



Матурски рад из Програмирања

Хеуристичко програмирање

Ученик:
Иван Бокор IV₁₂

Ментор:
Вукман Кораћ

Земун, мај 2023.

Садржај

1	Увод	1
2	Пример рада генетског алгоритма	2
2.1	Иницијализовање популације	2
2.2	Репрезентација јединки	2
2.3	Селекција	2
2.4	Укрштање	3
2.5	Мутација	4
2.6	Добијена решења	4
3	Тестирање генетског алгорита	4
3.1	Грејево кодирање и иницијализација популације	4
3.2	Селекција, укрштање и мутација	5
3.3	Приказ рада алгоритма	5
4	Резултати тестирања	7
5	Закључак	9
6	Литература	9

1 Увод

Алгоритам представља низ дефинисаних корака (или операција) који воде ка решавању неког проблема. Алгоритмима су се дуго бавили математичари, али је тек након конструисања првих рачунара њима посвећена значајнија пажња. Током 20. века осмишљени су алгоритми за решавање разних сложених проблема. Међутим, неки проблеми су били сувише комплексни те су алгоритми осмишљени за њихово решавање били неефикасни - захтевали су пуно меморијског простора или је просто било потребно пуно времена за њихово извршавање (нпр. Проблем трговачког путника). Стога су осмишљени алгоритми који повећавају ефикасност на рачун тачности решења. Такви алгоритми се често називају хеуристике а коришћење тих метода се зове хеуристичко програмирање. У овом раду бавићемо се генетским алгоритмима који припадају тој категорији.

Генетски алгоритам (у даљем тексту ГА) представља хеуристичку методу оптимизације[1] који моделује механизме еволуције у природи. Идеја је да почнемо од неког скупа насумичних решења и онда еволуцијом добијемо оптимално решење. Еволуција се одвија помоћу два механизма: **укрштања** и **мутације**. Укрштањем два решења добијамо ново потенцијално решење проблема а мутацијом насумично мењамо решење чиме се подиже степен различитости у целом скуп решења. Процес бирања решења која ћемо укрстити се назива **селекција**. Пошто је терминологија готово иста као она која се користи у биологији, скуп потенцијалних решења неког проблема ћемо називати **популацијом** а неко конкретно решење **јединком**. Свака јединка је одређена конкретним **генетским кодом** (такође се користе термини ген, хромозом) који енкодира неко решење проблема. С обзиром да ГА моделује еволуцију, има смисла говорити и о генерацијама. Генерацију представљају све јединке које су се нашле у датом тренутку у популацији. У почетној генерацији се налазе оне јединке које су насумично генерисане, а свака следећа настаје применом еволутивних механизма (укрштања и мутације) на ону претходну.

Математички говорећи, ГА је оптимизациона метода усмереног случајног претраживања простора решења[1]. Циљ алгоритма је пронаћи глобално решење неког проблема, што се може поистоветити са тражењем глобалног минимума или максимума неке функције. Ако осмислимо неку функцију која ће евалуирати потенцијална решења проблема до којих генетски алгоритам буде дошао, оптимално решење ћемо управо наћи у максимуму или минимуму те функције (у зависности од типа проблема који се решава). Зато дефинишемо **функцију добротe** $fitness(x)$ или **функцију грешке** $cost(x)$ која на неки начин евалуира наша решења, тј. сваком решењу придружује неки реалан број којим меримо колико је дато решење "добро" (отуда и назив функција добротe). Постоји више начина за одређивање да ли је алгоритам стигао до максимума/минимума али онај најлакши је да се унапред дефинише број итерација алгоритма. За то дефинишемо глобални параметар који чува тај број итерација. Осим броја итерација, глобалним параметрима су одређене и следеће величине: број јединки у популацији,

проценат популације који мутира, величина и дужина хромозома. У одељцима 3. и 4. ће бити испитано и приказано како ови параметри утичу на перформансе ГА.

У овом раду биће приказан начин рада генетског алгоритма решавањем два проблема: први проблем је тражење жељене ниске а други тражење максимума неке функције. Први проблем служи за упознавање са функционисањем генетског алгоритма, док други служи за његово тестирање и евалуацију. За оба проблема испрограмиран је генетски алгоритам у програмском језику C++. Код за оба алгоритма је могуће пронаћи у GitHub репозиторијуму на следећем линку: github.com/blin04/maturski.

2 Пример рада генетског алгоритма

Начин рада генетског алгоритма ћемо објаснити на проблему претраге ниски. Односно, алгоритам ће имати задатак да "пронађе" неку задату ниску, нпр. 'Земунска гимназија'. Ниску коју треба наћи се уноси приликом покретања програма и чува се у глобалном параметру *GOAL*. Дужина те ниске се означава са n .

2.1 Иницијализовање популације

За почетак, потребно је иницијализовати скуп, тј. популацију, потенцијалних решења. Пошто може бити унета било каква ниска, почетни скуп садржи насумично генерисане ниске. У популацији се налази 200 јединки. Алгоритам ће изгенерисати 500 генерација како би смо били сигурни да је добијено решење оно најоптималније.

2.2 Репрезентација јединки

Као што су људи једнозначно одређени ДНК молекулама, тако су и јединке у популацији одређене неким генетским кодом. Генетски код, тј. хромозом, може да буде било шта, под условом да је могуће на основу њега конструисати потенцијално решење проблема. Обично се користе бројеви, ниске или неке друге структуре са којима рачунари лако манипулишу. У овом случају, генетски код јединке ће бити управо та ниска коју она "представља". То значи да дужина генетског кода мора бити n .

2.3 Селекција

Процес селекције служи да би се изабрали родитељи - јединке од којих ће настати 2 нове јединке (тј. 2 нова решења). Како би алгоритам стигао до оптималног решења, потребно је да се бирају што бољи родитељи, како би и њихова деца била што боља. За то служе функција добороте или функција грешке које вреднују јединке. У овом случају користимо функцију грешке, која евалуира колико је нека јединка лоша. Функција је дефинисана на следећи начин:

$$cost(s) = \sum_{i=1}^n |s_i - GOAL_i|$$

где је s потенцијално решење а $GOAL$ ниска коју желимо достићи. Овима постижемо да јединке које представљају ниске чији су карактери ближи карактерима (по ASCII вредности) коначне ниске имају мању вредност функцију грешке (напоменимо само да оваква дефиниција функције грешке није добра, али ће послужити за демонстрацију генетског алгорита). Пошто је циљ да се бирају што бољи родитељи, сваком родитељу се додељује нека вероватноћа да буде изабран која ће бити обрнуто пропорционална функцији грешке ($p_i \propto \frac{1}{cost(x)}$). Избор родитеља се онда врши *Roulette wheel* алгоритмом. Тај алгоритам се може објаснити као да имамо неки точак са фиксираним показивачем на врху, који је издељен на онолико делова колико има јединки у популацији (делови изгледају као парћини пие). Површина сваког дела је пропорционална вероватноћи да та јединка буде изабрана за родитеља. Бирање родитеља се врши тако што се точак заврти, па се изабере она јединка на коју показује показивач након заустављања точка. Овакав начин селекције се зове једноставна селекција и захтева да приликом укрштања направимо нови низ у који убацујемо новодобијене јединке. Прављење новог низа се може избећи коришћењем елиминацијске селекције, о чему ће бити речи у одељку 3.2. Испод је дат псеудокод *Roulette wheel* алгоритма.

```

 $r \leftarrow random(0, TOTAL\_COST)$ 
 $suma \leftarrow 0$ 
 $izabran \leftarrow NULL$ 
for  $i \leftarrow 1 \dots n$  do
     $suma \leftarrow suma + cost(p_i)$ 
    if  $r \geq suma$  then
         $izabran \leftarrow p_i$ 
    end if
end for

```

2.4 Укрштање

Након што се родитељи изабери, потребно је извршити укрштање генетских кодова родитеља како би се добио нови генетски код који представља нову јединку. Постоје разни алгоритми укрштања и обично у њима постоји одређена доза насумичности, како би се симулирао процес укрштања гена у природи. За потребе ове демонстрације имплементиран је алгоритам који функционише на следећи начин: за неку позицију i ($i \in [1, n]$) пореде се карактери који се налазе на том месту у генетском коду родитеља и у генетски код детета се додаје онај који је ближи одговарајућем карактеру у коначној ниски.

2.5 Мутација

Мутација има изузетно значајну улогу у раду ГА. Пошто је у уводу речено да оно што ГА ради јесте у суштини тражење минимума (или максимума) неке функције, јавља се опасност да алгоритам заврши у неком локалном минимуму (максимуму) и не пронађе оптимално решење. Такве ситуације се настоје спречити мутацијом. Мутација, као и у природи, подразумева насумичну промену генетског кода. Пошто се она у природи ретко дешава, обично се ГА праве тако да мутира $\leq 5\%$ популације. Насумична промена генетског кода може довести до тога да се јединка помери од локалне екстремне тачке чиме се повећава шанса за проналаском оне глобалне.

2.6 Добијена решења

Алгоритму је дато да пронађе ниску ‘Земунска Гимназија’. Решења алгоритма су варијирала због насумичне иницијализације почетне популације, али су се минимално разликовала од тражене ниске. Ове минималне разлике је могуће уклонити бољом дефиницијом функције грешке као и бољим алгоритмом укрштања. У слици X. приказан је поступак тражења.

3 Тестирање генетског алгорита

У овом делу рада биће укратко објашњен начин рада генетског алгоритма који проналази максимум неке задате функције (минимум функције се може наћи истим поступком, уз измене функције добротe). Формално, треба пронаћи реалан број x за који важи $x \in [a, b]$ и $f(x) = \max$ за неку задату функцију $f(x)$ која је дефинисана на интервалу $[a, b]$. У одељку 4. биће приказано како перформансе овог алгоритма зависе од различитих вредности глобалних параметара.

3.1 Грејево кодирање и иницијализација популације

Избор генетског кода је у овом случају мало тежи. Уколико би смо покушали да као генетски код просто користимо онај реални број који јединка представља, било би поприлично тешко имплементирати укрштање и мутацију. Испоставља се да се ово може заобићи ако користимо бинарне бројеве. Бинарни бројеви се записују се помоћу цифара 0 и 1 и врло су погодни за наведене операције због битовских операција (and, or, not, shift). Ако за генетски код искористимо бинарни број, лако конструишемо одговарајући реалан број који је њиме представљен:

$$x = a + \frac{t}{2^n - 1}(b - a) \quad (1)$$

где је t бинарни генетски код а n његова дужина. Једини проблем који се јавља са оваким генетским кодом је следећи: нека имамо у популацији јединку са генетским кодом 01111 а нека се максимум налази у тачки која је

представљена генетским кодом 10000. Дакле, у популацији постоји јединка која је јако близу коначном решењу, али због начина репрезентације бинарних бројева разликују се у чак 4 бита. Ово је проблем јер бројеви који су јако близу (готово један поред другог) имају доста различите репрезентације. То може изазвати лоше укрштање јединки. Као решење можемо искористити Грејево кодирање, јер се суседни бројеви представљени овим кодирањем разликују у само једном биту. Бинарни број $t = t_1 t_2 \dots t_n$ можемо превести у Грејев код $g = g_1 g_2 \dots g_n$ на следећи начин:

$$\begin{aligned} g_k &= b_k \oplus b_{k+1} & k &= 0, 1, \dots, n-1 \\ g_n &= b_n & k &= n \end{aligned}$$

Дакле, генетски код ће бити бинарни бројеви који су записани Грејевим кодирањем. Одговарајући реалан број на основу генетског кода добијамо тако што генетски код декодирамо из Грејевог кодирања у бинарни број а онда тај број убацимо у једначину 1.

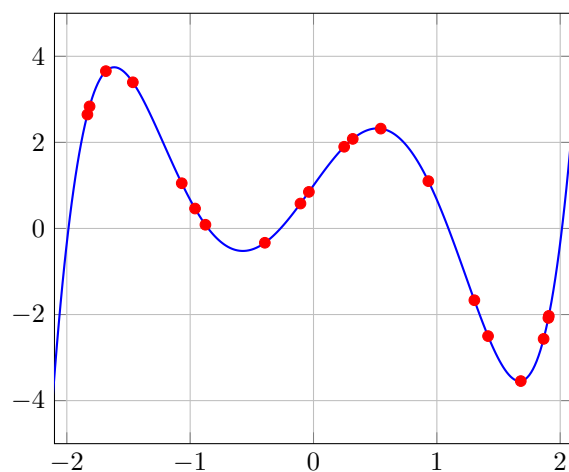
3.2 Селекција, укрштање и мутација

Селекција је извршена на два начина. Први начин је једноставна селекција која је објашњена у одељку 2.3. У овом случају користимо функцију добротe и то ће бити управо та функција чији максимум тражимо плус нека константа. Та константа служи да функција добротe нема негативне вредности (то је услов за алгоритам селекције). Ако је функција која нам је задата позитивна на интервалу $[a, b]$ онда је вредност ове константе 0. У супротном, она добија следећу вредност $CONST = |\min(f(x_1), f(x_2), \dots, f(x_n))|$. Други начин је елиминацијска селекција. Она се од једноставне разликује у томе што се прво обрише M најгорих јединки из популације (M је глобални параметар), а након тога се укрштањем оних преосталих додаје M нових јединки. Избор јединки за елиминацију се такође врши Roulette wheel алгоритмом. Функција добротe остаје иста, само што сада нема потребе уводити константу која осигурава да је вредност функције позитивна.

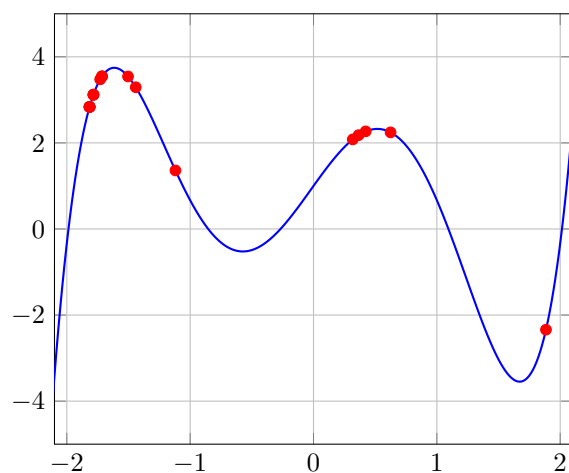
Укрштање два генетска кода се врши бит по бит, сваки бит има вероватноћу p да добије вредност од прогв родитеља а $1 - p$ да добије вредност другог. У овом случају је коришћено $p = 0.5$. Мутација је поново одређена глобалним параметром PM и функционише тако што се насумично промени један бит у хромозому. Коришћено је $PM = 2\%$.

3.3 Приказ рада алгоритма

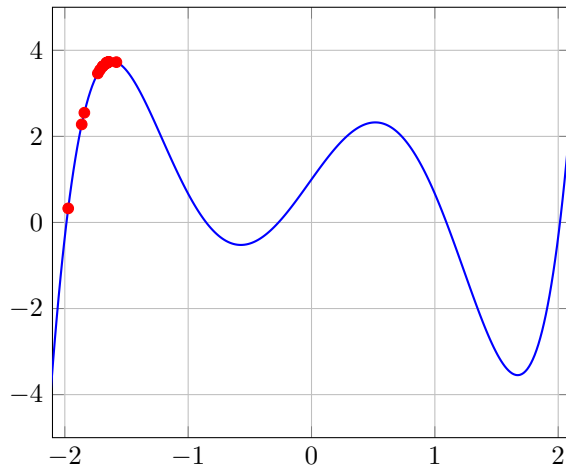
Како бисмо приказали рад алгоритма, приказаћемо претрагу графички. Графици су добијени на основу података који су се нашли у популацији у неким генерацијама. Коришћена је популације величине 20 како графици не би били претрпани.



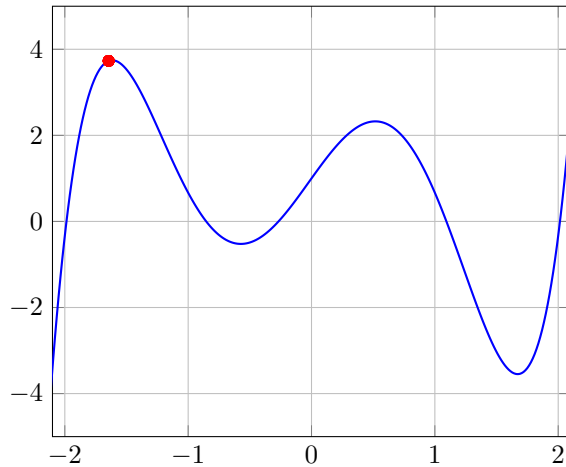
Слика 1: Прва генерација



Слика 2: Друга генерација



Слика 3: Осма генерација



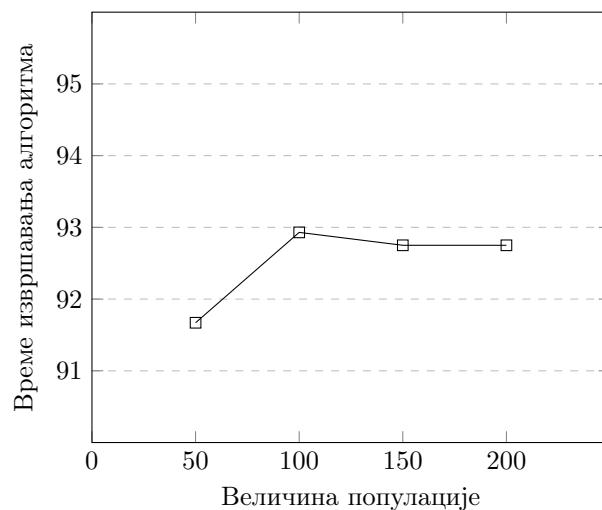
Слика 4: Последња генерација

4 Резултати тестирања

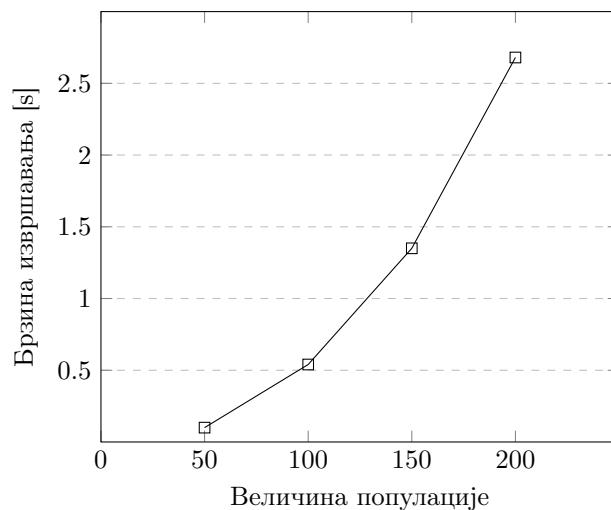
Глобални параметри знатно утичу на извршавање ГА. Нпр. уколико је популација сувише мала, може се десити да алгоритам не дође до тачног решења, а са друге стране ако је сувише велика извршавање алгоритма може трајати сувише дуго. Зато су испитане перформансе алгоритма наведеног у одељку 3. у зависности од следећих параметара: величина популације, број генерација, проценат популације који мутира. Испитана је тачност решења које проналази алгоритам (изражена у процентима) и брзина извршавања.

Такође, поређене су две врсте селекције: једноставна и елиминацијска, како би се видело да ли нека од њих има боље перформансе. Ова тестирања су извршена над функцијом $f(x) = x^{\sin x}$. Резултати су приказани у следећим графицима. Прва два графика се односе на алгоритам који користи једноставну селекцију а друга два на алгоритам који користи елиминацијску селекцију.

Резултат алгоритма у зависности од величине популације



Брзина извршавања алгоритма у зависности од величине популације



Поред тестирања перформанси алгоритма у зависности од вредности глобалних параметара, испитане су и перформансе алгоритма над различи-

тим функцијама. Конкретно, искоришћене су следеће функције:

$$f_1(x) = x^5 - 5x^3 + \frac{1}{3}x^2 + 4x + 1 \quad f_2(x) = x^{\sin x} \quad f_3(x) = x \cos(\tan x)$$

Пошто у раду ГА има доста насумичности, у табелама испод су приказане средње вредности одговарајућих метрика након 100 итерација програма. У првој табели стоје резултати добијени користећи једноставну селекцију а они у другој користећи елиминацијску селекцију. Разлика решења представља тачност алгоритама, јер што је она мања то је алгоритам бољи. Сви резултати су добијени користећи популацију од 200 јединки, од 2% којих мутира. Број генерација је износио 150.

Функција	Право решење	Добијено решење	Разлика решења [%]	Брзина [s]
$f_1(x)$	-1.6195	-1.6192	0.2	2.61
$f_2(x)$	95.821	92.240	3.1	2.27
$f_3(x)$	6.00	5.98	0.2	2.81

Табела 1: Резултати једноставне селекције

Функција	Право решење	Добијено решење	Разлика решења [%]	Брзина [s]
$f_1(x)$	-1.6195	-1.6159	0.2	3.96
$f_2(x)$	95.821	95.01	0.85	15.89
$f_3(x)$	6.00	5.99	0.17	16.25

Табела 2: Резултати једноставне селекције

5 Закључак

Генетски алгоритми представљају леп пример хеуристичког програмирања, јер користе поприлично нестандардан начин за решавање проблема. Моделовање еволуције може довести до решења неких проблема које је тешко решити класичним методама. Испоставља се да неке проблеме можемо решити моделовањем разних појава из природе и постоје још неке хеуристичке методе које функционишу на овакав начин (нпр. оптимизација колонијом мрава (*Ant colony optimization*)). Постоје и проблеми за које генетски алгоритам не може наћи добро решење и то су углавном они проблеми за које је тешко осмислити функцију грешке, односно доброте. Међутим, велики број проблема се може прилагодити генетском алгоритму, због чега је ова хеуристичка метода постала изузетно популарна.

6 Литература

- [1] Генетски алгоритам, Марин Голуб, 2004
- [2] Genetic Algorithms: An Overview, Melanie Mitchell, 1995

[3]