1	0,56
0,28	0,44	0	0,16	0,28

⟺

Starš 2

0,84	0,44	0,16	0,72	0,72
0	0,44	0,72	0	0,16
0	0,16	0,44	0,28	0,28
0,72	0	0,44	0,84	1
0	0,28	0	0,28	0,16

Otrok 1

0,72	0,84	0,44	0,84	0,84
0	0,56	0,84	0	0,44
0	0,16	0,44	0,28	0,28
0,72	0	0,44	0,84	1
0	0,28	0	0,28	0,16

Otrok 2

0,84	0,44	0,16	0,72	0,72
0	0,44	0,72	0	0,16
0	0,28	0	0,44	0,16
0,84	0	0,56	1	0,56
0,28	0,44	0	0,16	0,28

Slika 4. 5: Primer eno-pozicijskega križanja starša 1 in 2

Mutacija, ki jo uporablja GAUPP, je izpeljanka mutacije *naključno obračanje bitov*, opisane v razdelku 2.5. Pri naključnem obračanju bitov obrnemo vrednosti enega ali več naključno izbranih bitov v kromosomu. Pri naši metodi naključno spremenimo vrednosti vseh genov v naključno izbrani vrstici i kromosoma $M=(m_{ij})$ (slika 4.6). Pri tem popravimo naključno spremenjeno vrstico kromosoma M, da ohrani ničle, ki so pogojene z prvim pravilom sestavljanja kromosomov. Drugo pravilo nad mutiranim kromosomom ohranimo tako, da izpeljemo normalizacijo. Faktor mutacije p_m uporabnik nastavi preko uporabniškega vmesnika (UV).

Otrok 1

0,72	0,84	0,44	0,84	0,84
0	0,56	0,84	0	0,44
0	0,16	0,44	0,28	0,28
0,56	1	0,84	0,72	0,16
0	0,28	0	0,28	0,16

Slika 4. 6: Primer mutacije otroka 1

4.6 Posodabljanje populacije in zaključek programa

Zadnji korak vsake generacije GAUPP je izvršitev metode za posodabljanje populacije. Kot vhod po selekciji in križanju dobimo novo populacijo otrok *O(i)*, na izhod podamo novo populacijo *P(i)*, katere velikost odraža izbran način rasti populacije. Uporabnik ima preko uporabniškega vmesnika (UV) na izbiro naslednje načine: *konstanta* rast, ki definira velikost nove populacije enako velikosti *P(i)* na začetku te generacije, *počasna* rast (10% na generacijo) in *hitra* rast (33%). Pri počasni in hitri rasti imamo opravka z naraščajočo populacijo. V GAUPP je uporabljena metoda *odstranjevanje najslabših iz populacije* za posodabljanje populacije. Metoda opisana v razdelku 2.6. Deluje tako, da iz skupne populacije staršev *P(i)* in otrok *O(i)* izberemo in odstranimo *n* najslabših kromosomov, kjer je *n* izračunan v skladu z izbranim načinom rasti populacije. Po opravljenemu posodabljanju kromosome populacije *O(i)* prenesemo v *P(i)*, ki sedaj predstavlja novo izhodno populacijo. Podrobnejše delovanje vseh treh načinov rasti populacije je prikazano v psevdokodi spodaj.

Psevdokoda **posodabljanje populacije**:

Vhod: populacija *P(i), O(i)*

---

začni

velikost_pop_n = je velikost *P(i)* + *O(i)*, *pocisti_velikost* = 0

pogoj (če je izbrana konstanta rast populacije*)*
pocisti_velikost = velikost_pop_n / 2
konec pogoj

pogoj (če je izbrana počasna rast populacije*)*
pocisti_velikost = velikost_pop_n – (*velikost_pop_n * 0.55)*
konec pogoj

pogoj (če je izbrana hitra rast populacije*)*
pocisti_velikost = velikost_pop_n / 3
konec pogoj

ponovi (⌊pocisti_velikost ⌋- *krat*)
odstrani element z največjo fitnes oceno najden v *P(i)* ali *O(i)*
ponavljaj ponovi

prenesi elemente *O(i)* v *P(i)*

konec

Izhod: nova populacija *P(i)*

Zaključek GAUPP

GAUPP se zaključi na enega izmed dveh načinov opisanih v razdelku 2.7. GAUPP se zaključi v dveh primerih: ali je doseženo maksimalno število generacij ali pa je uporabnik prek UV prekinil izvajanje algoritma. Kot rešitev se izpiše najboljši kromosom zadnje generacije.

4.7 Ocena časovne zahtevnosti algoritma

V GAUPP kromosom predstavimo kot matriko, kjer so vrstice označene z zaposlenimi iz množice zaposlenih *Zp*, stolpci pa z aktivnostmi iz množice aktivnosti *Ak* projekta *Pv*. Število genov, ki sestavljajo posamezni kromosom, je tako enako $|Ak| \cdot |Zp|$. Pri ocenitvi časovne zahtevnosti GAUPP vzamemo najslabši primer, kjer je $|Ak| = |Zp|$. Če označimo $n = |Ak|$, ima torej kromosom n^2 genov.

Naprej ocenimo časovno zahtevnost posameznih genetskih operacij: ustvarjanje, križanje, mutacija in izračun fitnes ocene kromosoma. Pri operacijah ustvaritve, križanja, mutacije kromosoma se n^2- krat izvede ali ustvarjanje ali kopiranje ali mutiranje genov kromosoma. Pri določitvi časovne zahtevnosti genetskih operacij bomo privzeli, da trajanje operacije ustvarjanja, kopiranja in mutiranja gena traja natanko 1 časovno enoto. Zato časovno zahtevnost korakov ustvarjanja, križanja in mutacije ocenimo z $O(n^2)$. Pri operaciji izračuna fitnes ocene se naprej n^2- krat shrani (kopira) vrednost gena, nato se izvrši metoda kritične poti in izračuna fitnes ocena kromosoma. Ker metoda kritične poti in izračun fitnes ocene ne presegajo časovne zahtevnosti reda $O(n^2)$, operacijo izračuna fitnes ocene ocenimo z $O(n^2)$.

Označimo s s_0 velikost začetne populacije, število generacij GAUPP z i in velikost populacije v generaciji i s s_i. Pri GAUPP pred začetkom prve generacije naprej izvedemo ustvaritev začetne populacije, kjer ustvarimo s_0 kromosomov. Torej časovno zahtevnost ustvarjanja začetne populacije ocenimo $O(s_o \cdot n^2)$. GAUPP nato vsako generacijo izvede selekcijo $\left\lfloor \frac{s_i}{2} \right\rfloor$ - krat. Po selekciji nato posodobi populacijo kromosomov. Časovna zahtevnost koraka posodabljanja populacije je linearne časovne zahtevnosti in jo pri izračunu ne upoštevamo. Pri selekciji se zaporedno izvedejo koraki križanja, mutacije in izračuna fitnes ocene kromosoma. Časovno zahtevnost selekcije tako ocenimo z $O(s_i \cdot n^2)$.

Torej, pri i generacijah je časovno zahtevnost GAUPP ocenimo z $O(\sum_i s_i \cdot n^2)$, se pravi da ima GAUPP časovno zahtevnost *polinomskega* reda glede na velikost projekta. V primeru konstantne populacije je časovna zahtevnost linearno odvisna od velikosti začetne populacije in števila generacij. V primeru počasne ali hitre rasti populacije pa je časovna zahtevnost eksponentna glede na število generacij.

5 Meritve

V tem poglavju bomo predstavili meritve, opravljene na GAUPP, ki smo ga predstavili v 4. poglavju. Pri slednjem smo realizirali tri selekcijske metode in tri načine rasti populacije, ki jih lahko uporabnik poda kot vhodne parametre preko uporabniškega vmesnika (UV) pred zagonom algoritma. Če odštejemo (tudi preko UV nastavljive) vhodne parametre uteži fitnes ocene, določitev začetne velikosti populacije, število generacij in faktor mutacije imamo devet (tri selekcijske metode z tremi načini rasti populacije) možnosti zagona algoritma – meritev testnega primera $\mathrm{Pr}_i = \{Pv_i, Zaposleni_i, Znanje_i\}$. Na naključno generirani množici testnih primerov Pr velikosti 10 bomo iskali kombinacijo izbire selekcijske metode in načina rasti populacije, ki bo dajala najboljše rešitve. Tako bomo empirično pokazali katera kombinacija GAUPP najde najboljše rešitve. Uteži fitnes ocene, število generacij in faktor mutacije so konstantni v vseh meritvah. Velikosti začetne populacije je enaka za vse meritve z konstantno in počasno rastjo populacije, medtem ko je drastično zmanjšana v meritvah s hitro rastjo populacije. Število generacij je za vse meritve omejeno na 20. Razlog za omejitve je neobvladljiva rast trajanja GAUPP vsako generacijo, kadar imamo opravka z hitro rastjo populacije. Vsako meritev opravimo samo enkrat. Za generiranje vhodne množice testnih primerov smo uporabili algoritem opisan podrobneje s psevdokodo spodaj. Vsa naključna števila so *enakomerno* porazdeljena.

Psevdokoda **algoritma za generiranje naključne množice testnih primerov**:

Vhod: *vel_test* – velikost množice testnih primerov

Začni

i = 0, jp=0, jza=0, jzn=0, ustvarimo prazno množico testnih primerov Pr velikosti *vel_test*

ponavljaj_dokler (i < *vel_test*)

jzn = določimo naključno število na intervalu [2,10]
ustvarimo naključno množico znanj $Znanje_i$ velikosti *jzn*

ustvarimo prazno množico Pv_i
jp = določimo naključno število na intervalu [5,20]
j=0
ponavljaj_dokler (*j* < *jp*)
ustvarimo novo aktivnost Ak_j
Ak_j^t = naključno določimo trajanje aktivnosti v intervalu [10,100]
$Ak_j^{prednik}$ = ustvarimo naključno množico prednikov aktivnosti *j*, ki je podmnožica Pv_i. Aktivnost Akt_0 nima prednikov
Ak_j^{zz} = ustvarimo naključno množico zahtevanega znanja aktivnosti, ki je podmnožica $Znanje_i$
aktivnost Ak_j dodamo v množico Pv_i
j = *j* + 1
ponavljaj dokler

jza = določimo naključno število na intervalu [2,10]
ustvarimo naključno množico zaposlenih $Zaposleni_i$ velikosti *jza*
j=0

ponavljaj_dokler (*j* < *jza*)
zp_j^{placa} = naključno določimo mesečno plačilo v intervalu [900,1700]
$zp_j^{\max ef}$ = naključno določimo maksimalno predanost: 0 ali 50 ali 100
zp_j^{znaja} = ustvarimo naključno množico znanja zaposlenega *j*, ki je podmnožica $Znanje_i$
$j = j + 1$
ponavljaj dokler

$\Pr_i = \{Pv_i, Zaposleni_i, Znanje_i\}$
$i = i + 1$
ponavljaj dokler

konec

Izhod: množica testnih primerov Pr

Vhodni parametri GAUPP:

Velikost začetne populacije (konstanta in počasna rast) $s_0 = 200$
Velikost začetne populacije(hitra rast) $s_0 = 40$
Število generacij $i = 20$
Faktor mutacije $p_m = 23$

Utež kritične poti $u_{kp} = 0.95$
Utež stroškov $u_{Str} = 0.20$
Utež prekrivanja aktivnosti $u_{pr} = 0.5$

Seznam selekcijskih metod: *naključna, naključna ruleta, elitizem*
Načini rasti populacije (na generacijo)*: konstanta, počasna (10%), hitra (33%)*

5.1 Podrobnejši prikaz meritev za prvi testni primer

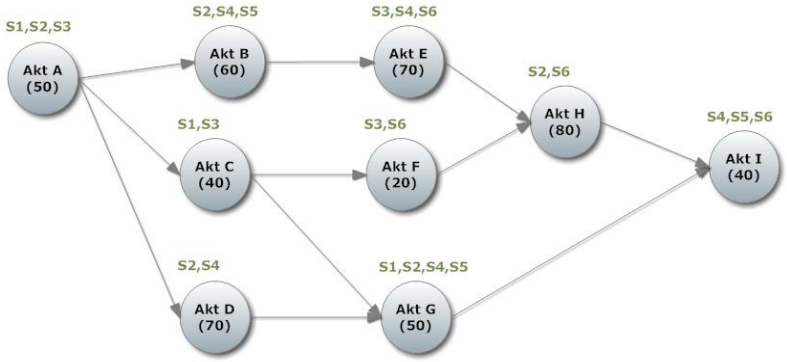
Za lažjo predstavitev delovanja GAUPP, za prvi testni primer (slika 5.1) $\Pr_1 = \{Pv_1, Zaposleni_1, Znanje_1\}$ prikažemo grafe (razdelek 5.1.1) vseh devet meritev (kombinacij izbire metode selekcije in načina rasti populacije), ter podrobneje izpišemo (razdelek 5.1.2) najboljšo rešitev. Vrednosti rešitev ostalih testnih primerov so zajete v tabeli meritev (razdelek 5.2). Za vsak testni primer $\Pr_i$ in meritev *m* definiramo fitnes oceno rešitve Oc_{im}.

Opis testnega primera Pr_1

Za izvedbo projekta Pv_i potrebujemo znanja $Znanje_1 = \{Zn_1, Zn_2, Zn_3, Zn_4, Zn_5, Zn_6\}$ z nazivi $Zn_1^{naziv} = S1,\ Zn_2^{naziv} = S2,\ Zn_2^{naziv} = S3,\ Zn_4^{naziv} = S4,\ Zn_5^{naziv} = S5,\ Zn_6^{naziv} = S6.$

Projekt Pv_i sestavlja devet aktivnosti $\{Ak_1, Ak_2, Ak_3, Ak_4, Ak_5, Ak_6, Ak_7, Ak_8, Ak_9\}$. Kako si aktivnosti sledijo je prikazano na sliki 5.1. Aktivnosti imajo naslednje čase trajanja in zahtevanja znanja:

$Ak_1^{naziv} = Akt\ A,\ Ak_1^{t} = 50,\ Ak_1^{zz} = \{S1, S2, S3\}$
$Ak_2^{naziv} = Akt\ B,\ Ak_2^{prednik} = \{Akt\ A\},\ Ak_2^{t} = 60,\ Ak_2^{zz} = \{S2, S4, S5\}$
$Ak_3^{naziv} = Akt\ C,\ Ak_3^{prednik} = \{Akt\ A\},\ Ak_3^{t} = 40,\ Ak_3^{zz} = \{S1, S3\}$
$Ak_4^{naziv} = Akt\ D,\ Ak_4^{prednik} = \{Akt\ A\},\ Ak_4^{t} = 70,\ Ak_4^{zz} = \{S2, S4\}$
$Ak_5^{naziv} = Akt\ E,\ Ak_5^{prednik} = \{Akt\ B\},\ Ak_5^{t} = 70,\ Ak_5^{zz} = \{S3, S4, S6\}$
$Ak_6^{naziv} = Akt\ F,\ Ak_6^{prednik} = \{Akt\ C\},\ Ak_6^{t} = 20,\ Ak_6^{zz} = \{S3, S6\}$
$Ak_7^{naziv} = Akt\ G,\ Ak_7^{prednik} = \{Akt\ C, Akt\ D\},\ Ak_7^{t} = 50,\ Ak_7^{zz} = \{S1, S2, S4, S5\}$
$Ak_8^{naziv} = Akt\ H,\ Ak_8^{prednik} = \{Akt\ E, Akt\ F\},\ Ak_8^{t} = 80,\ Ak_8^{zz} = \{S2, S6\}$
$Ak_9^{naziv} = Akt\ I,\ Ak_9^{prednik} = \{Akt\ G, Akt\ H\},\ Ak_9^{t} = 40,\ Ak_9^{zz} = \{S4, S5, S6\}$



Slika 5. 1: Graf Pv_1

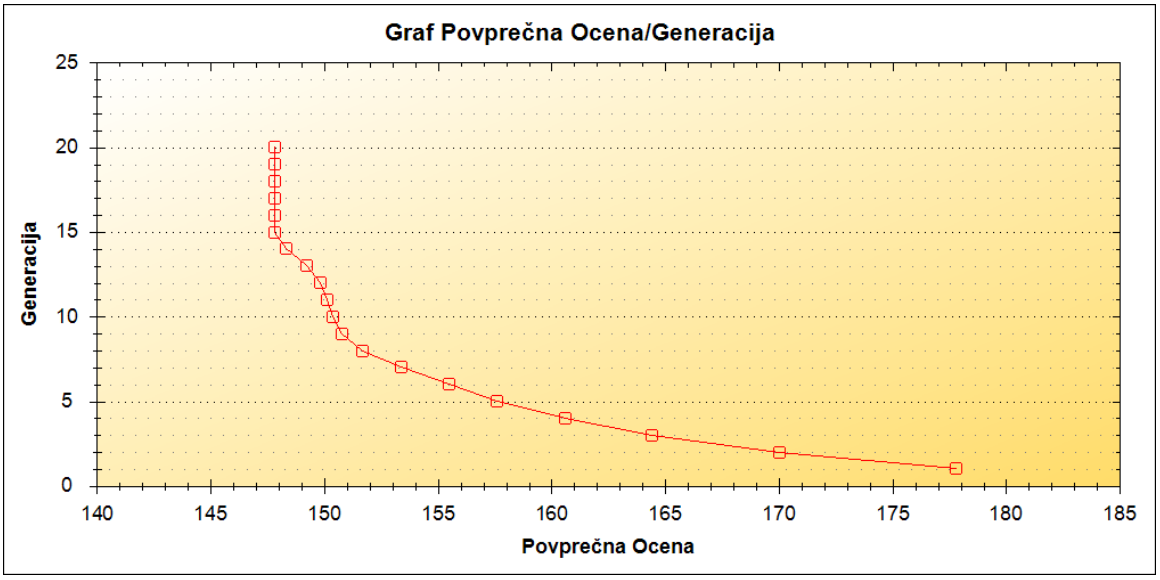
Za delo na projektu je na voljo množica zaposlenih $Zaposleni_1 = \{Zp_1, Zp_2, Zp_3, Zp_4, Zp_5, Zp_6, Zp_7\}$, ki imajo podane naslednje plače, znanja in maksimalne predanosti:

$Zp_1^{naziv} = Z1, Zp_1^{placa} = 1200, Zp_1^{znanja} = \{S1, S2, S3\}, Zp_1^{\max ef} = 100$
$Zp_2^{naziv} = Z2, Zp_2^{placa} = 1400, Zp_2^{znanja} = \{S1, S3, S4, S5\}, Zp_2^{\max ef} = 100$
$Zp_3^{naziv} = Z3, Zp_3^{placa} = 900, Zp_3^{znanja} = \{S4, S6\}, Zp_3^{\max ef} = 50$
$Zp_4^{naziv} = Z4, Zp_4^{placa} = 1600, Zp_4^{znanja} = \{S1, S3, S4, S6\}, Zp_4^{\max ef} = 100$
$Zp_5^{naziv} = Z5, Zp_5^{placa} = 1100, Zp_5^{znanja} = \{S2, S3, S5\}, Zp_5^{\max ef} = 100$
$Zp_6^{naziv} = Z6, Zp_6^{placa} = 1200, Zp_6^{znanja} = \{S2, S4, S6\}, Zp_6^{\max ef} = 100$
$Zp_7^{naziv} = Z7, Zp_7^{placa} = 1000, Zp_7^{znanja} = \{S2, S4\}, Zp_7^{\max ef} = 50$

5.1.1 Grafi meritev testnega primera

Na testnem primeru smo pognali GA za vseh devet nastavitev. Na grafih so prikazali povprečne ocene vsako generacijo. Za vsak primer izpišemo še fitnes oceno najboljše najdene rešitve.

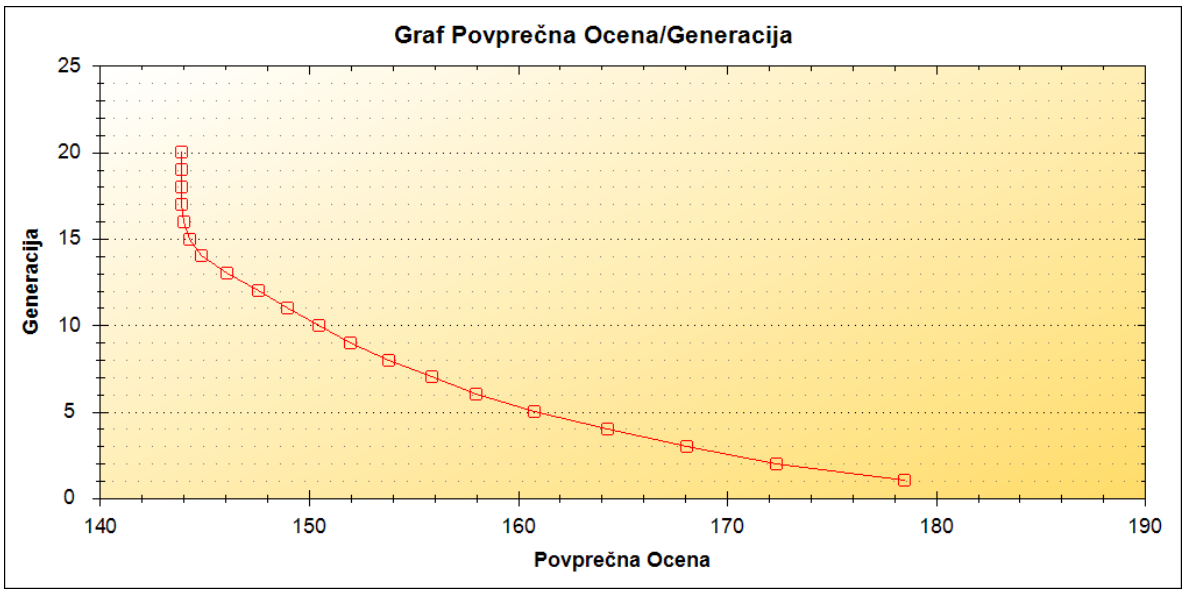
1. Meritev - selekcijska metoda: *naključna*, rast populacije*: konstanta* (slika 4.2)



Slika 5. 2 Graf Pr_1 (naključna, konstantna)

Fitnes ocena rešitve $Oc_{11} = 147{,}80$, velikost končne populacije $s_{20} = 200$.

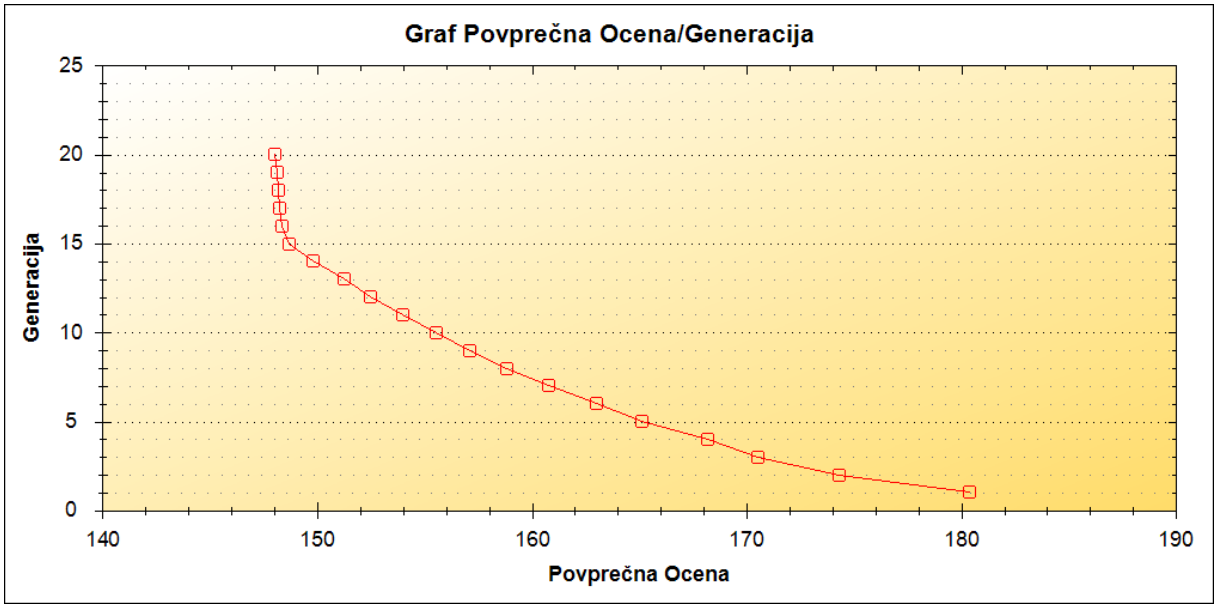
2. Meritev - selekcijska metoda: *naključna*, rast populacije*: počasna* (slika 4.3)



Slika 5. 3: Graf Pr_1 (naključna, počasna)

Fitnes ocena rešitve $Oc_{12} = 142{,}65$, velikost končne populacije $s_{20} = 1333$.

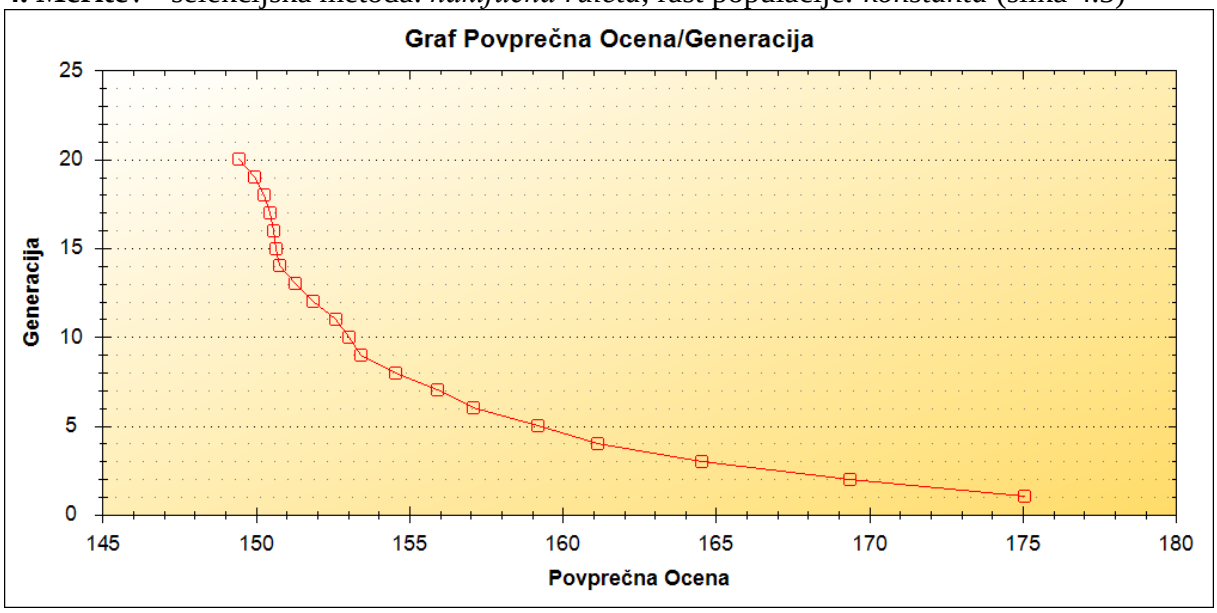
3. Meritev - selekcijska metoda: *naključna*, rast populacije*: hitra* (slika 4.4)



Slika 5. 4: Graf Pr_1 (naključna, hitra)

Fitnes ocena rešitve $Oc_{13} = 143{,}00$, velikost končne populacije $s_{20} = 12844$.

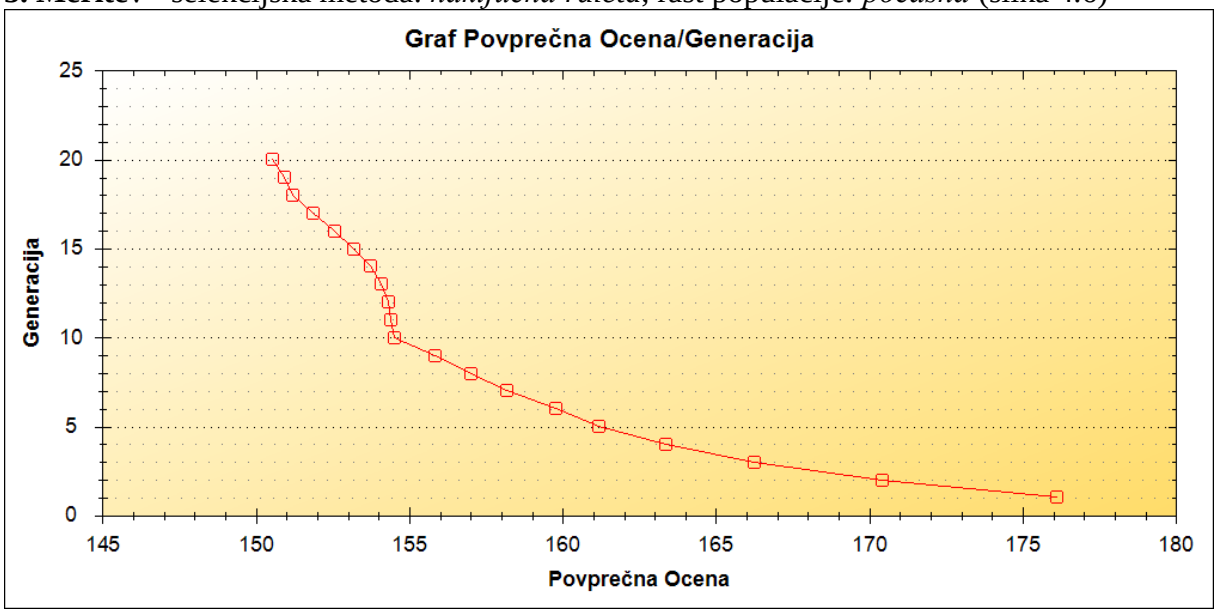
4. Meritev - selekcijska metoda: *naključna ruleta*, rast populacije*: konstanta* (slika 4.5)



Slika 5. 5: Graf Pr_1 (naključna ruleta, konstantna)

Fitnes ocena rešitve $Oc_{14} = 148{,}30$, velikost končne populacije $s_{20} = 200$.

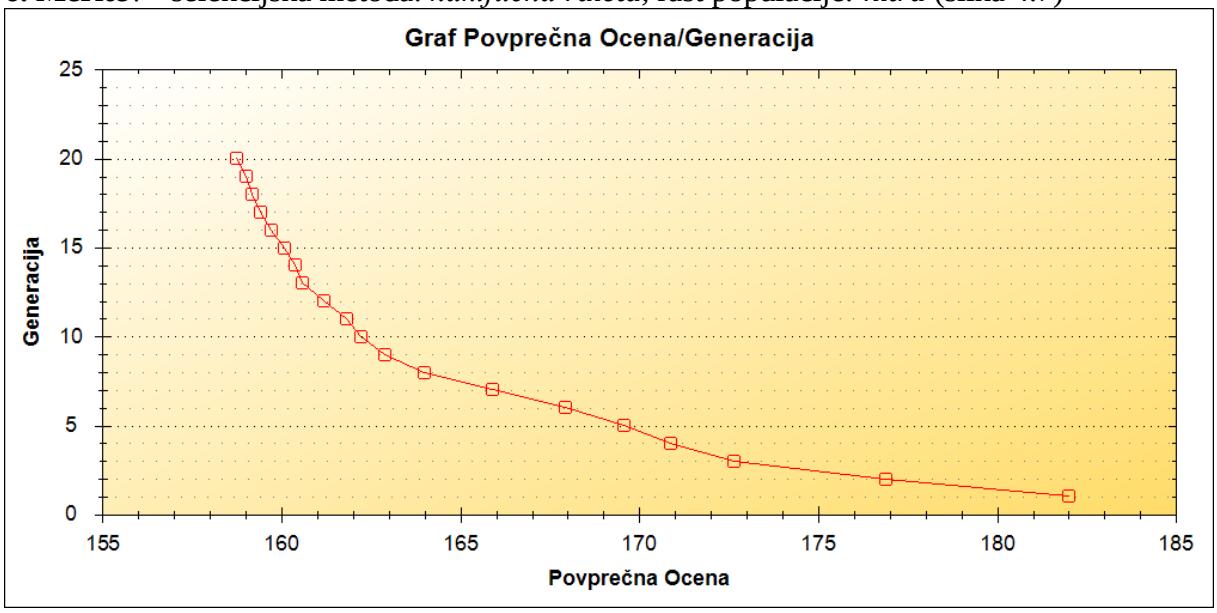
5. Meritev - selekcijska metoda: *naključna ruleta*, rast populacije*: počasna* (slika 4.6)



Slika 5. 6: Graf Pr_1 (naključna ruleta, počasna)

Fitnes ocena rešitve $Oc_{15} = 144{,}60$, velikost končne populacije $s_{20} = 1333$.

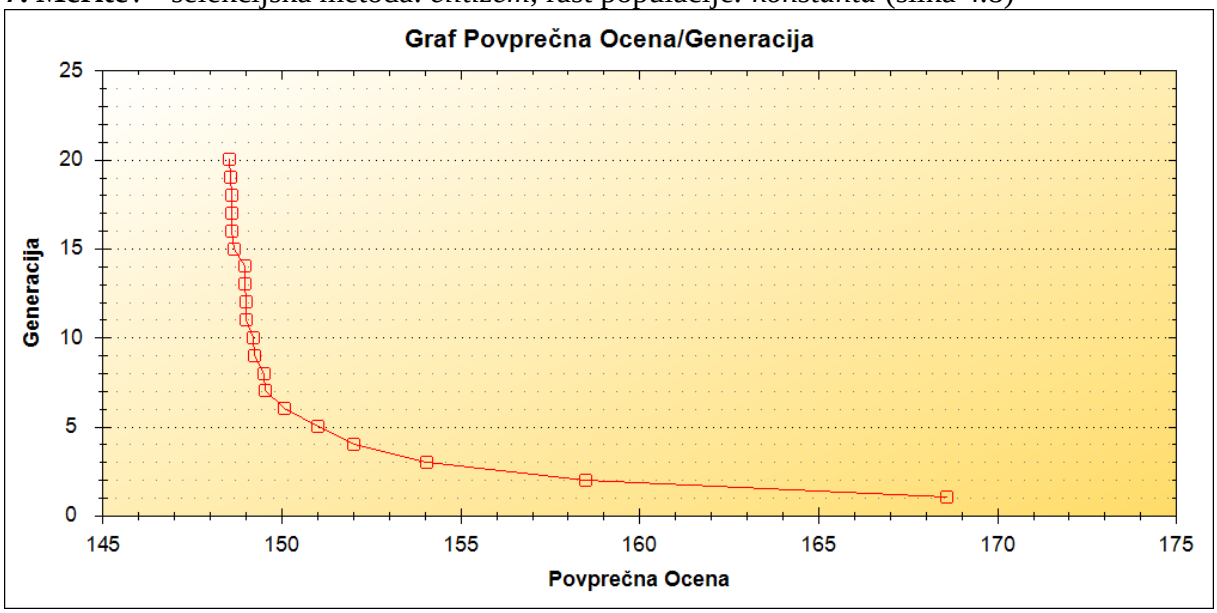
6. Meritev - selekcijska metoda: *naključna ruleta*, rast populacije*: hitra* (slika 4.7)



Slika 5. 7: Graf Pr_1 (naključna ruleta, hitra)

Fitnes ocena rešitve $Oc_{16} = 150{,}25$, velikost končne populacije $s_{20} = 12844$.

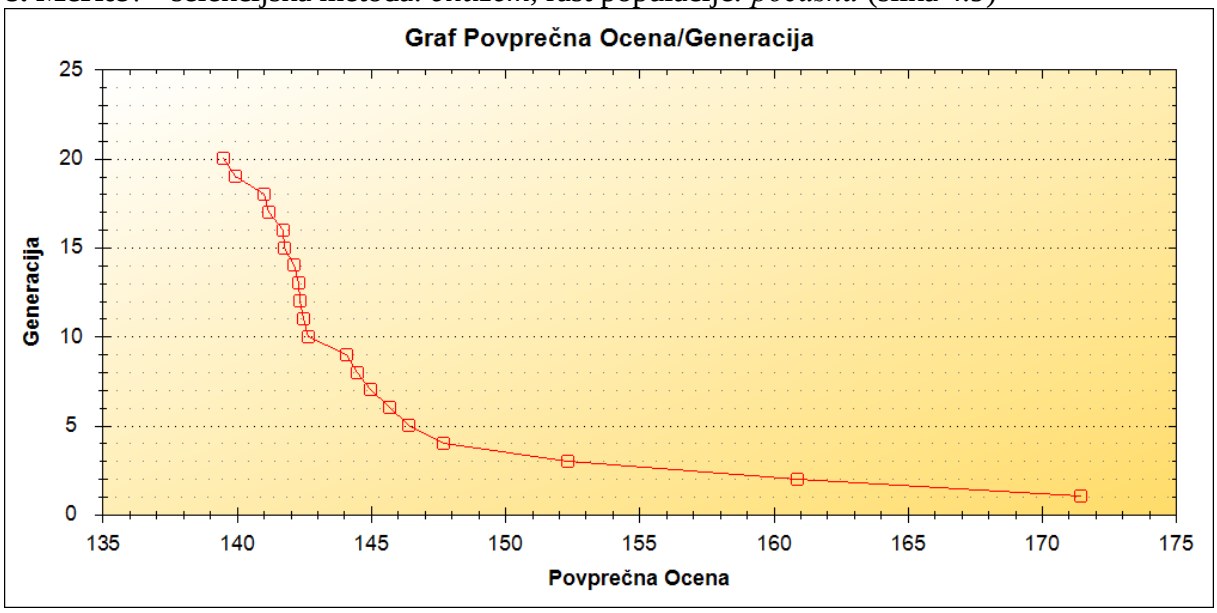
7. Meritev - selekcijska metoda: *elitizem*, rast populacije*: konstanta* (slika 4.8)



Slika 5. 8: Graf Pr_1 (elitizem, konstanta)

Fitnes ocena rešitve $Oc_{17} = 146{,}8$, velikost končne populacije $s_{20} = 200$.

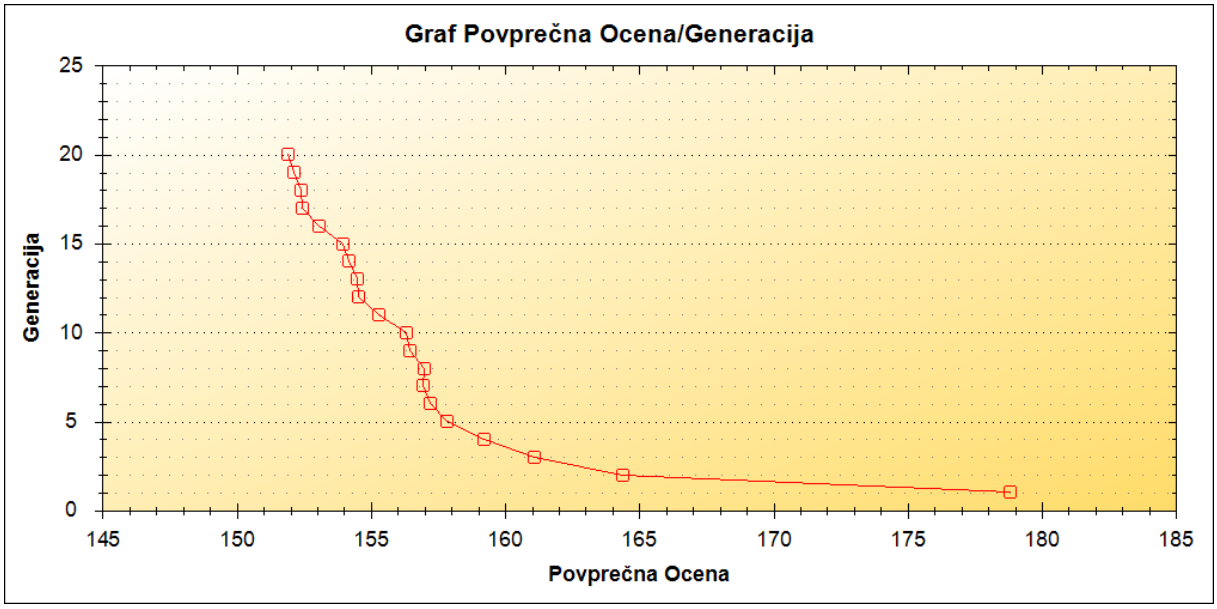
8. Meritev - selekcijska metoda: *elitizem*, rast populacije*: počasna* (slika 4.9)



Slika 5. 9: Graf Pr_1 (elitizem, počasna)

Fitnes ocena rešitve $Oc_{18} = 137{,}35$, velikost končne populacije $s_{20} = 1333$.

9. Meritev - selekcijska metoda: *elitizem*, rast populacije*: hitra* (slika 4.10)

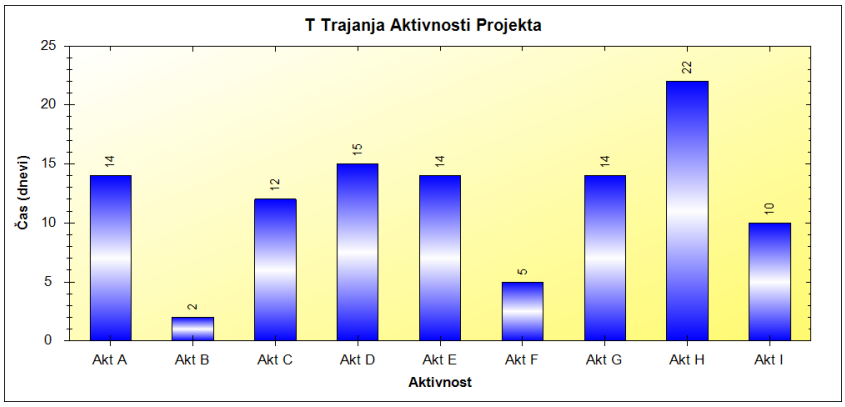


Slika 5. 10: Graf Pr_1 (elitizem, hitra)

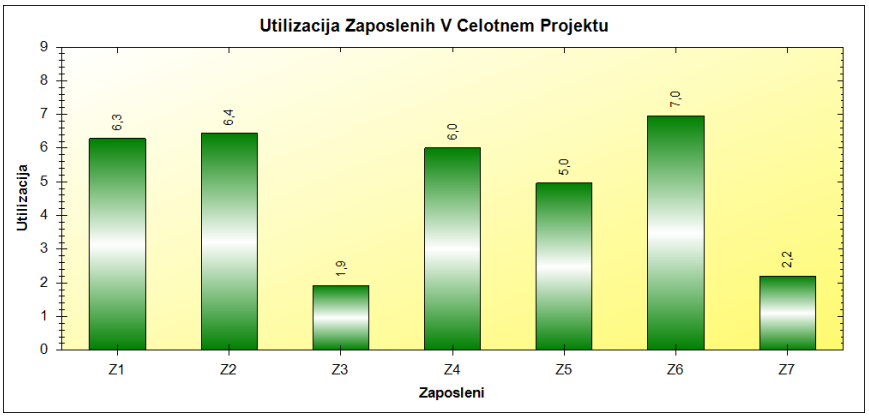
Fitnes ocena rešitve $Oc_{19} = 147{,}10$, velikost končne populacije $s_{20} = 12844$.

5.1.2 Podrobni izpis najboljše rešitve testnega primera

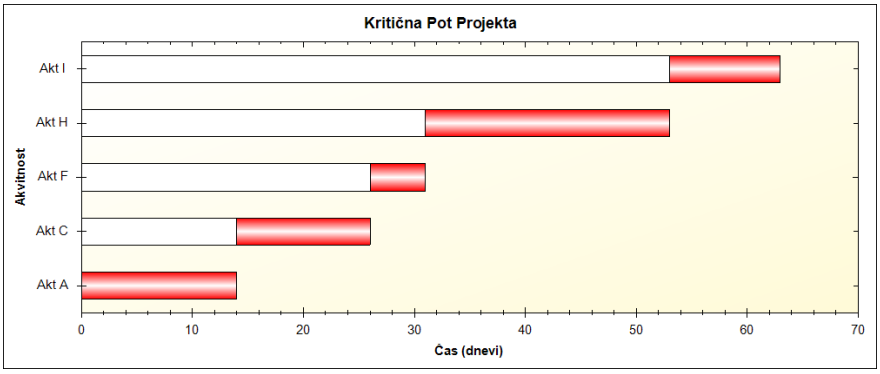
Najboljša rešitev testnega primera Pr_1 je $Oc_{18} = 137{,}35$ (kombinacija selekcije elitizma in počasne rasti populacije). Trajanje kritične poti projekta Pv_1 je $t_{kp} = 63\,dni$, stroški projekta so 17387 € in število dni prerivanja aktivnosti je $t_{pr} = 45\,dni$. Podrobni izpis najboljše rešitve Oc_{18} je podan v slikah 4.11, 4.12, in 4.13.



Slika 5. 11: Prikaz trajanja posameznih aktivnosti pri najboljši rešitvi (Oc_{18})



Slika 5. 12 Utilizacija (seštevek vseh odstotkov časa sodelovanja zaposlenih $Zaposleni_1$ na aktivnostih projekta Pv_1) pri najboljši rešitvi (Oc_{18})



Slika 5. 13: Prikaz trajanja kritične poti pri najboljši rešitvi (Oc_{18})

5.2 Tabela meritev testne množice in ugotovitve

Spodaj je podana tabela meritev za vseh deset testnih primerov množice *Pr*. Za lažje razumevanje tabele Pr_i zapišemo kot $\mathrm{Pr}_1 = (|Pv_1|, |Zaposleni_1|, |Znanje_1|)$.

Mer. m/ Pr_i	1	2	3	4	5	6	7	8	9
$\mathrm{Pr}_1(9,7,6)$	147,80	142,65	144,00	148,30	147,95	150,25	146,80	**137,35**	147,31
$\mathrm{Pr}_2(6,4,2)$	71,34	70,11	72,10	71,90	70,35	75,52	72,10	**67,18**	71,18
$\mathrm{Pr}_3(12,8,9)$	178,48	166,14	171,20	179,12	176,54	183,90	175,40	**163,11**	174,93
$\mathrm{Pr}_4(15,8,6)$	215,30	206,79	205,78	218,18	216,05	229,23	215,34	**197,21**	214,56
$\mathrm{Pr}_5(14,4,9)$	327,82	312,00	319,15	326,49	325,55	332,24	322,75	**302,64**	324,70
$\mathrm{Pr}_6(19,8,5)$	210,90	202,06	211,32	212,27	212,33	213,44	211,42	**201,87**	212,03
$\mathrm{Pr}_7(8,2,4)$	198,43	184,40	189,30	197,41	196,08	201,85	195,12	**181,14**	198,31
$\mathrm{Pr}_8(9,10,6)$	85,56	84,02	87,70	88,39	88,60	90,17	86,32	**82,67**	87,18
$\mathrm{Pr}_9(12,8,7)$	162,10	159,98	163,90	167,91	168,87	174,22	169,50	**157,11**	168,43
$\mathrm{Pr}_{10}(18,7,8)$	217,05	215,20	213,93	230,81	232,45	238,46	230,25	**213,40**	231,23
Povprečje	181,48	174,33	177,83	184,08	183,48	188,93	182,50	**170,37**	182,99

Ugotovitve meritev

Iz tabele meritev razberemo zmagovalno kombinacijo selekcije elitizma in počasne rasti populacije (osmi stolpec tabele). Najslabše se je odrezala kombinacija selekcije naključne rulete in hitre rasti populacije (osmi stolpec tabele).

Za najbolj uspešno od selekcijskih metod se je izkazala selekcija elitizma, kjer se pri križanju uporabi samo dvajset odstotkov najboljše populacije. Starši slednje populacije zato naredijo znatno več otrok, ki imajo v povprečju boljše ocene. Dovolj velik faktor mutacije ($p_m = 23$) selekciji elitizma dodatno pomaga ohranjati genetsko raznolikost populacije. Zato pri reševanju problemov z GAUPP močno priporočam uporabo selekcije elitizma z faktorjem mutacije $p_m > 20$.

Za najbolj učinkovit način rasti populacije se izkaže počasna rast. Pri počasni rasti se populacija vsako generacijo poveča za deset odstotkov prejšnje populacije. To nam omogoča večjo genetsko raznolikost populacije. Slednja lastnost pomaga odvračati algoritem od zgodnje konvergence v lokalne minimume. Pri selekcijskih metodah z počasno rastjo algoritem konvergira tudi po dvajseti generaciji, čeprav v vedno manjših korakih. Če želimo še boljšo rešitev, povečamo število generacij, a se moramo zavedati da časovna zahtevnost algoritma vsako sledečo generacijo močno narašča. Na stopnjo časovne zahtevnosti algoritma najbolj vpliva izbira načina rasti populacije.

Razočaranje meritev so rezultati hitre rasti populacije. GAUPP je v tej kombinaciji dosegal najslabše rešitve (izjema je naključna selekcija). Problem hitre rasti je, da moramo algoritem omejiti na majhno velikost začetne populacije in majhno število generacij. Zato je maksimum števila generacij vseh meritev samo dvajset. Večja kot je začetna populacija, bolj skokovito raste časovna zahtevnost novih generacij. Zato je algoritem smiselno po določenemu število generacij ustaviti, saj napredek povprečne ocene generacije pride ob prevelikem časovnem strošku. Zaganjanje GAUPP ob velikosti začetnih populacij nad 100 in števila generacij nad 20 ne priporočam. Ob manjših začetnih populacijah (pod 30) trpi genetska raznolikost populacije, kar zgodaj upočasni konvergenco rešitve GAUPP.

6 Sklepne ugotovitve

V sklopu diplomske naloge smo spoznali genetski algoritem (GA) kot vrsto evolucijskih algoritmov. GA se primarno uporablja za reševanje problemov, kjer so deterministični algoritmi časovno prezahtevni. Spoznali smo, da GA poteka v več iteracijah (generacijah), v katerih se nad množico dopustnih rešitev, poimenovano populacija kromosomov, izvajajo genetske operacije – koraki GA: selekcija, križanje in mutacija. Ob koncu vsake iteracije GA se populacija kromosomov posodobi. Razložili smo tudi kriterijsko funkcijo in poudarili njen vpliv pri iskanju zadovoljivih rešitev.

Razložili smo osnovne gradnike GA, kromosome in gene, ter ponazorili različne oblike zapisa kromosoma. Pri opisu populacije smo razložili pojem genetske raznolikosti. Definirali smo genetsko raznolikost kot nihanja fitnes ocen kromosomov v populaciji, ter določili dva tipa raznolikosti: močno in šibko. Populacijo močne genetske raznolikosti sestavljajo kromosomi, katerih fitnes ocene so dovolj raznolike, da preprečijo hitro konvergenco GA v lokalne optimume. Za populacijo šibke genetske raznolikost pa je značilen hiter zaključek GA v lokalnih optimumih, ki niso dobre rešitve.

Opazili smo, da izbira selekcijske metode, način križanja in posodabljanja populacije močno vpliva na genetsko raznolikost populacije. Opisali smo štiri selekcijske metode: *naključno ruleto*, *naključno selekcijo*, *selekcijo po prioriteti* in *elitizem*. Spoznali smo se tudi tri vrste križanja kromosomov: *K-pozicijsko*, *uniformno* in *prioritetno uniformno*, ter razložili delovanje in pomen mutacije. Pri koraku posodabljanja populacije smo definirali konstantno ali naraščajočo rast populacije in predstavili štiri metode posodabljanja populacije.

Nadaljnje smo temeljito spoznali problem upravljanja programskih projektov (PUPP). Definirali smo vhodne podatke PUPP: aktivnosti in viri: zaposleni z znanji. Natančno smo definirali množico dopustnih rešitev, ter navedli pogoje za njeno članstvo. V obliki enačb smo zapisali kriterijsko funkcijo PUPP in podrobneje opisali metodo kritične poti. Izračun fitnes ocene kromosoma smo za lažje razumevanje predstavili na demonstracijskem programskem projektu. Omenili smo tudi problem prekrivanja aktivnosti in ugotovili, da ga lahko rešujemo z večanjem uteži prekrivanja aktivnosti in dodajanjem novih kvalificiranih zaposlenih med izvajanjem projekta.

Na osnovi razumevanja PUPP smo izdelali opis realizacije genetskega algoritma za upravljanje programskih projektov (GAUPP) in zapisali njegovo psevdokodo. Pri opisu realizacije GAUPP smo podprli selekcijske metode: *naključno selekcijo*, *naključno ruleto* in *elitizem*. Nadaljnje smo predstavili *K-pozicijsko* križanje in opisali potek mutacije. Definirali smo tri načine rasti populacije: *konstantna*, *počasna* in *hitra*. Pri posodabljanju populacije smo podprli metodo *odstranjevanje najslabših iz populacije*.

Na koncu smo v poglavju meritev generirali množico desetih naključnih programskih projektov in z rezultati meritev pokazali zmagovalno kombinacijo selekcije elitizma in počasne rasti populacije. Za najslabšo se je izkazala kombinacija selekcije naključne rulete in hitre rasti populacije. Iz rezultatov smo razbrali, da se za najboljšo izbiro načina selekcije izkaže elitizem, kjer pri križanju izbiramo med samo dvajset odstotkov najboljših staršev. Za najbolj učinkovit način rasti populacije pa se je izkazala počasna rast. Pri slednji se populacija vsako generacijo poveča za deset odstotkov prejšnje populacije, kar dodatno povečuje genetsko raznolikost populacije. Pri načinu hitre rasti populacije smo izpostavili problem skokovite rasti časovne zahtevnosti vsako novo generacijo. Uporabo slednjega načina rasti pri

reševanju zahtevnejših problemov ne priporočam. Pokazali smo tudi, da dovolj velik faktor mutacije pomaga ohranjati genetsko raznolikost populacij pri vseh selekcijskih metodah in načinih rasti populacije tudi daleč po dvajseti generaciji.

Dodatek: kazalo slik

Slika 2. 1: Graf lokalnih in globalnih minimumov v iskalnem prostoru 6
Slika 2. 2: Preslikava problemske domene iz matematičnega zapisa v genetski zapis 7
Slika 2. 3: Kromosom 7
Slika 2. 4: Primer binarnega zapisa kromosoma 8
Slika 2. 5: Primer desetiškega zapisa kromosoma 8
Slika 2. 6: Primer znakovnega zapisa kromosoma 8
Slika 2. 7: Primer kromosoma zapisanega v obliki permutacijskega koga 8
Slika 2. 8: Populacija kromosomov 9
Slika 2. 9: Koraki izvajanja genetskega algoritma 11
Slika 2. 10: Primer zgodnje konvergence GA ob strogem selekcijskem kriteriju in $p_m = 0$ 12
Slika 2. 11: Primer naključne rulete 13
Slika 2. 12: Primer pozicijskega križanja za *k*=1 14
Slika 2. 13: Primer pozicijskega križanja za *k*=2 14
Slika 2. 14: Primer uniformnega križanja 15
Slika 2. 15: Primer prioritetnega uniformnega križanja 15
Slika 2. 16: Primer mutacije obračanje bita 16

Slika 3. 1: Ganttov diagram programskega projekta *Pv* iz razdelka 3.4 21
Slika 3. 2: Primer grafa *G(V,A)* projekta *Pv* 26
Slika 3. 3: Primer kromosoma *M* projekta *Pv* 27

Slika 4. 1: Prikaz uporabniškega vmesnika aplikacije, ki uporablja GAUPP 29
Slika 4. 2: Primer zaposlenih z vsemi zahtevanimi znanji za aktivnosti 32
Slika 4. 3: Primer normalizacije za tretjega zaposlenega ob $Max_{ef} = 50$ 32
Slika 4. 4: Primer, kjer prva aktivnost ni obdelana 33
Slika 4. 5: Primer eno-pozicijskega križanja starša 1 in 2 36
Slika 4. 6: Primer mutacije otroka 1 36

Slika 5. 1: Graf Pv_1 41
Slika 5. 2 Graf Pr_1 (naključna, konstantna) 42
Slika 5. 3: Graf Pr_1 (naključna, počasna) 42
Slika 5. 4: Graf Pr_1 (naključna, hitra) 43
Slika 5. 5: Graf Pr_1 (naključna ruleta, konstantna) 43
Slika 5. 6: Graf Pr_1 (naključna ruleta, počasna) 44
Slika 5. 7: Graf Pr_1 (naključna ruleta, hitra) 44
Slika 5. 8: Graf Pr_1 (elitizem, konstanta) 45
Slika 5. 9: Graf Pr_1 (elitizem, počasna) 45
Slika 5. 10: Graf Pr_1 (elitizem, hitra) 46
Slika 5. 11: Prikaz trajanja posameznih aktivnosti pri najboljši rešitvi (Oc_{18}) 46
Slika 5. 12 Utilizacija (seštevek vseh odstotkov časa sodelovanja zaposlenih 47
Slika 5. 13: Prikaz trajanja kritične poti pri najboljši rešitvi (Oc_{18}) 47

Literatura

[1] Enrique Alba, J. Francisco Chicano, Software project management with GAs, *Information Sciences* **177** (2007), 2380-2401.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, Frank D. Francone, *Generic Programming: An Introduction*, Morgan Kaufmann 1st edition, 1997.

[3] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, *Discrete Applied Mathematics*, **5**:11–24, 1983.

[4] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, **112**:3–41, 1999.

[5] Lance D. Chambers, *The Practical Handbook of Genetic Algorithms: Applications, Second Edition*, Chapman and Hall/CRC; 2nd edition, 2000.

[6] Janez Demšar, *Projektiranje in organizacija informacijskih sistemov - mrežno načrtovanje*, Ljubljana: fakulteta za računalništvo in informatiko, 2008.

[7] David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional; 1st edition (1989).

[8] Sonke Hartmanna, *A Self-Adapting Genetic Algorithm for Project Scheduling under Resource Constraints*, Naval Research Logistics **49**:433–448, John Wiley&Sons, Inc. (2002).

[9] Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos, *The Longest Path Problem has a Polynomial Solution on Interval Graphs*, Algorithmica - ALGORITHMICA, vol. **57**, no. 2, 2010.

[10] Milena Karova, Julka Petkova, Vassil Smarkov, *A Genetic Algorithm for Project Planning Problem*, Technical University - Varna, Varna, Bulgaria (International Scientific Conference Computer Science 2008).

[11] K.F.Man, K.S.Tang, S.Kwong, *Genetic Algorithms*, Springer-Verlag London Limited, 1999.

[12] Marjan Mernik, Matej Črepinšek, Viljem Žumer, *Evolucijski algoritmi*, Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za računalništvo.

[13] S. N. Sivanandam, S. N. Deepa, *Introduction to Genetic Algorithms*, Springer 1st edition (12 December 2007).

[14] Peter Zidar, *Genetski algoritmi*, Življenje in tehnika: revija za poljudno tehniko, znanost in amaterstvo ISSN: 0514-017X.- Letnik. 59, št. 11 (November 2008).

[15] Predstavitev algoritma metode kritične poti. Dostopno na: http://www.cob.sjsu.edu/davis_r/courses/QBAreader/cpmalgorithm.html

[16] Primer algoritma za metodo kritične poti v jeziku C#. Dostopno na: http://www.codeproject.com/KB/recipes/CriticalPathMethod.aspx

[17] ZedGraph, odprtokodna rešitev za risanje grafov v Visual Studio .NET 2010, jezik C#. Dostopno na: http://zedgraph.sourceforge.net/samples.html