Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Beno Šircelj

Izpolnljivost logičnih funkcij in evolucija

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Gašper Fijavž

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Livnat in soavtorji v članku (Satisfiability and evolution, FOCS, 2014) obravnavajo konvergenco populacije naborov k populaciji modelov logične funkcije v primeru uporabe produktne genetske evolucije s šibko selekcijo. Njihove teoretične rezultate eksperimentalno preverite. Eksperimentalno obravnavajte tudi nekatere družine logičnih funkcij, ki ne zadoščajo pogojem njihovega glavnega rezultata, ravno tako pa uporabite tudi druge metode reprodukcije, ne zgolj produktne variante.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
GA	genetic algorithm	genetski algoritem
DTS	deterministic Turing machine	deterministični Turingov stroj
NTS	nondeterministic Turing machine	nedeterministični Turingov stroj
P	polynomial time complexity class	razred polinomskih problemov
NP	nondeterministic polynomial time complexity class	razred problemov, rešljivih v nedeterminističnem polinomskem času
SAT	boolean satisfiability problem	problem izpolnljivosti

Povzetek

Naslov: Izpolnljivost logičnih funkcij in evolucija

Avtor: Beno Šircelj

V delu smo eksperimentalno ovrednotili rezultate Livnata in soavtorjev [4], ki so pokazali, da populacija naborov slučajne logične funkcije z uporabo šibke selekcije in produktne genetske evolucije konvergira k deležu samih modelov. Preizkuse smo izvajali na družini 28-mestnih logičnih funkcij, v genetskem postopku pa delali s populacijami velikosti 10000. Pri tem smo eksperimentalno potrdili njihove rezultate ter za nekaj dodatnih razredov logičnih funkcij, in ne za zgolj monotone funkcije, pokazali, da genetski postopek k populaciji modelov ravno tako pripelje hitreje kot v splošnem primeru. Eksperimente smo izvedli tudi z uporabo klasičnih reproduktivnih metod genetskih algoritmov, križanja in mutacij. Tudi v tem primeru smo pri slučajno generiranih logičnih funkcijah zaznali konvergenco k populaciji z velikim deležem modelov. Pri nekaterih razredih logičnih funkcij z uporabo križanja in mutacij konvergence k modelom nismo zaznali.

Ključne besede: genetski algoritem, izpolnljivost izjavnih funkcij, evolucija, šibka selekcija.

Abstract

Title: Satisfiability of logical functions and evolution

Author: Beno Šircelj

This thesis contains experimental evaluation of results of Livnat et al. [4], who have shown that, given a logical function, the population of binary vectors converges to a population of function models, under the assumption of weak selection and product reproduction. Our experiments worked with population size 10000 and genotype length 28. We have experimentally confirmed their theoretical results and for a handful of special logical functions shown that the convergence speed towards the population of models matches the convergence speed of monotone logical functions. We have also run the experiments using crossover and mutations reproduction model. In this case we have for a random logical function experimentally detected convergence towards a population with high ratio of models. However, there are some classes of logical function for which we failed to achieve a similar convergence.

Keywords: genetic algorithm, boolean satisfiability problem, evolution, weak selection.

Poglavje 1

Uvod

Za izjavno formulo φ je v splošnem težko poiskati model, nabor logičnih vrednosti spremenljivk, pri katerem je izjavna formula φ resnična. Izjavna formula je v konjuktivni normalni obliki, če je sestavljena iz konjunkcije členov, v katerih so disjunkcije literalov. Znano je, da lahko vsako izjavno formulo spremenimo v enakovredno formulo v konjuktivni normalni obliki. Odločiti, ali je φ, ki je zapisana v konjunktivni normalni obliki, izpolnljiva, je prvobitni NP-polni problem [2]. V razred NP-polnih problemov spadajo vsi problemi, pri katerih lahko rešitev preverimo v polinomskem času. Za najtežje probleme v NP, tako imenovane NP-polne probleme, pa se zdi, da za reševanje ne dopuščajo algoritmov, ki bi rešitev poiskali hitreje kot v eksponentnem času [3].

Kombinatorični problemi takšne (in večje) računske zahtevnosti v praksi zahtevajo pristope, ki ne vključujejo preiskovanja celotnega prostora dopustnih rešitev. Za iskanje njihovih rešitev lahko uporabimo hevristične metode, lokalno optimizacijo, fizikalne metode in druge postopke [1]. Eden od pristopov so genetski algoritmi [6].

Pri genetskih algoritmih opazujemo populacijo potencialnih rešitev danega problema. Algoritem optimalno rešitev išče tako, da posnema evolucijo in iterativno spreminja populacijo čez več generacij. V vsaki iteraciji generacijo spremeni tako, da obstoječe rešitve združi ali jih malo spremeni.

Združitev imenujemo križanje in jo izvedemo tako, da iz dveh ali več osebkov vzamemo dele ter jih uporabimo za izdelavo nove potencialne rešitve. Izvajamo lahko tudi mutacije, ki so majhne spremembe v zapisih osebkov. Za vsak osebek izračunamo njegovo kvaliteto in mu glede na to dodelimo prednost pri križanju v naslednji generaciji. Tipično prvo populacijo ustvarimo naključno. Osebke, ki jih izberemo za uporabo v ustvarjanju naslednje generacije, določimo naključno, vendar damo kvalitetnejšim osebkom prednost. V našem primeru genetske algoritme uporabljamo za iskanje modelov izjavne funkcije.

Za razliko od tipične uporabe genetskih algoritmov ne iščemo optimalne rešitve, ampak spremljamo, kaj se dogaja s celotno populacijo. Modelom damo le majhno evolucijsko prednost pred ne-modeli. Zanima nas, kaj se dogaja s populacijo pri uporabi majhne evolucijske prednosti. Spremljamo potek evolucije za različne izjavne funkcije in iščemo strukture funkcij, pri katerih dosežemo konvergenco k populaciji samih modelov hitreje kot za tipično funkcijo in tudi funkcije, pri katerih do konvergence k samim modelom ne pride.

Poglavje 2

Izjavne funkcije

Izjavna funkcija n spremenljivk je preslikava iz $\{0,1\}^n$ v $\{0,1\}$, kjer je $n \in \mathbb{N}$. Vrednosti 0 in 1 predstavljata logični vrednosti, pri tem 0 označuje neresnico in 1 resnico. Vsako izjavno funkcijo je mogoče predstaviti z resničnostno tabelo, iz katere lahko za vsako n-terico $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ razberemo vrednost $\varphi(\mathbf{x})$. Za nabor spremenljivk $\mathbf{x}$, pri katerem velja $\varphi(\mathbf{x}) = 1$, pravimo, da *izpolni* izjavno funkcijo φ. Naboru, ki dano izjavno funkcijo izpolni, pravimo tudi *model*; če je ne izpolni, mu pravimo *ne-model*. Funkcija φ, opisana s tabelo 2.1, ima natanko štiri modele. Eden izmed modelov je $(0,0,1)$.

x_1	x_2	x_3	$\varphi(\mathbf{x})$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Tabela 2.1: Resničnostna tabela.

Izjavno funkcijo lahko predstavimo tudi z izjavnim izrazom. Za izjavne izraze poznamo dve normalni obliki, konjuktivno in disjuktivno. Če opišemo izjavni izraz, ki se nahaja v tabeli 2.1 tako, da povemo, da je resničen, če se nahajamo v eni od vrstic 1, 2, 4 ali 8, dobimo disjuktivno normalno obliko. Vsako vrstico lahko opišemo z osnovno konjunkcijo, ki je konjunkcija literalov in je lahko sestavljena iz enega samega člena. Druga vrstica je opisana s stavkom: x_1 je neresničen in x_2 je neresničen in x_3 je resničen, kar lahko predstavimo z $(\neg x_1 \wedge \neg x_2 \wedge x_3)$. Vrstice 1, 2, 4 in 8 združimo z disjunkcijami:

$$(\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$$

Če pa opišemo tabelo ravno obratno in povemo, da se ne nahajamo v vrsticah, ki ne izpolnijo izjavnega izraza, dobimo konjuktivno normalno obliko. V našem primeru so to vrstice 3, 5, 6 in 7. Konjuktivna normalna oblika je sestavljena iz več osnovnih disjunkcij, ki so združene z konjunkcijami:

$$(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$$

2.1 Monotone funkcije

Izjavna funkcija φ je *monotona*, če za vsak par naborov $\mathbf{a} = (a_1, \ldots, a_n)$ in $\mathbf{b} = (b_1, \ldots, b_n)$ velja implikacija: če je $\mathbf{a} \leq \mathbf{b}$, potem je $\varphi(\mathbf{a}) \leq \varphi(\mathbf{b})$. Pri tem za logični vrednosti privzamemo $0 < 1$, nabore pa primerjamo produktno: $\mathbf{a} \leq \mathbf{b}$ natanko tedaj, ko za vsak indeks i velja $a_i \leq b_i$.

x_1	x_2	$\varphi(\mathbf{x})$
0	0	0
0	1	1
1	0	0
1	1	1

Tabela 2.2: Monotona funkcija φ.

Nabor $\mathbf{x}$ lahko zavzame naslednje vrednosti: $\mathbf{x} \in \{00, 01, 10, 11\}$. V tabeli 2.3 primerjamo vsak nabor z ostalimi.

$$
\begin{array}{lll}
00 < 01 & \text{in} & \varphi(00) = \mathbf{0} \leq \varphi(01) = \mathbf{1} \\
00 < 10 & \text{in} & \varphi(00) = \mathbf{0} \leq \varphi(10) = \mathbf{0} \\
00 < 11 & \text{in} & \varphi(00) = \mathbf{0} \leq \varphi(11) = \mathbf{1} \\
01 < 11 & \text{in} & \varphi(01) = \mathbf{1} \leq \varphi(11) = \mathbf{1} \\
10 < 11 & \text{in} & \varphi(10) = \mathbf{0} \leq \varphi(11) = \mathbf{1}
\end{array}
$$

Tabela 2.3: Primerjave naborov.

Nabora 01 in 10 nista primerljiva. Če ju primerjamo po prvi koordinati, je drugi nabor večji, saj je $0 < 1$. Če jih primerjamo po drugi koordinati, je pa prvi nabor večji, saj je $1 > 0$. Vidimo, da imajo vsi nabori, ki so večji od drugih, tudi vrednosti, ki so večje ali enake od drugih, tako da je funkcija φ monotona.

V tabeli 2.4 imamo primer funkcije ψ, ki ni monotona. Vidimo, da za nabora 00 in 10 velja $00 < 10$, za njuni vrednosti pa velja $\psi(00) = 1 > \psi(10) = 0$.

x_1	x_2	$\psi(\mathbf{x})$
0	0	1
0	1	1
1	0	0
1	1	1

Tabela 2.4: Funkcija ψ ni monotona.

2.2 Parnostne funkcije

Parnostna funkcija je vsaka funkcija, pri kateri je logična vrednost odvisna zgolj od parnosti števila resničnih oz. neresničnih vhodov. Zgled je ekskluzivna disjunkcija, ki je resnična natanko tedaj, ko je liho mnogo izjavnih spremenljivk $x_1, x_2, \ldots, x_n$ resničnih.

$$\varphi(\mathbf{x}) = x_1 \veebar x_2 \veebar \cdots \veebar x_n$$

.

x_1	x_2	$\varphi(\mathbf{x})$
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 2.5: Ekskluzivna disjunkcija je parnostna.

2.3 Pragovne funkcije

Izjavna funkcija φ je *pragovna*, ko obstajata naravni števili a in b, pri katerih je $f(x_1, x_2, \ldots, x_n)$ resnična natanko tedaj, ko je $\sum_i x_i \leq a$ ali $\sum_i x_i \geq b$, torej, ko je število enic v vhodnem vektorju bodisi dovolj majhno bodisi dovolj veliko. Negaciji pragovne funkcije rečemo *ko-pragovna* funkcija. Taka funkcija je resnična tedaj, ko je število enic večje od a in manjše od b. V tabeli 2.6 je primer pragovne funkcije φ. Ta funkcija je resnična natanko tedaj, ko je število enic v naboru največ 1 ali vsaj 3.

x_1	x_2	x_3	$\varphi(\mathbf{x})$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tabela 2.6: Pragovna funkcija φ pri $a = 1$ in $b = 3$.

Poglavje 3

Težki problemi

3.1 Razred P

V razredu P (*ang. polynomial time*) je družina odločitvenih problemov, za katere velja, da jih je mogoče rešiti v polinomskem času. Polinomski čas pomeni, da je število enotskih operacij $O(n^k)$ za nek nenegativen k, če je velikost vhoda reda n. Urejanje tabele števil je, kot primer, izvedljivo v polinomskem času.

3.2 Razred NP

V razredu NP (*ang. nondeterministic polynomial time*) [2] so odločitveni problemi, za katere velja, da jih je mogoče rešiti na *nedeterminističnem Turingovem stroju* v polinomskem času. Poleg te definicije obstaja še definicija s preverjanjem rešitve. Problem spada v razred NP, če obstaja tak algoritem, ki preveri rešitev problema v polinomskem času. Primer problema, ki je v NP, je iskanje Hamiltonove poti. Taka pot v grafu obišče vsako vozlišče natanko enkrat. Algoritem, ki odloči, ali graf ima Hamiltonovo pot ali ne, bi v najslabšem primeru moral preiskati vsa možna zaporedja točk grafa, da to ugotovi. Če pa imamo podano rešitev, lahko v polinomskem času preverimo, ali je resnična — pregledamo samo, ali so obiskana vsa vozlišča in to samo

enkrat.

Razred NP vsebuje podrazred NP-polnih problemov. Problem je NP-poln, če je nanj mogoče v polinomskem času prevesti katerega koli izmed problemov, ki so v razredu NP.

Omenimo tudi razred NP-težek, ki ga NP ne vsebuje. V tem razredu so težji problemi kot v NP. Primer problema, ki spada med NP-težke probleme, je iskanje najkrajše Hamiltonove poti v grafu. Iskanje najkrajše Hamiltonove poti je težje kot odločanje, ali taka pot obstaja. Tudi če imamo podano najkrajšo takšno pot, ne moremo enostavno preveriti, ali res ne obstaja še krajša.

3.3 Izpolnljivost izjavnih funkcij

Eden izmed NP-polnih problemov je problem izpolnljivosti izjavnih funkcij ali SAT (*ang. Boolean Satisfiability problem*). Naloga je ugotoviti, ali lahko za neko izjavno funkcijo, zapisano v konjuktivni normalni obliki, najdemo tak nabor spremenljivk, da bo funkcija vrnila vrednost 1. Za funkcijo, ki tak nabor ima, rečemo, da je *izpolnljiva*. Problem k-SAT je podrazred problema izpolnljivosti SAT, pri katerem se omejimo zgolj na tiste formule v konjuktivni normalni obliki, ki imajo v vsakem členu kvečjemu k literalov.

Impagliazzo in Paturi [3] sta podala domnevo o težavnosti problema izpolnljivosti. Trdita, da za noben problem k-SAT, pri katerem je $k \geq 3$, ne obstaja algoritem, ki bi v najslabšem primeru našel rešitev hitreje kot v eksponentnem času.

3.4 Pristopi k reševanju

Reševanja NP problemov se lahko lotimo na več načinov. Eden od načinov je izčrpno preiskovanje. Tukaj preverimo vsako možno rešitev; za to bi potrebovali eksponentno mnogo časa v odvisnosti od velikosti vhodnih podatkov. Tak način reševanja nam vzame preveč časa, da bi ga lahko uporabljali v pra-

ksi. Zato raje uporabimo postopke, ki nam sicer ne zagotavljajo optimalne rešitve, tečejo pa hitreje kot v eksponentnem času.

3.4.1 Metahevristike

Metahevristike so strategije iskanja rešitve za optimizacijske probleme, ki so težji od problemov v razredu P. Cilj je preiskovanje prostora rešitev in iskanja rešitve, za katero ni nujno, da je optimalna. Tako rešitev najdejo hitreje kot v eksponentnem času. Primeri metahevristik so lokalno iskanje, simulirano ohlajanje, tabu iskanje in genetski algoritmi.

- **Lokalno iskanje**: Pri lokalnem iskanju se premikamo od ene do druge dopustne rešitve z uporabo lokalnih sprememb. V vsakem koraku pogledamo vsa sosednja stanja in se premaknemo v neko stanje, če ima le-to boljšo oceno. Na primer, to storimo z algoritmom: če iščemo, ali je neka logična funkcija v konjuktivni normalni obliki izpolnjena, bomo šteli število členov, ki so izpolnjeni glede na vrednost spremenljivk. V vsakem koraku bo algoritem spremenil le eno spremenljivko. S takim postopkom se bomo približevali vedno večjemu številu izpolnjenih členov in vedno večji verjetnosti, da bomo dobili nabor spremenljivk, ki izpolnijo funkcijo.

- **Simulirano ohlajanje**: Simulirano ohlajanje je izboljšava lokalnega iskanja, ki se od njega razlikuje v tem, da se proti boljšemu stanju premaknemo le z neko verjetnostjo. Dalj časa kot se algoritem izvaja, večjo prednost daje boljšemu stanju. Ta algoritem se za razliko od lokalnega iskanja lahko izogne lokalnim optimumom.

- **Tabu iskanje**: Tudi tabu iskanje je izboljšava lokalnega iskanja. Pri tem načinu imamo tabu seznam, na katerem so nedavno obiskana stanja, v katera se ne smemo vračati. Omogoča nam, da se izognemo stanjem, v katerih ima več sosednjih stanj enako vrednost. S tem načinom iskanja preidemo v slabša stanja, če smo vsa boljša že raziskali. Ker ne

moremo ostati na majhni množici stanj več časa, se izognemo lokalnim optimumom.

Poglavje 4

Genetski algoritmi

Genetski algoritmi (GA) črpajo navdih iz narave in delujejo podobno kot evolucija. Pri naravni selekciji se dobre lastnosti osebkov prenašajo v nove generacije, saj imajo taki osebki večjo verjetnost preživetja in razmnoževanja. Pri GA ustvarimo populacijo dopustnih rešitev problema in nad temi rešitvami simuliramo naravno selekcijo tako, da čez več generacij iz staršev ustvarjamo potomce. Osebke, ki jih bomo izbrali za ustvarjanje nove generacije, izberemo glede na njihove lastnosti. Dopuščamo tudi možnost mutacij. Ko izvajamo tak algoritem, ne moremo točno vedeti, ali smo dobili najboljšo rešitev, zato moramo imeti nek ustavitveni pogoj. Ta pogoj, ki ga moramo določiti vnaprej, je lahko kvaliteta najboljšega osebka ali pa čas izvajanja. Za izvajanje GA potrebujemo naslednje:

- Predstavitev osebkov: Osebek ali element populacije predstavimo z naborom, zaporedjem ničel in enic dolžine n, ki mu pravimo tudi *kromosom*. Kromosom je sestavljen iz več genov. V našem primeru ima vsak gen dva alela (različica gena), ki ju označimo z 0 in 1. Dopuščamo, da ima več osebkov populacije isti kromosom, tako da je populacija dejansko več-množica kromosomov. V tabeli 4.1 je prikazana populacija s štirimi osebki, ki se nahajajo vsak v svoji vrstici. Vsak od njih ima osem genov.

- Generiranje začetne populacije: Tipično je ustvarjena naključno. Možne

osebke izberemo iz vseh možnih. Število osebkov v populaciji bomo označili z N.

0	0	0	1	0	0	1	0
1	1	0	1	1	0	0	0
1	0	1	1	0	0	1	1
1	0	1	1	0	0	1	1

Tabela 4.1: Predstavitev osebkov.

- *Kriterijska funkcija*: Funkcija, ki nam pove kakovost osebka. Za vrednosti spremenljivk lahko izračunamo, ali je funkcija izpolnjena ali ne. Za kriterij imamo samo ta dva pogoja. V našem primeru uporabljamo šibko selekcijo, modeli imajo samo majhno evolucijsko prednost pred ne-modeli. Sledimo [4] in modele ocenimo s kriterijsko vrednostjo $1+\epsilon$, ne-modele z 1, pri čemer je $\epsilon = 1/(2 \cdot n)$.

- Križanje: Iz staršev ustvarimo potomca, ki je kombinacija genov staršev. Starše izbiramo glede na kriterijsko funkcijo.

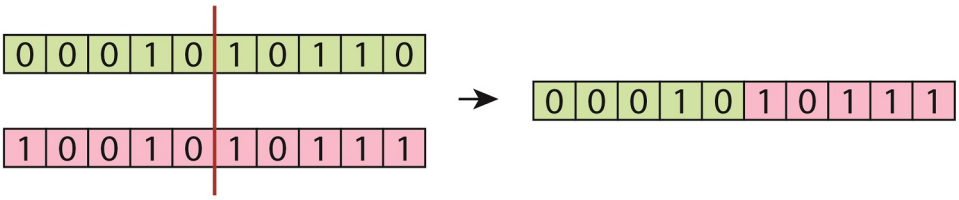
- Mutacije: So naključne spremembe v genih kromosoma. Mutacije se izvajajo v manjši meri kot križanje, običajno mutiramo okoli 1% osebkov [5]. GA lahko implementiramo tudi brez mutacij, a jih ponavadi uporabljamo, da se izognemo lokalnim optimumom. Mutacija, ki jo uporabljamo v našem primeru, spremeni natanko en gen v kromosomu osebka.

4.1 Križanje

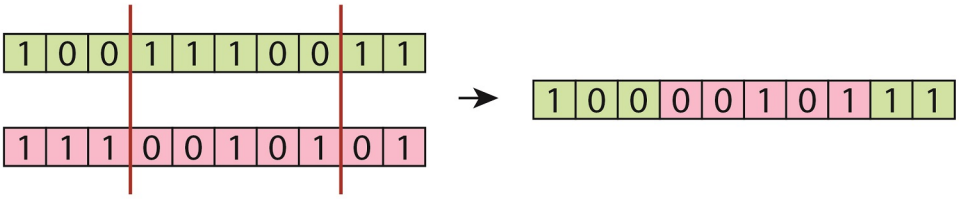
Pri križanju najprej izberemo osebke, iz katerih bomo ustvarili potomca. To lahko naredimo na več načinov. En način je selekcija z ruleto. Verjetnost, da bo posameznik izbran, je usklajena z njegovo kriterijsko oceno. Drugi način je selekcija glede na rang. V tem primeru uredimo osebke glede na

kriterijsko oceno in jih izberemo sorazmerno z njihovim položajem. V praksi se za uspešno metodo izkaže tudi turnirska selekcija, kjer naključno izberemo nekaj osebkov in za križanje izberemo najboljšega.

Po izboru parov nad vsakim parom izvedemo križanje. Pri vsakem osebku izberemo, katere dele kromosoma bomo uporabili pri ustvarjanju potomca. Lahko izberemo eno prekinitev v kromosomu in od vsakega starša vzamemo del le-tega. Primer je na sliki 4.1. Križanje lahko izvedemo tudi na več točkah. To je prikazano na sliki 4.2.



Slika 4.1: Križanje na eni točki.



Slika 4.2: Križanje na večih točkah.

Za naše potrebe smo uporabili križanje z uporabo dveh staršev. Za njihov izbor smo uporabili selekcijo z ruleto. Križanje smo izvedli v eni sami slučajno izbrani točki. Algoritem 4.1 je psevdo koda našega postopka križanja. Križanje smo vedno izvajali tudi z mutacijami.

4.2 Produktna reprodukcija

Poleg križanja smo za ustvarjanje nove generacije uporabili tudi produktno reprodukcijo. Ta način so uporabili Livnat in soavtorji [4]. Postopek repro-

Algoritem 4.1 Križanje

```
1: function KRIŽANJE(populacija)
2:     modeli ← vzemi modele iz populacije
3:     ne-modeli ← vzemi ne-modele iz populacije
4:     štModelov ← preštej modele
5:     štNe-Modelov ← preštej ne-modele
6:     verjetnostIzbire ← razmerje med kriterijsko oceno modelov in ne-
   modelov
7:     for osebek od števila mutantov do zadnjega osebka do
8:         for osebek je osebek1 do osebek2 do
9:             skupina ← izberi skupino z verjetnostjo verjetnostIzbire
10:            if skupina je modeli then
11:                osebek ← izberi naključnega iz modelov
12:            else
13:                osebek ← izberi naključnega iz ne-modelov
14:            end if
15:        end for
16:    end for
17:    presek ← izberi naključno mesto v kromosomu
18:    osebek ← vzemi del osebka1 do preseka in vzemi del osebka2 od pre-
   seka do konca
19: return populacija
20: end function
```

dukcije poteka tako, da za vsako generacijo t ustvarimo populacijo velikosti N. Med celotnim potek evolucije se N ne spreminja. Najprej izračunamo frekvence ν^t alelov prejšnje generacije za vsak i-ti gen $y^{(1)}, \ldots, y^{(N)}$.

$$\nu_i^t = \frac{1}{N} \sum_{j=1}^{N} y_i^{(j)} \tag{4.1}$$

Nato uporabimo še kriterijsko oceno $f(x)$ vsakega osebka x, da dobimo pričakovano vrednost vsakega i-tega gena za naslednjo generacijo:

$$\mu_i^{t+1} = \frac{E_{\nu^t}[f(x) \cdot x_i]}{E_{\nu^t}[f]} \tag{4.2}$$

Ko imamo te frekvence, določimo gene naslednje generacije z Bernoullijevo porazdelitvijo glede na izračunane verjetnosti μ_i. Taka porazdelitev nam da vrednost 1 z verjetnostjo p in 0 z verjetnostjo $1-p$. Če je nek gen v populaciji enak pri vseh osebkih, bo tudi v naslednjih generacijah ohranjal isto vrednost brez možnosti spremembe. Pravimo, da je takšen gen fiksen.

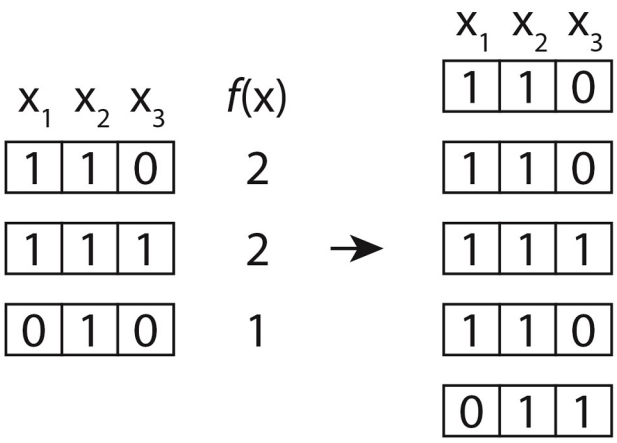
Na sliki 4.3 imamo primer produktne reprodukcije. Iz populacije treh osebkov bomo naredili populacijo petih osebkov. Velikost populacije se načeloma ne spreminja, a jo zaradi lažjega prikaza v našem primeru bomo. Vrednost za ϵ, ki jo bomo uporabili je 1; modeli torej imajo vrednost $1 + \epsilon$, kar je 2. Ne-modeli imajo vrednost 1. Na sliki 4.3 sta prva dva osebka modela in imata kriterijsko oceno 2. Seštevek vseh kiterijskih ocen je 5. Če izračunamo verjetnost μ_1, ki nam pove, s kakšno verjetnostjo bo x_1 enak 1, dobimo:

$$\mu_1 = \frac{f(x^{(1)}) \cdot x_1^{(1)} + f(x^{(2)}) \cdot x_1^{(2)} + f(x^{(3)}) \cdot x_1^{(3)}}{5} = \frac{2 \cdot 1 + 2 \cdot 1 + 1 \cdot 0}{5} = 0.8$$

Pričakovano število enic na prvem genu je 4 (od 5 osebkov). Verjetnost enice na drugem mestu je 1, torej bo ta gen vedno ostal enak. Verjetnost enice na tretjem mestu je 0,4 — torej je pričakovano število enic 2.

Algoritem 4.2 Produktna reprodukcija.

```
1: function REKOMBINACIJA(populacija)
2:     sestevekOceneGenov ← nov niz dolzine stGenov
3:     for all osebek do
4:         for  gen od 1 do stGenov do
5:             if osebek(gen) je 1 then
6:                 sestevekOceneGenov(gen) pristej oceno od osebka
7:             end if
8:             sestevekVsehOcen pristej oceno od osebka
9:         end for
10:    end for
11:    for gen od 1 do stGenov do
12:        pričakovanaVrednost(gen) ← sestevekOceneGenov(gen) deli z se-
   stevekVsehOcen
13:    end for
14:    for all osebek do
15:        for all gen v osebek do
16:            gen ← Izberi 1 z verjetnostjo pričakovanaVrednot(gen)
17:        end for
18:    end for
19: return populacija
20: end function
```



Slika 4.3: Produktna reprodukcija s kriterijsko funkcijo f, ki iz populacije treh osebkov naredi populacijo s petimi osebki.

Poglavje 5

Evolucija populacije

Livnat in soavtorji [4] so genetski pristop k reševanju kombinatoričnih problemov usmerili v drugačno smer. Namesto iskanja optimalne rešitve, ki jo genetski postopek poskuša poiskati tako, da skozi več zaporednih generacij sledi trenutno najboljši rešitvi, opazujejo kakovost celotne populacije.

Če predpostavimo, da imajo v vsaki generaciji modeli izjavnega izraza φ opazno evolucijsko prednost pred nabori, ki istega izraza ne izpolnijo, domnevamo, da bo delež modelov v populaciji naraščal. V članku predlagajo kromosom, ki ima dva alela (različici gena) za vsak gen, tako kot imajo izjavne funkcije dve vrednosti za vsako spremenljivko. Dva alela tukaj ne predstavljata večjih omejitev, saj si lahko alel 0 razlagamo kot kateri koli drugi alel, ki ni alel 1. V članku uporabljajo tudi dve predpostavki. Prva je, da je kriterijska ocena kromosoma 1 ali $1 + \epsilon$, kjer je $\epsilon > 0$ zelo majhna vrednost. Druga predpostavka pa je, da za razmnoževanje namesto križanja in mutacij uporabljamo produktno reprodukcijo, opisano v razdelku 4.2.

Začetna porazdelitev alelov je označena z μ^0. Verjetnost, da ima kromosom v generaciji t kriterijsko oceno $1+\epsilon$, je označena z $\mu^t(f) = Pr_{\mu^t}[f(x) = 1+\epsilon]$. Trdijo, da se $\mu^t(f)$ približa 1 po $O(\frac{n}{\epsilon\mu^0(f)})$:

Izrek 1 *Če je f monotona, potem $\mu^t(f) \geq 1 - \frac{n(1+\epsilon)}{\epsilon t\mu^0(f)}$.*

Pokazali so, da za neskončno populacijo obstajajo funkcije, pri katerih z uporabo produktne reprodukcije ne dosežemo konvergence. Ena od ta-

kih funkcij je parnostna funkcija. Denimo, da imamo neskončno populacijo π^0 vseh možnih genotipov za parnostno funkcijo z n geni. Vsak od njih je zastopan z enako verjetnostjo $\frac{1}{2^n}$. Slučajno izbran osebek populacije π bo z verjetnostjo 1/2 model, saj je med 2^n možnimi nabori natanko 2^{n-1} tistih, ki imajo liho število enic in so torej modeli za ekskluzivno disjunkcijo. Ravno tako je za vsako mesto v kromosomu verjetnost, da ima model na njem zapisano enico, enaka 1/2. S produktno reprodukcijo bomo v naslednji generaciji π^1 torej dobili identično porazdelitev alelov. Do konvergence k večjemu deležu modelov ne pride, saj imajo vse generacije, to pokažemo z indukcijo, isto porazdelitev osebkov kot začetna populacija π^0.

Livnat in soavtorji [4] pa so pokazali naslednji izrek:

Izrek 2 *Naj bo* $\beta = \sqrt{\frac{\epsilon}{N(1-n\epsilon)}}$. *Če je* $\widetilde{f}(\mu^0) > 1 + \sqrt{2\beta \ln \frac{2}{\beta}}$ *in* $1/N^{1/3} < \epsilon < 1/n$, *potem obstaja taka konstanta* C, *da za vsak* $T \geq C \cdot \frac{\epsilon n^8 \cdot N^4}{1-n\epsilon}$ *velja:*

$$Pr[\widetilde{f}(\mu^{(T)}) = 1 + \epsilon] \geq 1 - 2\beta - 2/n$$

Pri tem je $\widetilde{f}(\mu^0)$ povprečna vrednost kriterijske funkcije na začetni populaciji μ_0. Vemo, da je $\widetilde{f}(\mu^0) \leq 1 + \epsilon$. Ker imajo modeli vrednost kriterijske funkcije enako $1+\epsilon$, ne-modeli pa 1, je povprečna vrednost kriterijske funkcije na populaciji enaka $1+\epsilon$ natanko tedaj, ko imamo v populaciji same modele. Osebek se bo razmnoževal z majhno prednostjo, če bo predstavljal model. Ker je ϵ majhen, imamo opravka s šibko selekcijo. Konvergenco dosežemo tudi, če imamo velik koeficient, a v članku avtorji trdijo, da deluje najbolje pri majhnih vrednostih ϵ in velikemu N. Ena od razlag, zakaj to drži, pravi, da selekcija deluje kot gradientni spust. Če je koeficient prevelik in imamo velike korake pri spustu, lahko preskočimo iskane točke. Pomemben je tudi začetni delež modelov v populaciji, ki mora biti večji kot $N^{-1/4}$. Iz izrekov 1 in 2 razberemo tudi to, da je konvergenca pri monotonih funkcijah hitrejša.

5.1 Izbira logične funkcije

Kako izbrati slučajno logično funkcijo z denimo 40 spremenljivkami? Takšnih logičnih funkcij je kar $2^{2^{40}}$ in ena izmed možnosti za slučajno konstrukcijo je, da za vsakega izmed 2^{40} naborov slučajno (z verjetnostjo $1/2$) in neodvisno od drugih izberemo ali je model ali ne. Število modelov tako generirane slučajne funkcije je porazdeljeno binomsko s pričakovano vrednostjo 2^{39} modelov (natanko polovica vseh možnih) in standardnim odklonom 2^{19}. Normalna aproksimacija pri izbiri odklona $5\sigma = 5 \cdot 2^{19}$ trdi, da število modelov tako zgrajene logične funkcije pripada intervalu $[2^{39} - 5 \cdot 2^{19}, 2^{39} + 5 \cdot 2^{19}]$ z verjetnostjo *vsaj* $1 - 6 \cdot 10^{-6}$. Trdimo lahko, da je pri slučajni izbiri logičnih vrednosti naborov število modelov zelo koncentrirano okoli pričakovanega števila modelov.

Denimo, da logično funkcijo konstruiramo na drugačen način. Fiksirajmo število modelov na npr. 5000 in nato slučajno izberemo 5000 naborov, ki so modeli. Kakšno je pričakovano število modelov, ki se pojavijo kumulativno v vseh generacijah genetskega algoritma? Zaradi linearnosti matematičnega upanja je to število enako produktu števila generacij, velikosti populacije in verjetnosti, da je posamezen nabor model. Pri velikosti populacije 10000, številu generacij 5000, dolžini kromosoma 40 je to število $5000 \cdot 10000 \cdot \frac{5000}{2^{40}}$, kar je manj od 0,23. Pod črto lahko sklenemo, da je potrebno obravnavati zgolj takšne logične funkcije, ki imajo znaten delež modelov. Tiste, ki imajo premajhno (ali preveliko) število modelov, so zelo daleč od slučajnih, ravno tako pa na njih ne moremo pričakovati bistvenih evolucijskih rezultatov.

Poglavje 6

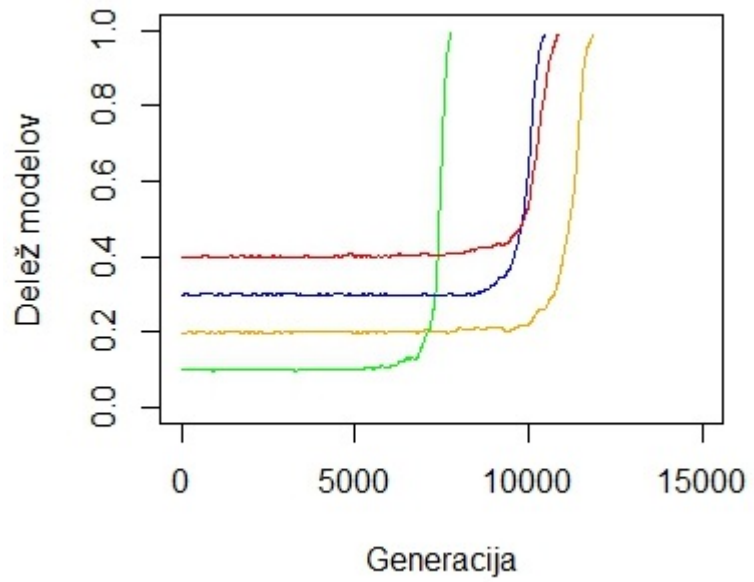
Preizkusi

Preizkuse smo izvajali na prenosnem računalniku ASUS N751JX z jedrom Intel i7 i7-4720HQ, ki deluje na frekvenci 2,6 GHz in ima 6 MB predpomnilnika. Računalnik ima 8 GB DDR3L pomnilnika, ki deluje na frekvenci 1600 MHz. Koda je napisana v programskem jeziku C++ z uporabo okolja Visual Studio 2015 na operacijskem sistemu Windows 7.
Preizkuse, razen v primerih, ki so posebej navedeni, smo izvajali pri dolžini kromosoma $n = 28$ in velikosti populacije $N = 10000$. Pri večjih vrednostih smo presegli dovoljeno mejo dodelitve pomnilnika za posamezne procese, ki je v našem primeru 2 GB. Najdaljši še uspešen preizkus smo izvajali 43 minut, šlo je pa za parnostno funkcijo. Tipično obnašanje funkcij prikazujemo z grafi. Na navpični osi je prikazan delež modelov, na vodoravni pa število generacij od začetka izvajanja. Za lepši prikaz podatkov so vsi grafi zglajeni s kubičnim glajenjem z zlepki (*ang. cubic smoothing spline*)[7]. Večino populacij ustvarimo naključno in sicer tako, da izmed vseh možnih primerov osebkov izberemo N naključnih. Izbirali smo s ponavljanjem, torej dopuščamo, da imamo v populaciji več identičnih osebkov (z istim kromosomom).

6.1 Adaptacija populacije

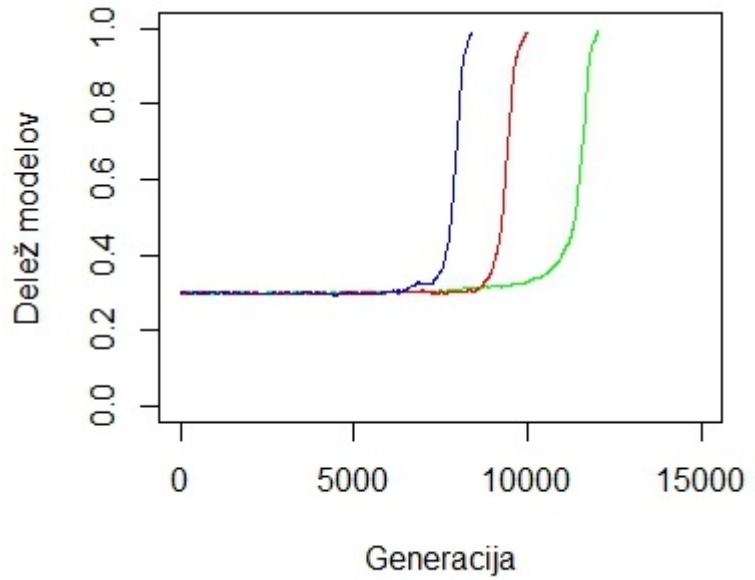
6.1.1 Produktna reprodukcija

Najprej smo opazovali obnašanje deleža modelov za naključno izbrano izjavno funkcijo. Tako funkcijo smo ustvarili tako, da smo vsakemu izmed 2^{28} naborov spremenljivk z verjetnostjo p dodelili vrednost 1. Ker smo populacijo ustvarili naključno, je odstotek modelov pričakovano enak p z majhno deviacijo. Na sliki 6.1 je prikazan potek evolucije za funkcije z različnimi vrednostmi p, ki segajo od 10% pa do 40%. Na sliki 6.2 je prikaz slučajnih funkcij z istimi vrednostmi p.



Slika 6.1: Slučajna funkcija, produktna reprodukcija, $p = 0{,}1;\ 0{,}2;\ 0{,}3;\ 0{,}4$, $n = 28$.

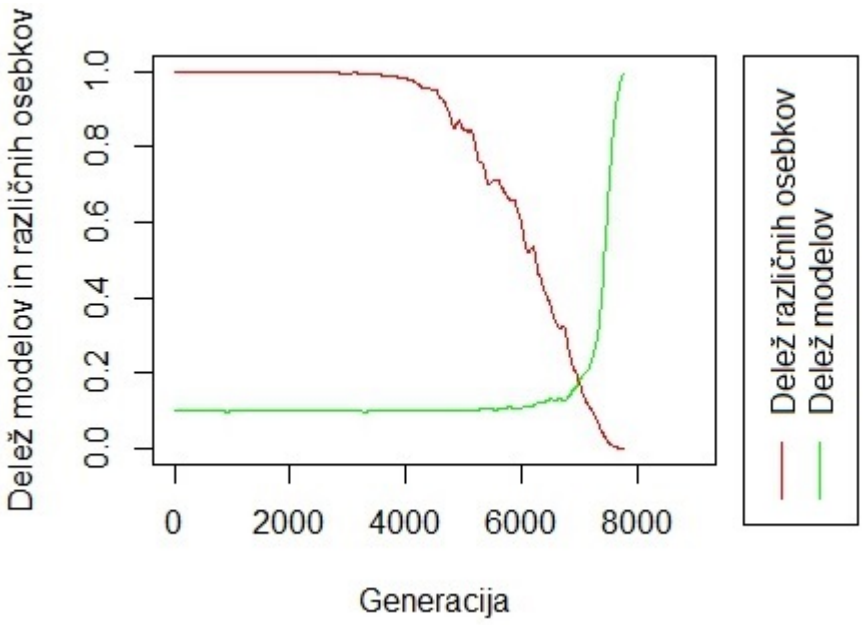
Na Sliki 6.3 je delež modelov prikazan z zeleno črto, delež različnih osebkov pa z rdečo. Delež različnih osebkov populacije izračunamo kot kvocient števila različnih osebkov in velikosti populacije. Če je vseh 10000 osebkov različnih, potem je delež različnih osebkov enak 1. Če pa imamo v celotni populaciji samo 100 različnih osebkov/kromosomov, je delež različnih osebkov



Slika 6.2: Slučajna funkcija, produktna reprodukcija, $p = 0,3$.

enak 0,01.

Na začetku je od 10000 osebkov v populacij skoraj vsak različen od drugega. Proti koncu pričakujemo bistveno manjši delež različnih osebkov.



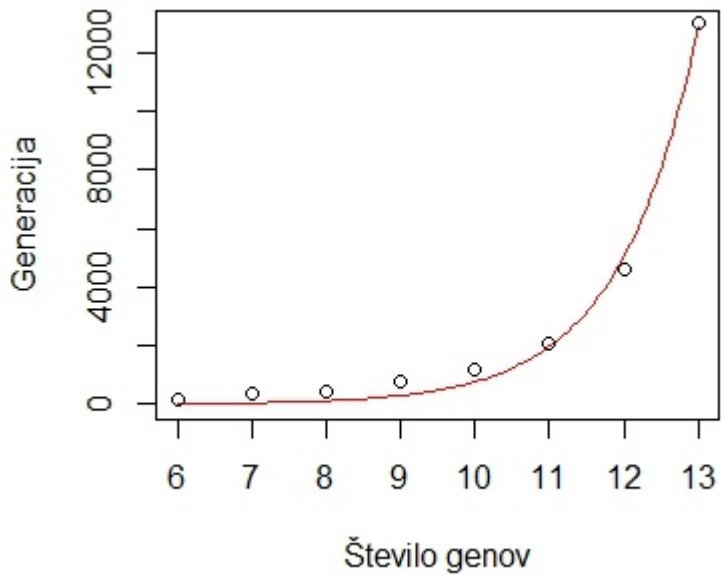
Slika 6.3: Slučajna funkcija, produktna reprodukcija, $p = 0{,}1$, $n = 28$.

6.1.2 Križanje

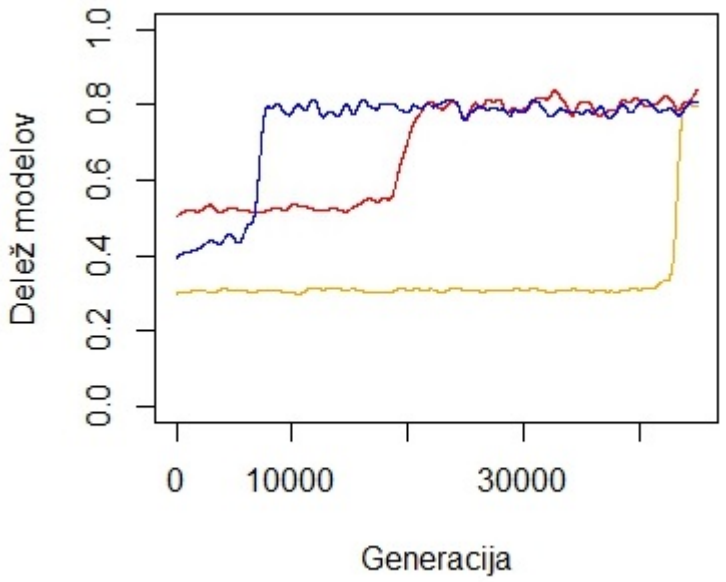
Poleg produktne reprodukcije smo spremljali obnašanje populacije pri uporabi metode križanja. Skupaj s križanjem smo vedno izvajali tudi mutacije. Pri preizkusu metode križanja z 28 geni se tudi po nekaj dneh delež modelov v populaciji ni bistveno spremenil. Opazili smo, da pri manjših dolžinah kromosoma dosežemo konvergenco k modelom. Poskusili smo oceniti, koliko časa bi potrebovali, da dosežemo konvergenco v primeru 28-ih genov. Izvedli smo več preizkusov za število genov od 6 do 28 z do 30000 generacijami in spremljali, koliko generacij smo potrebovali do konvergence. Na naši opremi smo za izvedbo križanja in mutacij v 30000 generacijah potrebovali v povprečju 40 minut. Pri nobenemu od primerov z uporabo 14-ih genov nismo dosegli konvergence. Pri preizkusih z 13 genih smo jo dosegli, vendar ne vedno. Na sliki 6.4 so s krogi prikazane povprečne vrednosti števila generacij za vsak primer dolžine kromosoma med 6 in 13. Ugotovili smo, da odvisnost števila generacij in števila genov raste eksponentno, predstavljena je na sliki 6.4. Če bi potrebno število generacij naraščalo z enako mero, bi za 28 genov potrebovali več milijard generacij, za kar bi na naši opremi potrebovali več let.

Zaradi časovnih omejitev smo se v tem primeru omejili na uporabo osebkov s samo 13 geni. Funkcijo in populacijo smo tudi tukaj ustvarili naključno. Na sliki 6.5 vidimo, da poteka na začetku evolucija dokaj podobno, vendar se zaradi prisotnosti mutacij število modelov ustali okoli vrednosti 80%. Prikaz križanja za funkcije z istimi izbirami deleža modelov je na sliki 6.6.

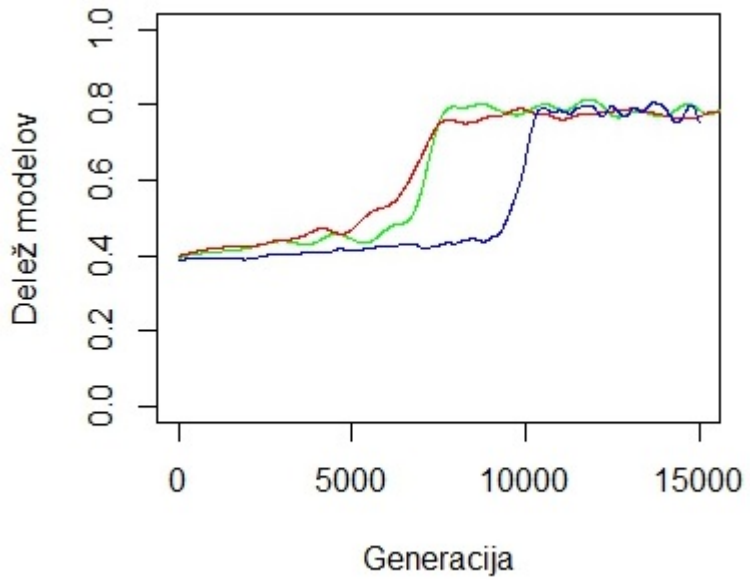
Na sliki 6.7 je z zeleno označen delež modelov, z rdečo pa delež različnih osebkov v populaciji. Vidimo, da je v tem primeru začetni delež modelov približno polovica, saj je vseh možnih naborov le 2^{13}, kar je okoli 8000. Če izberemo 10000 naključnih osebkov, vemo, da se bodo ponavljali.



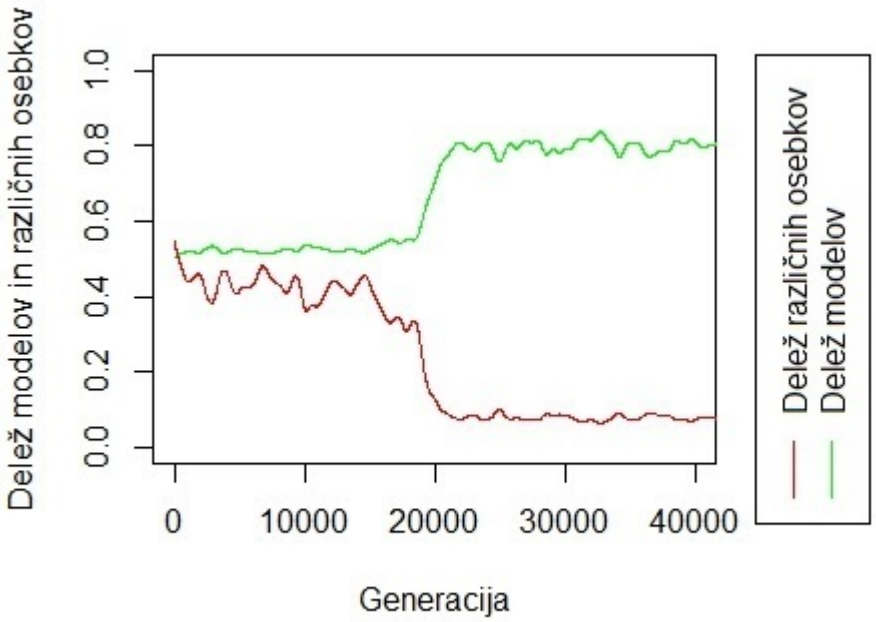
Slika 6.4: Potrebne generacije glede na velikost kromosoma (črna), ocenjena funkcija (rdeča).



Slika 6.5: Slučajna funkcija, križanje in mutacija, $p = 0{,}3;\ 0{,}4;\ 0{,}5$, $n = 13$.



Slika 6.6: Slučajna funkcija, križanje in mutacije, $p = 0{,}4$, $n = 13$.



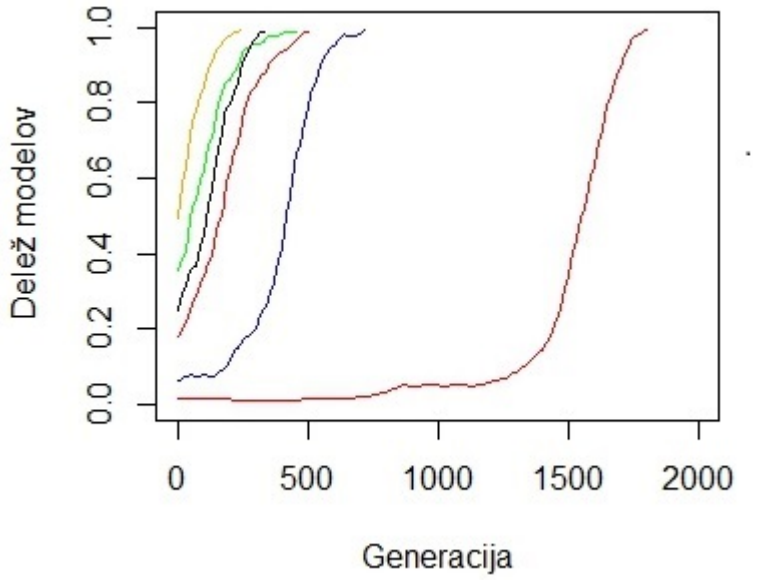
Slika 6.7: Slučajna funkcija, križanje in mutacije, $p = 0{,}5$, $n = 13$.

6.2 Posebne funkcije

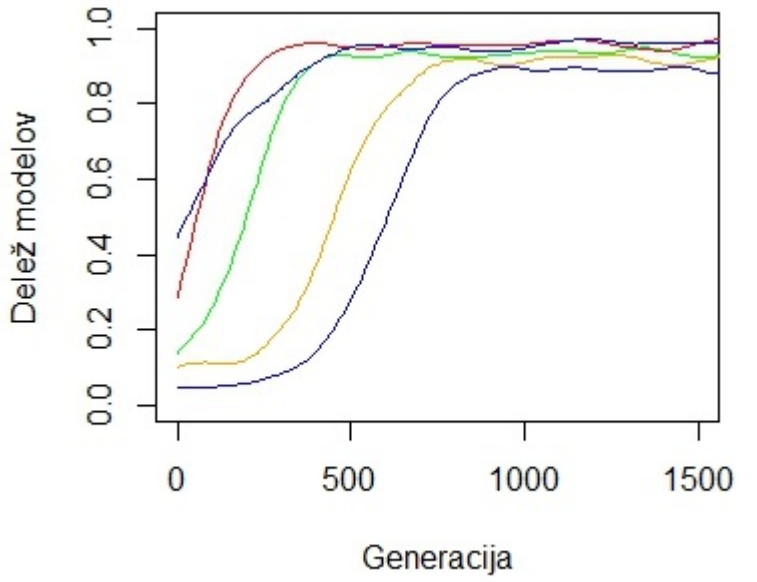
6.2.1 Monotone funkcije

Monotone funkcije smo predstavili drugače kot ostale logične funkcije. Zaradi njihove posebne strukture ni potrebno za vsak nabor shraniti funkcijske vrednosti. Monotono funkcijo opišemo z izbiro naborov $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ in pravilom, da je $\varphi(\mathbf{x}) = 1$ natanko tedaj, ko obstaja indeks i, za katerega je $\mathbf{x} \geq \mathbf{x}_i$. Naborom $\mathbf{x}_1, \ldots, \mathbf{x}_k$ pravimo osnovni nabori monotone funkcije φ. Ker bi imela funkcija, ki ima osnovne nabore z veliko enicami, le majhno število modelov, smo izbirali osnovne nabore z malo enicami z večjo verjetnostjo. Zaradi takega načina predstavitve smo lahko testirali funkcije, ki imajo več genov kot ostale testirane funkcije. Na sliki 6.8 je prikazan potek evolucije za funkcije z različnimi osnovnimi nabori. Ti osnovni nabori so generirani naključno, zato imajo nekatere funkcije večji delež modelov kot ostale. Sam začetni delež modelov v populaciji tudi odraža delež modelov celotne funkcije. Kot vidimo, take funkcije skonvergirajo k modelski populaciji hitreje kot slučajno generirane funkcije, in sicer že po nekaj 100 generacijah.

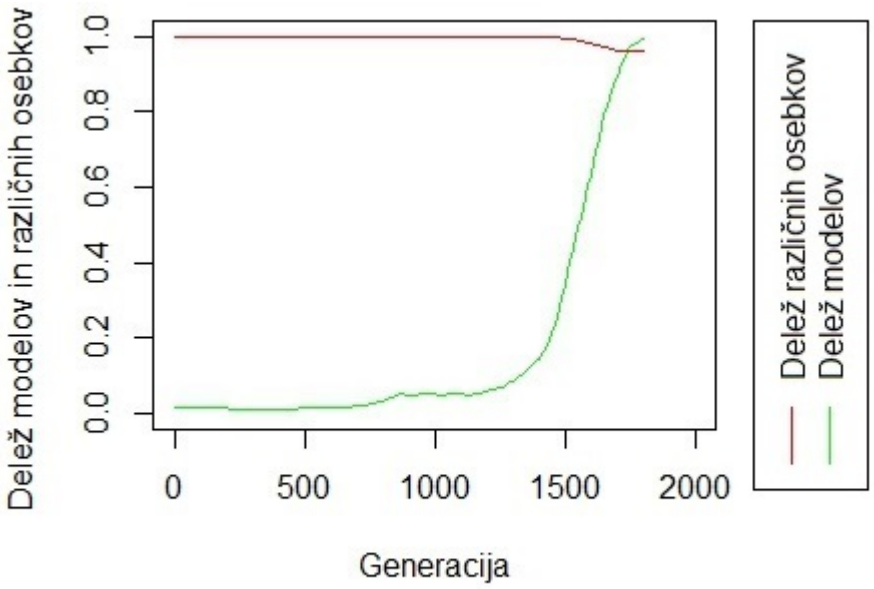
Za razliko od tipične funkcije je populacija pri monotonih funkcijah ohranila velik delež različnih osebkov tudi na koncu evolucije. Na sliki 6.10 je z rdečo črto prikazan delež različnih osebkov in z zeleno delež modelov.



Slika 6.8: Monotona funkcija, produktna reprodukcija, $n = 28$.



Slika 6.9: Monotona funkcija, križanje in mutacije, $n = 28$.

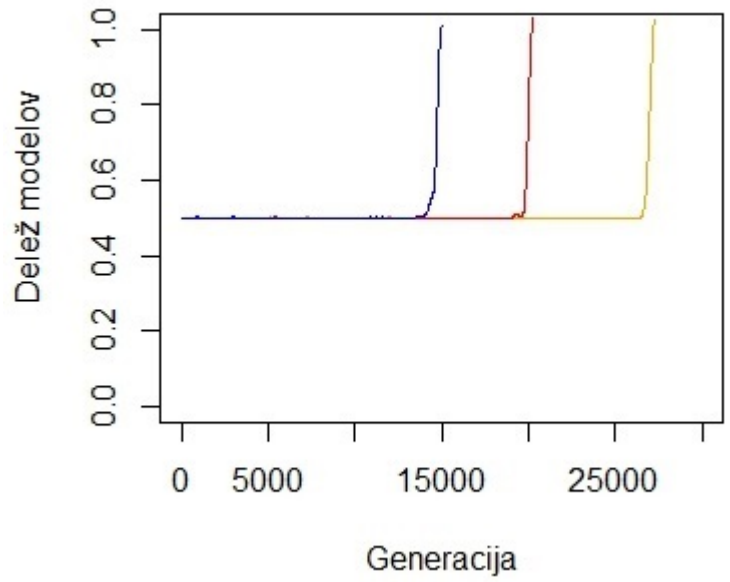


Slika 6.10: Monotona funkcija, produktna reprodukcija, $n = 28$.

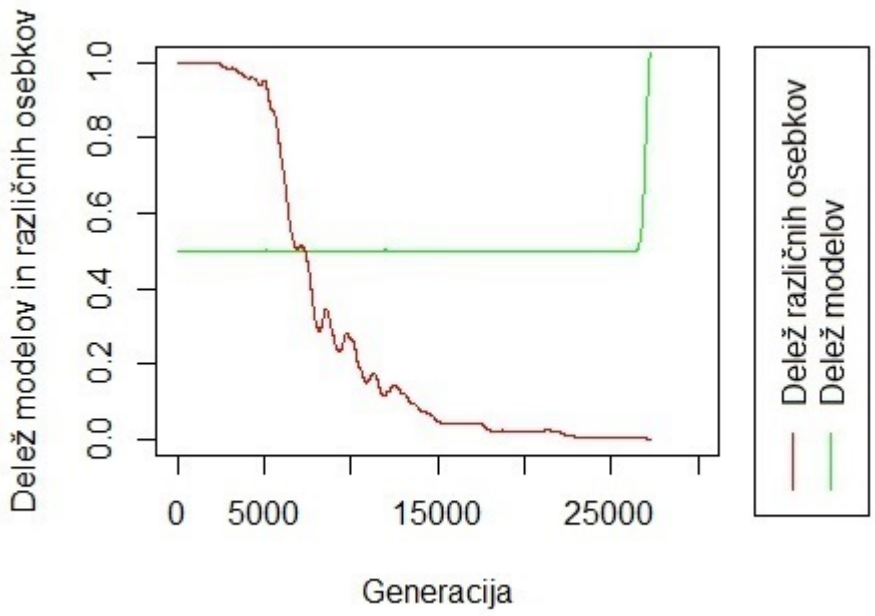
6.2.2 Parnostne funkcije

Livnat in soavtorji [4] pokažejo (glej tudi Poglavje 5), da ob predpostavki neskončne populacije le-te nikoli ne skonvergiramo do modelske populacije. V praksi se izkaže nasprotno. Slika 6.11 prikazuje potek evolucije za parnostno funkcijo z uporabo produktne reprodukcije. Na sliki 6.12 je prikazana sprememba deleža različnih osebkov ob poteku evolucije. Opazimo naslednji fenomen: v teku generacij se veča število fiksiranih genov. To pomeni, da imajo vsi osebki populacije na katerem od genov isti alel. Le-ta se pri produktni reprodukciji ohranja. V končni fazi dosežemo, da ima celotna populacija enak kromosom. V primerjavi s tipično logično funkcijo smo za konvergenco potrebovali vsaj dvakrat toliko časa. Pri uporabi križanja in mutacij pa fiksiranih bitov nimamo, saj nam mutacije lahko naključno spremenijo kateri koli alel. Zaradi tega pri križanju nismo nikoli dosegli konvergence k samim modelom in je delež modelov v populaciji ostal znatno pod 100%.

Sestavili smo tudi funkcijo, ki je resnična samo v odvisnosti od parnosti polovice genov. Druga polovica genov ne vpliva na vrednost. Pri taki funkciji z uporabo križanja prav tako nismo dosegli konvergence. Rezultatov te funkcije nismo prikazali, saj se obnaša podobno kot navadna parnostna funkcija. Če izvajanja ne začnemo s populacijo, ki ima polovico modelov, bo število modelov že po nekaj generacijah skočilo na polovico in se tam ustalilo.



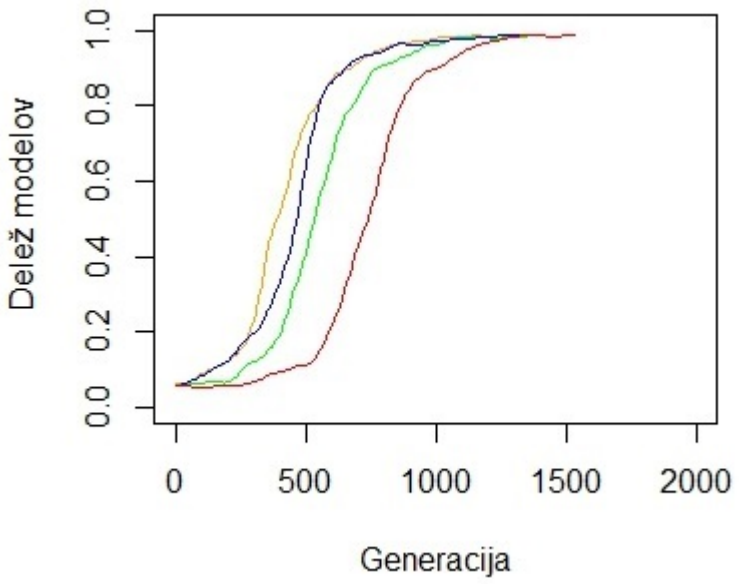
Slika 6.11: Parnostna funkcija, produktna reprodukcija, $n = 28$.



Slika 6.12: Parnostna funkcija, produktna reprodukcija, $n = 28$.

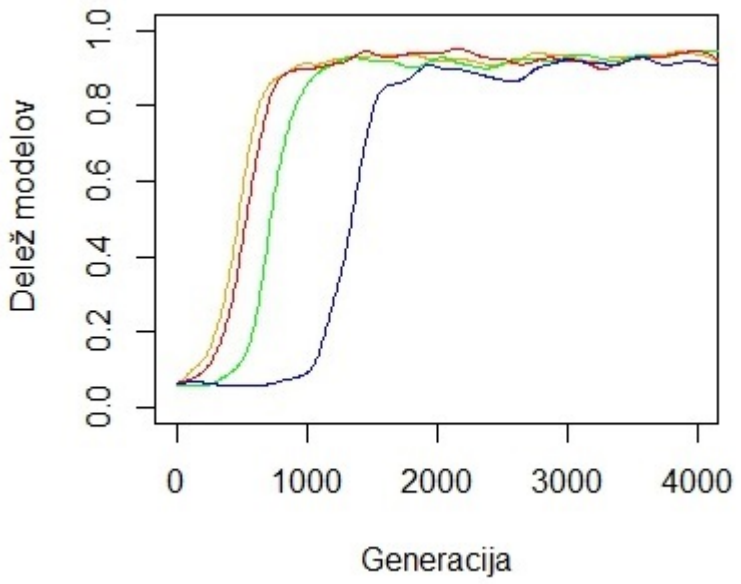
6.2.3 Pragovne funkcije

Na sliki 6.13 je prikazan potek evolucije za pragovno funkcijo z uporabo produktne reprodukcije. Znova opazujemo kromosome z 28 geni. Modeli so vsi nabori, pri katerih je število enic v genih manjše od 9 ali večje od 18. Tudi take funkcije dosežejo konvergenco zelo hitro in se obnašajo podobno kot monotone. Pri uporabi križanja in mutacij prav tako pride do hitre konvergence, delež modelov pa se ustali pri okoli 90%. Glej sliko 6.14.

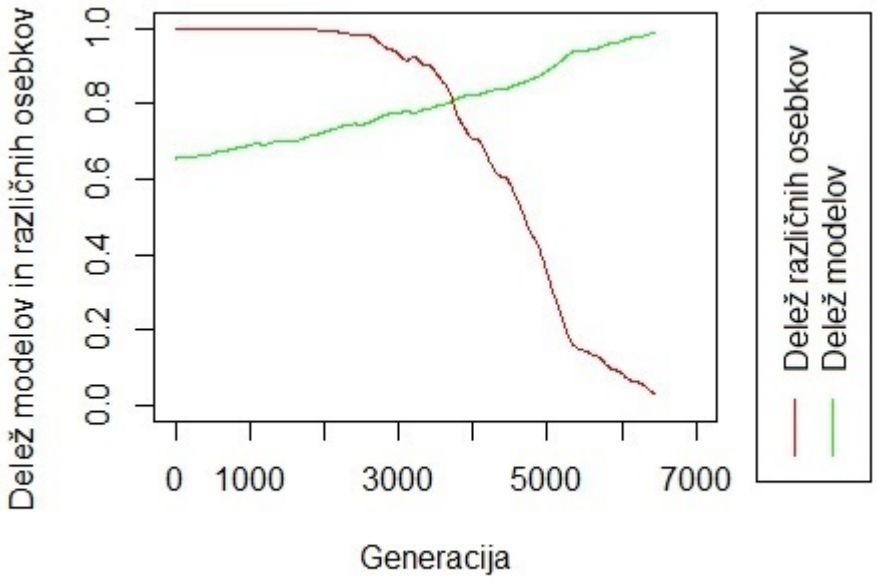


Slika 6.13: Pragovna funkcija, več ponovitev preizkusa, produktna reprodukcija, $n = 28$.

Preizkusili smo tudi ko-pragovno funkcijo in sicer tako, ki je resnična tedaj, ko je število enic večje od 11 in manjše od 16. Na sliki 6.15 je prikazan potek s produktno reprodukcijo. Z zeleno je označen odstotek modelov, z rdečo je označen odstotek različnih osebkov. Križanje z mutacijami se v tem primeru ni izkazalo: tudi ko smo dolžino kromosoma zmanjšali na 13 in ustrezno popravili parametra ko-pragovne funkcije, do konvergence ni prišlo.



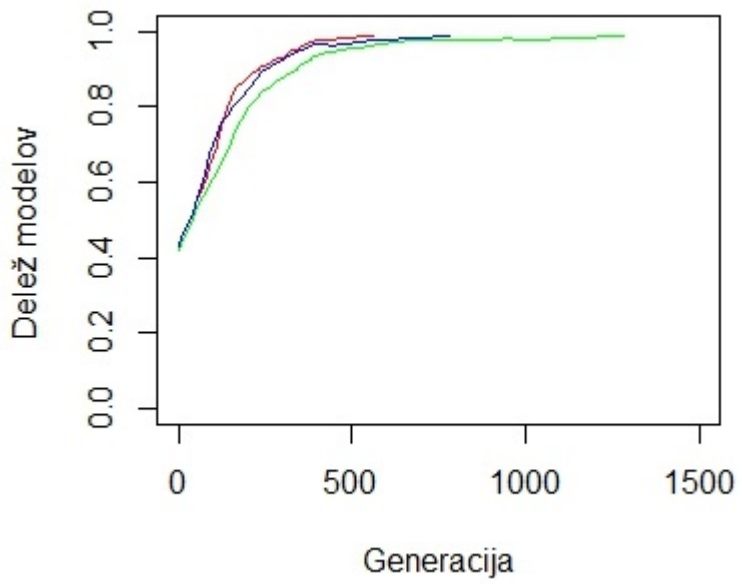
Slika 6.14: Pragovna funkcija, križanje in mutacije, $n = 28$.



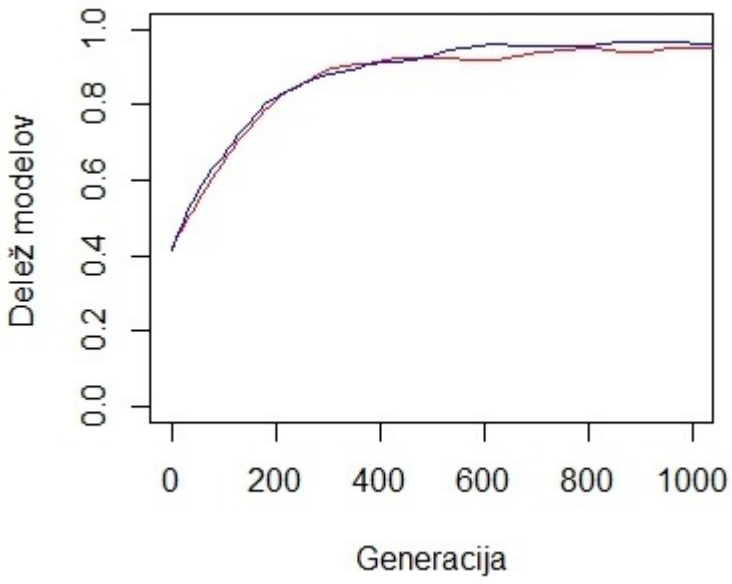
Slika 6.15: Ko-pragovna funkcija, produktna reprodukcija, $n = 28$.

6.2.4 Ostale funkcije

Preizkusili smo tudi funkcijo, ki je resnična natanko tedaj, ko ima leva stran kromosoma večje število enic kot desna stran. Taki funkciji rečemo leva funkcija. V našem primeru z 28 geni je to funkcija, ki je resnična, kadar ima osebek na mestih med 1 in 14 večje število enic kot na mestih med 15 in 28. Slika 6.16 prikazuje potek evolucije z uporabo produktne reprodukcije, slika 6.17 pa njen potek z uporabo križanja. Delež različnih osebkov se tudi pri tej funkciji zelo malo spremeni od začetne vrednosti.



Slika 6.16: Leva funkcija, produktna reprodukcija, $n = 28$.



Slika 6.17: Leva funkcija, križanje in mutacije, $n = 28$.

Poglavje 7

Zaključek

Testirali smo več razredov funkcij in sicer slučajne, monotone, parnostne, pragovne, ko-pragovne in leve. Opazovali smo delež modelov v populaciji, število različnih osebkov in število fiksiranih genov. Pokazali smo, da z uporabo produktne evolucije in z uporabo križanja in mutacij za slučajno logično funkcijo pridemo do konvergence k modelski populaciji. Funkcije, pri katerih smo dosegli hitrejšo konvergenco kot pri slučajni logični funkciji, so: monotona, pragovna in leva. Pri teh funkcijah se delež različnih osebkov ni spremenil. Z uporabo križanja in mutacij pri ko-pragovni in parnostni funkciji konvergence k deležu modelov nismo zaznali. S produktno metodo smo pri parnostni funkciji potrebovali več časa kot pri splošni funkciji. Pri parnostni in ko-pragovni funkciji je pri produktni metodi delež različnih osebkov tudi padel na vrednosti blizu ničle. Prav tako je delež različnih osebkov padel pri slučajni funkciji. Opazili smo, da pri splošni in parnostni funkciji s produktno metodo pride do fiksiranja genov. Pri parnostni funkciji je v končni populaciji večino genov fiksiranih.

Za nadaljnje delo bi bilo smiselno spremljati hitrost konvergence v odvisnosti od kriterijske funkcije in morebitno posplošitev na kriterijske funkcije z več vrednostmi.

Literatura

[1] Christian Blum in Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

[2] Stephen A Cook. The complexity of theorem-proving procedures. V *Proceedings of the third annual ACM symposium on Theory of computing*, strani 151–158. ACM, 1971.

[3] Russell Impagliazzo in Ramamohan Paturi. Complexity of k-sat. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, strani 237–240. IEEE, 1999.

[4] Adi Livnat, Christos Papadimitriou, Aviad Rubinstein, Gregory Valiant, in Andrew Wan. Satisfiability and evolution. V *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, strani 524–530. IEEE, 2014.

[5] Marek Obitko. Introduction to genetic algorithms. `http://www.obitko.com/tutorials/genetic-algorithms/recommendations.php`, 1998. Svetovni splet: 5. september 2017.

[6] Colin Reeves. Genetic algorithms. *Handbook of metaheuristics*, strani 55–82, 2003.

[7] Grace Wahba. *Spline models for observational data*. SIAM, 1990.