Матурски рад из Програмирања

Хеуристичко програмирање

Ученик:
Бокор Иван IV_{12}

Ментор:
Корађ Вукман

Земун, мај 2023.

Садржај

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 осмишњени 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 ће бити предсдтављени генетски алгоритми који припадају тој категорији.

Генетски алгоритам (у даљем тексту ГА) представља хеуристичку методу оптимизације[1] који моделује механизме еволуције у природи. Идеја је да почети од неког скупа насумичних решења и онда еволуцијом тих решења доћи до оптималног. Еволуција се одвија помоћу два механизма: **укрштања** и **мутације**. Укрштањем два решења добија се ново потенцијално решење проблема а мутацијом се насумично мења решење чиме се подиже степен различитости у целом скупу решења. Процес бирања решења која ће се укрстити се назива **селекција**. Пошто је терминологија готово иста као она која се користи у биологији, скуп потенцијалних решења неког проблема се назива **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 механизама (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 се осмисли нека функција која ће евалуирати потенцијална решења проблема до којих генетски алгоритам буде дошао, оптимално решење ће се управо наћи у максимуму или минимуму те функције (у зависности од проблема који се решава). Таква функција се назива **функција доброте** - $fitness(x)$ или **функција грешке** - $cost(x)$. Она на неки начин евалуира решења, тј. сваком решењу придружује неки реалан број којим се мери колико је дато решење "добро"или "лоше"(отуда и називи функција доброте тј. грешке). 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 се дефинише глобални парамтер 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 се може 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 150 генерација.

2.2 Репрезентација јединки

Као што су људи једнозначно одређени ДНК молекулима, 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 структре са којима рачунари лако манипулишу. У овом слуајју, 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 родтиељи - 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 доброте или функција грешке. У овом случају користиће се функција грешке. Она је дефинисана на следећи начин:

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

где је s потенцијално решење а $GOAL$ ниска коју алгоритам треба да достигне. Овиме се постиже да јединке које представљају ниске чији су карактери ближи карактерима коначне ниске имају (по ASCII вредности) мању вредност функцију грешке (напомена: оваква дефиниција функције грешке није добра, али ће послужити за ову демонстрацију). 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 (точак је подељен на делове онако како се обично сече пица). Површина сваког дела је пропорционална вероватноћи да та јединка буде изабрана за родтиеља. 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 се приликом укрштања направи нови низ у који ће се убацивати 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 $i \leftarrow 1...POP\_SIZE$ **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 изаберу, потребно је извршити укрштање генетских кодова родтиеља 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 родтиеља и у генетски код детета се додаје онај који је ближи одговарајућем карактеру у коначној ниски. Иако овакав алгоритам може послужити за демонстрацију

ГА, постоји бољи алгоритам укрштања. Међутим, он није искоришћен због недостатка времена.

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 '*ЗемунскаГимназија*'. Решења алгоритма су варирала 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. Испод је приказан поступак доласка до решења. Из сваке генерације је приказана најбоља јединка.

Генерација	**Јединка**
1	?llqirVXIklmVPfpX
2	allxirVgIilmVwfdc
3	ZalukrkbOohocyhfb
4	Z'lvmtlaHhmn'vlib
5	ZeoumskaIhonayhja
...	...
199	ZemunvkaGimnazija
200	ZemunskaGimnazija

3 Тестирање генетског алгортима

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 доброте). 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 бисмо 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:

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

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

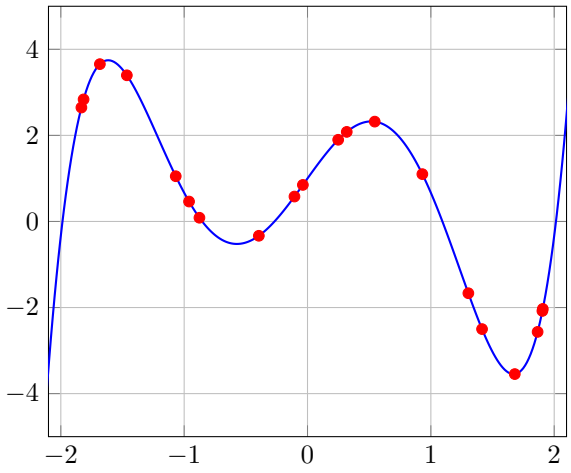
Селекција је извршена на два начина: уз помоћ једноставне селекције и елиминацијске селекције. Једонставна селекција је објашњена у одељку 2.3. У овом случају је коришћена функција доброте и то ће бити управо та функција чији максимум се тражи плус нека константа. Та константа служи да функција доброте нема негативне вредности (то је услов за алгоритам селекције). Ако је функција која је задата позитивна на интервалу $[a, b]$ онда је вредност ове константе 0. У супротном, она добија следећу вредност $CONST = |\min(f(x_1), f(x_2), ..., f(x_n))|$. елиминацијска селекција 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 доброте 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 провг родитеља а $1 - p$ да добије вредност другог. У овом случају је коришћено $p = 0.5$. Мутација је поново одређена глобалним парамтером PM и функционише тако што се насумично промени један бит у хромозому. Коришћено је $PM = 2\%$. Псеудокод алгоритма укрштања је дат испод.

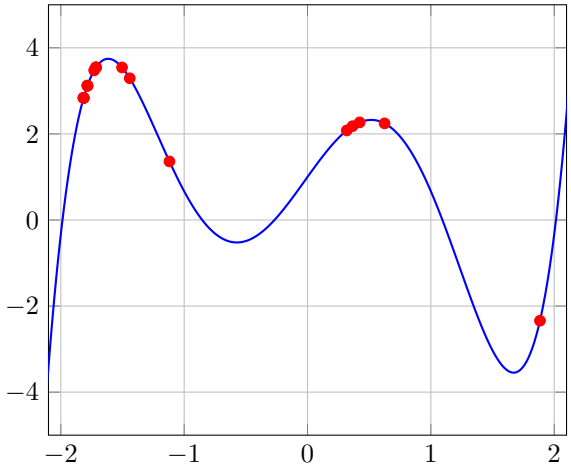
```
p1 ← select()
p2 ← select()
while p1 = p2 do
    p2 ← select()
end while
for i ← 1...N do
    if random(0, 1) <= 0.5 then
        uzmi i-ti bit od prvog
    else
        uzmi i-ti bit od drugog
    end if
end for
```

3.3 Приказ рада алгоритма

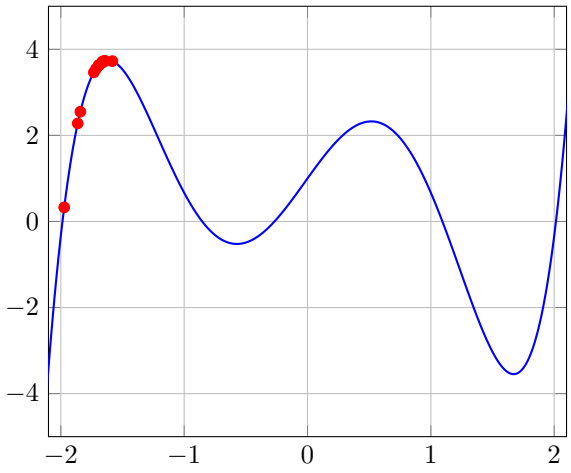
Рад алгоритма је најбоље приказати графички. Испод се налазе графици који приказују како алгоритам тражи коначно решење. Графици су добијени на основу података који су се нашли у популацији у неким генерацијама. Коришћена је популација величине 20 како графици не би били претрпани.



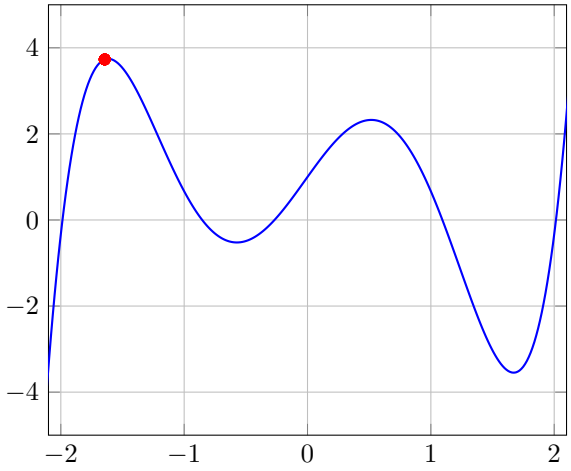
Слика 1: Прва генерација



Слика 2: Друга генерација



Слика 3: Осма генерација

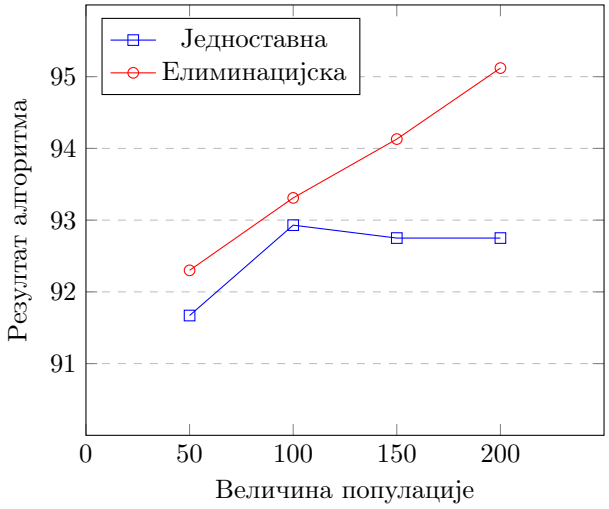


Слика 4: Последња генерација

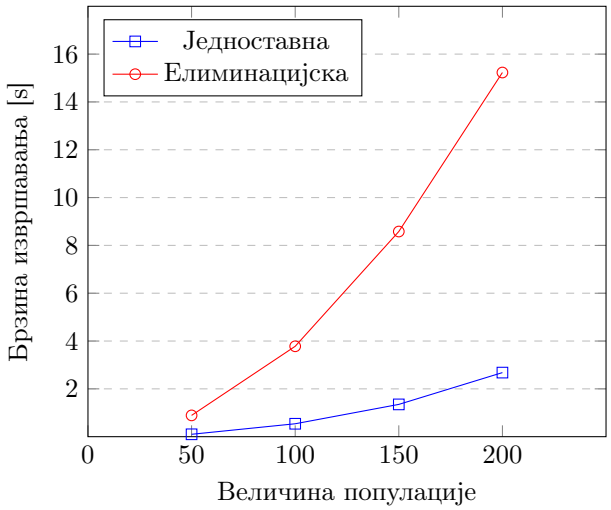
4 Резултати тестирања

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 величина популације и броја генерација. Перформансе су приакзане преко резултата алгоритма и брзине 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 $f(x) = x^{sinx}$ у интервалу [0, 100]. Приликом тестирања перформанси у зависности од величине популације алгоритам је генерисао 150 генерација, а када је тестирано у зависности од броја генерација, величина популације је била 200. 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. Сви резултати су добијени као средње вредности након 100 извршавања.

Резултат алгоритма у зависности од величине популације

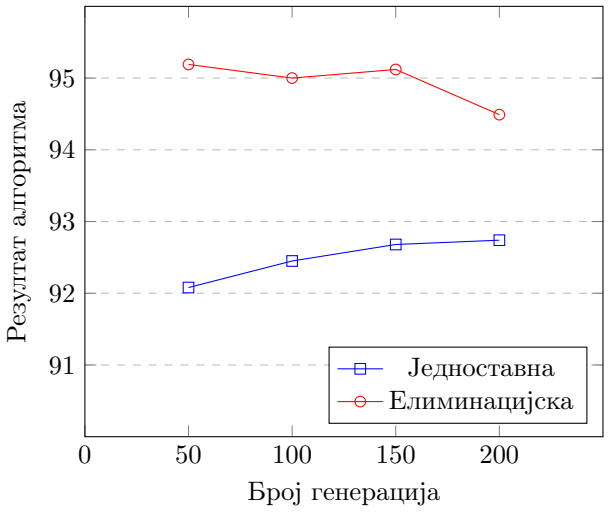


Брзина извршавања алгоритма у зависности од величине популације

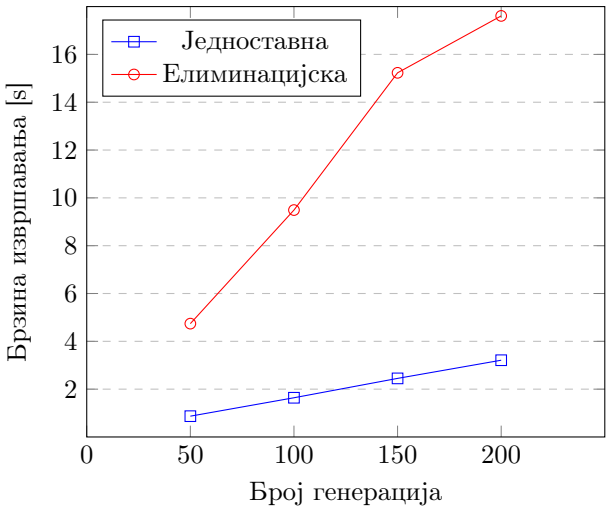


Види се да величина популације сразмерно утиче на брзину извршавања, што је и очекивано. Што се тиче решења, величина популације не утиче толико драстично. Једноставна селекција се извршава брже од елиминацијске. За мање популације се јавља проблем да доста јединки постане исто, што умањује разноликост решења.

Резултат алгоритма у зависности од броја генерација



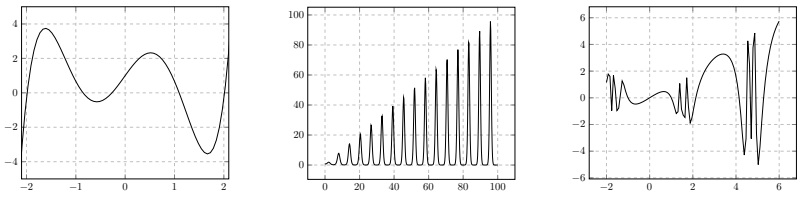
Брзина извршавања алгоритма у зависности од броја генерација



Уколико је број генерација мали, може се десити да алгоритам "не стигне"да изгенерише оптимално решење. Ово је надомешћено тиме што је број јединки био поприлично велик и због тога су увек неке јединке биле близу оптималног решења.

Поред тестирања перформанси алгоритма у зависности од вредности глобалних параметара, испитане су и перформансе алгоритма над различитим функцијама. Искоришћене су следеће функције:

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



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 алгоритма, 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	0.22	16.78
$f_2(x)$	95.821	95.12	0.73	15.23
$f_3(x)$	6.00	5.99	0.17	16.25

Табела 2: Резултати елиминацијске селекције

Једноставна селекција се извршава знатно брже од елиминацијске. Обе селекције дају слична решења и не може се једна издвојити као боља од друге по овом параметру. Генерално, ГА добро проналази тражена решења. Разлог зашто није било већих разлика у резултату алгоритма када су коришћење различите величине популације и број генерација је тај што су интервали над којима је рађено превише мали за толики број функција. Када се на мали интервал стави пуно тачака, неминовно је да ће неке завршити близу тачке у којој се постиже максимум. Приликом тумачења резултата такође треба узети у обзир да је алгоритам испрограмиран у C++, језику који нема најбољу подршку за генерисање насумичних бројева. Врло је вероватно да би резултати били још бољи да је коришћен језик попут Python-а. Такође, бољи резултати се могу постићи ако се посвети већа пажња дефинисању оптималних алгоритама за селекцију, укрштање и мутацију.

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] Марин Голуб, Генетски алгоритам ,2004
[2] Melanie Mitchell, "Genetic Algorithms: An Overview", 1995
[3] Vijini Mallawaarachchi, "Introduction to Genetic Algorithms — Including Example Code", `https://towardsdatascience.com/introduction-to-genetic-/algorithms-including-example-code-e396e98d8bf3`, приступљено 23.5.2023.