

# Automatsko generisanje test primera korišćenjem genetskog algoritma

Seminarski rad u okviru kursa  
Verifikacija softvera  
Matematički fakultet

Petar Mičić, 1104/2018  
micicpetar73@gmail.com

8. januar 2020

## Sažetak

Testiranje i verifikacija softvera predstavlja skup i neizostavni deo razvoja softvera. Jedan od izazovnih zadataka jeste generisanje test primera koji ispunjavaju određene kriterijume testiranja. Automatizovanje generisanja test primera je jedan od ključnih faktora koji ubrzava i smanjuje cenu razvoja softvera i njegovog održavanja. Cilj ovog rada jeste da prikaže tehniku generisanja test primera zasnovanu na biološkim principima.

## Sadržaj

<b>1</b>	<b>Uvod</b>	<b>2</b>
<b>2</b>	<b>Genetski algoritam</b>	<b>2</b>
<b>3</b>	<b>Program</b>	<b>3</b>
<b>4</b>	<b>Algoritam</b>	<b>5</b>
<b>5</b>	<b>Primer</b>	<b>7</b>
<b>6</b>	<b>Složenost algoritma</b>	<b>7</b>
<b>7</b>	<b>Evaluacija</b>	<b>8</b>
<b>8</b>	<b>Zaključak</b>	<b>8</b>

# 1 Uvod

Postoji nekoliko tehnika za automatsko generisanje test primera. Tehnika za slučajno generisanje test primera (eng. *random data generator*) selektuje nasumičnin odabirom ulazne podatke [5]. Tehnika orjentisana strukturi odnosno putanjama programa koristi graf kontrole toka pri generisanju test primera i koristi simboličko<sup>1</sup> izvršavanje da izgeneriše test primer za specifičnu putanju [4]. Više o simboličkom izvršavanju možete naći u [10]. Tehnike orjentisane ciljevima odaberu ulazne podatke koji izvršavaju određeni cilj, npr. izvršavanje određene instrukcije, ne uzimajući u obzir putanju na kojoj se nalazi instrukcija [7]. Tehnike orjentisane analizi koda vrše duboko ispitivanje koda na osnovu koje se generišu ulazni podaci [9].

Nedostaci opisanih tehnika ograničili su njihov obim korišćenja. Tehnika nasumičnog generisanja test primera ne uspeva da pronađe odgovarajući test primer, zbog činjenice da se u okviru nje ne postavlja uslov koji test primer mora da zadovolji. Mana tehnike orjentisane putanji programa ne uspeva da pronađe test primere za putanje koje bi ispitale nedostupne putanje<sup>2</sup> (eng. *infeasible paths*).

Prednost ovog pristupa jeste da ne može neograničeno da vrši pretragu test primera koji pokrivaju nedostupne putanje, zbog svojstva algoritma da nakon određenog vremena prekine izvršavanje programa i ponovo ga pokrene, ukoliko izgenerisani test primeri ne ispune uslove testiranja. Rad i istraživanje je bazirano na [11].

## 2 Genetski algoritam

Genetski algoritmi pripadaju širokoj grupi metaheurističkih algoritama pretrage koji koriste tehnike inspirisane biologijom<sup>3</sup>. Koriste pojmove kao što su selekcija, ukrštanje, mutacija i nasleđivanje. Evolucija je proces u kojoj jedinke koje su najprilagođenije okolini ostavljaju potomstvo, koje je isto tako prilagođeno okolini. Svaka ćelija svakog živog organizma sadrži hromosome<sup>4</sup>. Svaki hromozom sadrži skup gena — blokove DNK. Svaki gen određuje neku osobinu organizma. Reprodukcijski organizam uključuje kombinovanje gena roditelja, i pored toga, malu količinu mutacije. Više o genetskim algoritmima možete pronaći u [6].

U svakoj generaciji deo jedinki se izdvaja za reprodukciju i generisanje nove generacije. Način odabira jedinki za reprodukciju određen je funkcijom prilagođenosti, i generalno, jedinke koje su prilagođenije, imaju veće šanse da imaju potomstvo. U drugim pristupima, jedinke je moguće birati i slučajno, ali tako da prilagođenije jedinke imaju veću verovatnoću da budu izabrane. Ovakvim pristupom moguće je odabrati i one jedinke koje nisu najbolje prilagođene.

<sup>1</sup>Umesto dodavanja konkretnih vrednosti input parametrima, dodaju se simboli koji predstavljaju konkretne vrednosti. Izvršavanje se odvija uobičajeno, s tim što se formule sastoje od simboličkih promenljivih, umesto konkretnih vrednosti.

<sup>2</sup>Predikat  $traversable(path) = true$ , ako postoji input takav da se obilazi putanja  $path$ . Nedostižna putanja je ona za koju važi  $traversable(path) = false$ , za svaki input [8].

<sup>3</sup>Dr. John Holland je predstavio svoju ideju u knjizi "Prilagođavanje u prirodnim i veštačkim sistemima", koje predstavlja jedno od najcitiranijih radova u oblasti veštačke inteligencije i evolutivnog računarstva [2].

<sup>4</sup>Muški Y hromozom počeo je sa podjednakim brojem gena kao ženski, ali stotinama miliona godina polako se raspadao i zadržao manje od sto zdravih gena do današnjeg dana. To uključuje i gen koji određuje pol, tzv. „glavni prekidač“ zahvaljujući kojem embrion postaje muško ili žensko [1].



Slika 1: Hromozom

U procesu reprodukcije učestvuju jedinke koje su izabrane u fazi selekcije. Ove jedinke se nazivaju roditelji, dok se procesom ukrštanja dobija jedinka ili dve nove jedinke koje nazivamo neposrednim potomcima. Očekivano je da deca naslede osobine roditelja, uključujući njihovu prilagođenost pa i da budu bolje prilagođene od njih. Postoji nekoliko jednostavnih načina reprodukcije. U jednoj varijanti (jednopoloziciono ukrštanje) dovoljno je odabrati tačku prekida, i prekombinovati nizove podataka. Moguće je i višepoziciono ukrštanje, pri čemu se bira proizvoljan broj tačaka koji mora da bude manji od dužine hromozoma. Kod uniformnog ukrštanja svaki podatak se sa verovatnoćom  $p$  prenosi na prvo dete i sa  $1 - p$  na drugo dete. Verovatnoća je obično jednaka 0.5, a može biti i drugačija.

Mutacija je operator koji na određeni način menja jedan deo jedinke. Obično je verovatnoća mutacije veoma mala. Mutacija menja jedan ili više gena. Od jedne jedinke dobija se nova jedinka. Verovatnoća da će neki gen biti promenjen je parametar algoritma koji se određuje eksperimentalno. Uloga mutacije je da spreči da jedinke postanu suviše slične roditeljima i da pomognu u obnavljanju izgubljenog genetskog materijala.

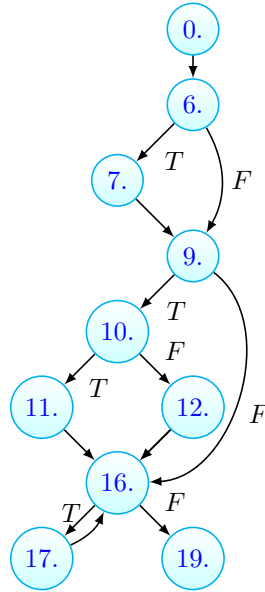
### 3 Program

Dat je prikaz jednostavnog C programa (kod 1), i njemu odgovarajući graf kontrole toka (slika 2) i graf zavisnosti izvršavanja (slika 3).

```

1 #include <stdio.h>
2 int main() {
3     int x, y, z, t;
4     scanf("%d %d %d %d", &x, &y, &z, &t);
5
6     if (x < y) {
7         x++;
8     }
9     if (z < t) {
10        if (y < t) {
11            y++;
12        }else {
13            z++;
14        }
15    }
16    while (t < x) {
17        t++;
18    }
19    return 0;
20 }
```

Listing 1: Primer koda sa nekoliko grana



Slika 2: Graf kontrole toka

Kada se međukod<sup>5</sup> izdela na osnovne blokove, predstavljamo tok izvršavanja programa grafom kontrole toka (eng. *control flow graph*)[3]. Čvorovi grafa kontrole toka su osnovni blokovi. Grana vodi od bloka B do bloka C ako i samo ako se prva instrukcija bloka C izvršava neposredno nakon poslednje instrukcije bloka B. Graf kontrole toka razlikuje ulazni (početni) i završni (krajnji) čvor. Postoje dva načina za uvođenje grane:

- blok B se završava uslovnim ili безусловnim skokom
- blok C neposredno se izvršava nakon bloka B, a B se ne završava безусловnim skokom

**Definicija 3.1.** Čvor  $V$  je dominantan nad čvorom  $W$  u grafu  $G$ , ako svaki usmereni put od čvora  $W$  do čvora  $exit$ , sadrži  $V$ .

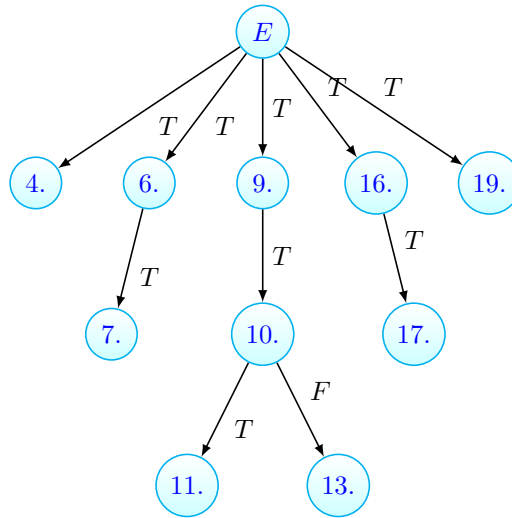
**Definicija 3.2.** Neka su  $X$  i  $Y$  čvorovi grafa  $G$ . Čvor  $Y$  je kontrolisan od strane čvora  $X$  ako:

1. postoji usmereni put  $P$ , u grafu  $G$ , od  $X$  do  $Y$  sa mogućim  $Z$  u  $P$  (ne uključujući  $X$  i  $Y$ )
2.  $X$  nije kontrolisan od  $Y$

Graf zavisnosti izvršavanja se dobija primenom *DFS* obilaskom grafa kontrole toka.

Aciklični put u grafu zavisnosti od početnog čvora do čvora naredbe predstavlja skup predikata koji mora da bude zadovoljen da bi čvor naredbe bio izvršen. Takav put se naziva put uslova izvršavanja (eng. *control dependence predicate path*). Za dati program, izvršavanje instrukcije rednog broja 13, odgovarajući put ima ograničenja  $(ET, 9T, 10F)$ .

<sup>5</sup>Međukod predstavlja kod između izvornog koda i ciljnog koda konkretne arhitekture.



Slika 3: Graf zavisnosti izvršavanja

## 4 Algoritam

U ovom poglavlju biće opisan sam algoritam. Pseudokod 2 predstavlja opšti genetski algoritam za generisanje test primera

```

1 begin
2 /* Step 1: Initialization and set up */
3 Create CDGPaths
4 Create and Initialize Scoreboard
5 Generate CurPaplulation
6
7 /* Step 2: Generate test cases */
8 while ((some (r,unmarked) E TestReq) and not OutOfTime0) do
9   Select unmarked Target from TestReq
10   while (Target not marked and not MazAttempts0 do
11     Compute tness values of CurPaplulation using
12     CDGPaths
13     Sort CurPaplulation according to tness
14     Select parents of NewPaplulation
15     Generate NewPaplulation from selected members of
16     CurPaplulation
17     Execute ngmm on each member of NewPaplulation
18     Update Scoreboard and mark TestReq to reflect those
19     test requirements that are satisfied
20   endwhile
21 endwhile
22 /* Step 3: Clean up and return */
23 Final = test cases that satisfy TestReq
24 return (Final, TestReq)
25 end

```

Listing 2: Algoritam za automatsko generisanje test primera

Ulazni podaci algoritma čine originalni program — predstavlja instrumentalizovanu verziju programa čiji test primere algoritam treba da izgeneriše, graf zavisnosti izvršavanja, inicijalna populacija — test primeri od kojih počinjemo da generišemo test primere, uslovni predikati — kriterijum koji moraju da zadovolje test primeri. Potrebno je definisati sledeće vrednosti: semafor — struktura u kojoj su označeni ispunjeni kriterijumi, skup predikata putanja u grafu, trenutna i nova populacija predstavljaju

skupove test primera koji se obrađuju, ciljni predikat (u nastavku *Target*) test primera — predstavlja cilj ka kom se usmerava generisanje, funkcija *maxAttempts()* — ograničava broj izgenerisanih jedinki koji teži da zadovolji ciljni predikat, *outOfTime()* — ograničenje generisanja novih populacija.

Algoritam generiše putanju (eng. *CDG path*) koja predstavlja čvor predikata koji sadrži taj predikat i sve predikate od početnog čvora. Skor tabela koja, na primer, prati frekvenciju izvršavanja naredbi može da bude vektor integera dimenzije  $n$ , gde je  $n$  broj naredbi programa, koji se uveća za jedan kada program izvrši naredbu. Inicijalni test primeri se dodaju u početnu populaciju. Ukoliko su algoritmu potrebni dodatni test primeri, oni se generišu na slučajan način uz ograničenja definisana od strane korisnika. Da bi se izbeglo zadovoljavanje nemogućih testnih uslova, spoljašnja petlja iterira sve dok se ne označe svi neoznačeni test uslovi ili dok se ne postigne određen broj iteracija. Nakon generisanja, neoznačeni test primeri se moraju manuelno ispitati. Glavni fokus unutrašnje petlje jeste jedinstvena jedinka koja mora da ispunjava uslov odabran iz skupa uslova *TestReq*. Petlja iterira dok se ne ispuni taj uslov, ili dok se ne postigne određeni broj iteracija. Za svaku test primer iz tekuće populacije računa se funkcija prilagođenosti. Ako test primer  $t_i$  sadrži više uslova u *CDGPaths* za određeni cilj, tada funkcija prilagođenosti dodeljuje veću vrednost test primeru  $t_i$ . Zatim se tekuća populacija sortira prema vrednostima funkcije prilagođavanja.

Algoritam teži da koristeći graf zavisnosti izvršavanja, traži test primer koji zadovoljava *Target*. Ako test primer ima više zadovoljenih predikata na datoj putanji, tada test primer je bliži da zadovolji *Target*. Takođe, kombinacijom dva test primera koja su bliža cilju, postoji veća šansa da se izgeneriše test primer koji zadovoljava *Target*.

Graf zavisnosti izvršavanja se koristi da se pronađu predikati koji zadovoljavaju cilj *Target*. Pretpostavka da je pronalaženje test primera koji zadovoljava cilj je bolja od generisanja test primera slučajnom metodom ili onih koji nisu blizu cilja. Test primer je blizu cilja ako ima više ispunjenih predikata tog *Target*. Ako test primer u većoj meri ispunjava cilj, smatra se dobrim, odnosno da poseduje dobar materijal, takav da kombinovanjem sa drugim test primerima se dobija test primer koji ima veće šanse da zadovolji cilj.

Zatim se biraju najprilagođenije jedinke, u zavisnosti od fitnes evaluacije, koje se dodaju u novu populaciju. Za vrednost funkcije prilagođenosti  $f_i$  se računa verovatnoća po formuli  $p_i = \frac{f_i}{\sum_{i=1}^n f_i}$ , pri čemu je  $n$  — broj jedinki u tekućoj populaciji. Treba napomenuti da vazi  $\sum_{i=1}^n p_i = 1$ . Verovatnoća da će test primer  $t_i$  biti odabran je tačno  $p_i$ . Proces selekcije se ponavlja sve dok je dovoljno roditelja za generisanje nove populacije. Nakon procesa selekcije, algoritam generiše nove jedinke ukrštajući roditelje i potencijalnom mutacijom roditelja. Prvobitni (jednopoloziciono ukrštanje) metod zahteva dva roditelja i proizvodi dva deteta koja se dodaju u novu populaciju. Zatim, sa unapred definisanom verovatnoćom se izvršava mutacija jedinki. Nakon izgenerisane nove populacije, za svaki izgenerisan test primer se izvršava originalni program, a potom se ažurira skor tabela. Ako barem jedan test primer zadovoljava *Target*, završava se iteracija unutrašnje petlje i pokušava se sa pronalaženjem novog cilja iz skupa uslova *TestReq*.

Oznaka test primera t	Ulazni podatak	Oznake izvršenih instrukcija
t1	(12 3 6 9)	(4,6,9,10,11,16,17,19)
t2	(23,12,12,5)	(4,6,9,16,17,19)
t3	(14,16,20,6)	(4,6,7,9,16,17,19)
t4	(5 6 9 10)	(4,6,7,9,10,11,16,19)

Slika 4: Inicijalna populacija i instrukcije koje izvršavaju

## 5 Primer

U ovoj sekciji biće ilustrovano izvršavanje algoritma i generisanje nekoliko novih test primera. Testiranje je usmereno ka pronalaženju grane koja izvršava instrukciju broj 13., odnosno čiji graf zavisnosti izvršavanja ima vrednosti  $(ET, 9T, 10F)$ . Kao inicijalna populacija nasumično se biraju jedinke, odnosno ulazni parametri, videti tabelu 5. Takođe se prikazuju oznake pokrivenih instrukcija za dati test primer. Oznake predstavljaju redni broj instrukcije originalnog programa prikazanog u kodu 1. Zatim se vrši izračunavanje funkcije prilagođavanja za svaku jedinku u tekućoj populaciji. Verovatnoće  $p_i$  da će jedinke biti odabrane su redom  $(0.5, 0, 0, 0.5)$ . Funkcija prilagođenosti je definisana tako da veću vrednost dobijaju jedinke koje izvršavaju instrukciju broj 9. Jedinke koje se uzimaju za generisanje nove populacije su  $t_1$  i  $t_4$ , na osnovu kojih se generiše nova populacija. Jednopolozicionim ukrštanjem ove dve jedinke na različitim pozicijama i mutacijom sa verovatnoćom 2%, dobijaju se sledeće jedinke  $(12, 10, 9, 10)$ ,  $(5, 6, 6, 9)$ ,  $(12, 6, 9, 10)$ ,  $(5, 3, 6, 9)$  od kojih  $t_1$  zadovoljavaju *Target*, a to je izvršavanje instrukcije broj 13.

## 6 Složenost algoritma

U ovoj sekciji biće prikazana vremenska i prostorna složenost algoritma. Varijable korišćene za određivanje složenosti su:

- $n$  — broj instrukcija u programu
- $P$  — veličina populacije
- $e$  — vreme izvršavanje instrumentalizovanog programa
- $k$  — dubina grananja uslova i petlji
- $r$  — broj uslova test primera
- $m$  — maksimalan broj pokusa za generisanje test slučajeva
- $L$  — dužina jedinke

Deo konstrukcije i inicijalizacije *CDGpaths* ima složenost  $O(n^2)$ , inicijalizacija semafora ima složenost  $O(n)$ , a generisanje populacije  $O(P)$ . U unutrašnjoj petlji se vrši izračunavanje fitnes funkcije nad svakom jedinkom, generiše nova populacija i ažurira skor tabela, pri čemu svaka od ovih operacija ima složenost  $O(P)$ . Unutar petlje se takođe sortira populacija  $O(P \times \log P)$  i izvršava instrumentalizovan program za svaki test primer  $O(P \times e)$ . Kako je unutrašnja petlja ograničena na maksimalan broj izvršavanja od  $m$ , tada algoritam ima u najgorem slučaju složenost  $O(m \times \max(P \times e, P \times \log P))$ . Izvršavanje spoljašnje petlje je ograničeno vremenski (parametar koji je određen od strane korisnika) ili dok svi ciljevi nisu ispunjeni, tako da broj iteracija spoljašnje petlje može da varira.

Prostorna složenost algoritma određena je glavnim strukturama koje koristi. CDGpaths zauzima prostor  $O(n \times k)$ , dok su tekuća i nova populacija  $O(P \times L)$ . Semafor zauzima prostor veličine  $O(r)$ , gde  $r$  može biti broj pokrivenih instrukcija, broj izvršenih grana, broj pokrivenih putanja, i drugo.

## 7 Evaluacija

U ovoj sekciji biće upoređena efektivnost generisanja test primera nasumičnim generisanjem (u nastavku *Random*) i genetskim algoritmom. Testiranje<sup>6</sup> je izvršeno nad 3 C programa: *bisect.c* — za dato epsilon i  $X$  vrednost, program računa koren vrednosti  $X$  (do na epsilon tačnost) koristeći metod bisekcije, *fourballs.c* — za date 4 integer vrednosti, program računa težine kuglica koje su u srodstvu, *tritype.c* — za date 3 integer vrednosti, program ispituje tip trougla sagrađen od dužina stranica (ako postoji). Genetski algoritam postigao je 100% pokrivenost koda nakon 29 iteracija, odnosno izgenerisao je 29 novih populacija, pri testiranju programa *bisect.c*. Random pristup je postigao 100% pokrivenost koda nakon 40 iteracija. *Random* je izgenerisao 100 test primera u jednoj iteraciji, odnosno 100 je veličina jedne populacije.

Rezultati testiranja programa *fourballs.c* pokazuju da genetski pristup postiže 100% pokrivenost koda nakon 95 iteracija, nasuprot random pristupu koji istu pokrivenost postiže za 159 iteracija. Kod programa *tritype.c*, random pristup postiže 100% pokrivenost koda nakon 1259 iteracija, dok genetski pristup nakon 145 iteracija.

Rezultati pokazuju da kod programa koji imaju manju kompleksnost, 100% pokrivenost se postiže vrlo brzo kod oba pristupa, dok kod programa koji imaju brojnije ugnježdene petlje ili grananja, genetski pristup nadmašuje *Random*, zbog uslova putanja koje je teško zadovoljiti.

## 8 Zaključak

Opisani algoritam koristi heuristiku pretrage za optimalno rešenje koja se naziva genetski algoritam. Genetski algoritam koristi graf kontrole toka i graf zavisnosti programa, u čemu se ogleda njegova jednostavnost. Pronalaženje test primera je orijentisano ka definisanom kriterijumu. Tehnika koja može u drastičnoj meri da ubrza pretragu test primera je paralelno izvršavanje<sup>7</sup>. Korisni aspekt opisanog algoritma je da ima potencijal kada su u pitanju veliki sistemi (eng. *large scale systems*). Algoritam je takođe u potpunosti automatski (osim podešavanja inicijalne konfiguracije), i ne zahteva ljudsku interakciju sa programom u toku generisanja podataka.

Za dalje istraživanje, moguće je izvršiti analizu ulazne konfiguracije algoritma i njegov uticaj na generisanje finalnog skupa test primera koji omogućavaju efikasnije testiranje programa sa velikim brojem operacija.

---

<sup>6</sup>Rezultati testiranja preuzeti su iz [11]

<sup>7</sup>Paralelno izvršavanje omogućava upotrebu većeg broja procesorskih jedinica pri izvršavanju jednog posla.



## Literatura

- [1] National geographic. <https://www.nationalgeographic.rs/vesti/1001-da-li-muskarcima-preti-izumiranje.html>.
- [2] New york times. <https://www.nytimes.com/2015/08/20/science/john-henry-holland-computerized-evolution-dies-at-86.html>.
- [3] Ravi Sethi Alfred V. Aho, Monica S. Lam and Jeffrey Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2009.
- [4] Dawson Engler Cristian Cadar, Daniel Dunbar. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. *USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008)*, 22, 2008.
- [5] C. U. Munoz D. L. Bird. Automatic generation of random self-checking test cases. *IBM Systems Journal*, 22, 1983.
- [6] Predrag Janicic. *Vestacka inteligencija*. 2018.
- [7] Bogdan Korel. Automated Software Test Data Generation. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 16, 1990.
- [8] J. Laski and W. Stanley. *Software Verification and Analysis*. Springer-Verlag, London, 2009.
- [9] Keith Mander Nigel Tracey, John Clark. Automated Program Flaw Finding using Simulated Annealing. *ISSTA '98 Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis*, 16:73–81, 1998.
- [10] Daniele Cono D'elia Camil Demetrescu and Irene Finocchi Roberto Baldoni, Emilio Coppa. A Survey of Symbolic Execution Techniques. *ACM Computing Surveys*, 51, 2018.
- [11] Mary Jean Harrold Roy P. Pargas and Robert R. Peck. Test-Data generation using genetic algorithm. *Journal of software testing, verification and reliability*, 2, 1999.